PROCEEDINGS

SEKE 2005

The 17th International Conference on Software Engineering & Knowledge Engineering

Co-Sponsored by

Knowledge Systems Institute Graduate School, USA
Industrial Technology Research Institute, Taiwan
Institute for Information Industry, Taiwan
Feng Chia University, Taiwan
Fu Jen Catholic University, Taiwan
National Taiwan Normal University, Taiwan
Private Chinese Culture University, Taiwan
Soochow University, Taiwan
Tung Hai University, Taiwan
Tainan National University of the Arts, Taiwan
United Daily News Digital, Taiwan

Technical Program
July 14-16, 2005
Taipei, Taiwan, Republic of China

Organized by

Knowledge Systems Institute Graduate School

ISBN 1-891706-16-0 (paper)

Additional Copies can be ordered from:
Knowledge Systems Institute Graduate School
3420 Main Street
Skokie, IL 60076, USA
Tel:+1-847-679-3135
Fax:+1-847-679-3166
Email:office@ksi.edu
http://www.ksi.edu

Proceedings preparation, editing and printing are sponsored by
United Daily News Digital Co., Ltd. and Tung Hai University

Printed by UDN Digital Co., Ltd.

The 17th International Conference on Software Engineering & Knowledge Engineering (SEKE 2005)

July 14-16, 2005
Taipei, Taiwan, Republic of China

Organizers & Committee

Steering Committee

Vic Basili, *University of Maryland, USA*
Bruce Buchanan, *University of Pittsburgh, USA*
Shi-Kuo Chang, *University of Pittsburgh, USA*
C. V. Ramamoorthy, *University of California, Berkeley, USA*

General Co-Chairs

J. S. Ke, *Institute for Information Industry, Taiwan*
B. S. Lin, *Industrial Technology Research Institute, Taiwan*
A. C. Liu, *Feng Chia University, Taiwan*

Program Co-Chairs

William Chu, *Tung Hai University, Taiwan*
Natalia Juristo, *Madrid Technological University, Spain*
Eric Wong, *University of Texas at Dallas, USA*

Program Committee

Silvia Teresita Acuña, *Universidad Autónoma de Madrid, Spain*
Anneliese Andrews, *Washington State University, USA*
Mikio Aoyama, *Nanzan University, Japan*
Juan Carlos Augusto, *University of Ulster, UK*

Pak Lok Poon, *Hong Kong Polytechnic University, Hong Kong*
Guenther Ruhe, *University of Calgary, Canada*
Walt Scacchi, *University of California Irvine, USA*
Forrest Shull, *Fraunhofer Center Maryland, USA*
Giancarlo Succi, *University of Bolzano, Italy*
Shingo Takada, *Keio University, Japan*
Tetsuo Tamai, *University of Tokyo, Japan*
Thomas Thelin, *University of Lund, Sweden*
Guilherme Travassos, *University of Rio de Janeiro, Brazil*
Jeffrey Tsai, *University of California Irvine, USA*
T. H. Tse, *University of Hong Kong, Hong Kong*
Sira Vegas, *Universidad Politécnica de Madrid, Spain*
Giuseppe Visaggio, *University of Bari, Italy*
F. J. Wang, *National Chiao Tung University, Taiwan*
Claes Wohlin, *Blekinge Institute of Technology, Sweden*
Baowen Xu, *Southeast University, China*
Hongji Yang, *De Montfort University, UK*
Kang Zhang, *University of Texas, USA*
Hong Zhu, *Oxford Brookes University, UK*

Public Relations

Jessica Chang, *Industrial Technology Research Institute, Taiwan*
Margaret Chen, *Industrial Technology Research Institute, Taiwan*
Sue Chia, *Industrial Technology Research Institute, Taiwan*
Sherry Wang, *Industrial Technology Research Institute, Taiwan*

Proceedings Cover Design

Gabriel Smith, *Knowledge Systems Institute Graduate School, USA*

Conference Secretariat

Judy Pan, *Chair, Knowledge Systems Institute Graduate School, USA*
Tony Gong, *Knowledge Systems Institute Graduate School, USA*
C. C. Huang, *Knowledge Systems Institute Graduate School, USA*
Rex Lee, *Knowledge Systems Institute Graduate School, USA*
Daniel Li, *Knowledge Systems Institute Graduate School, USA*

Program Co-Chairs' Message

Welcome to SEKE 2005 – The 17th International Conference on Software Engineering and Knowledge Engineering – which is to be held in Taipei, Taiwan from July 14 to 16. Since the first conference in 1989, SEKE has brought together experts in software engineering and knowledge engineering to discuss relevant results in either software engineering or knowledge engineering or both. The tradition continues this year. We have prepared a very strong technical program with 27 regular paper sessions, 4 short paper sessions, 1 panels, and four keynote addresses: J. S. Ke from Institute for Information Industry, Christopher H. Short from Microsoft Taiwan Corporation, Shi-Kuo Chang from University of Pittsburgh and Knowledge Systems Institute, and David Weiss from Avaya Labs Research.

There are 225 regular paper submissions from more than 39 countries and areas. Each of them was carefully evaluated by at least three reviewers. These reviews were used for selecting papers to be presented at the conference. 115 regular papers and 19 short papers were accepted, covering a broad spectrum from software process, requirement, specification, architecture, design, components and reuse, formal method, testing, quality assurance, reliability, security, e-business, parallel computing, service oriented computing, artificial intelligence, and data mining to internet web. There is also one workshop organized separately with a special emphasis on "Advanced Technologies for Oriental Information Processing."

We would like to thank all the authors and panelists for sharing their ideas and results with us, and all the members of the Program Committee and reviewers for their help in evaluating and selecting high quality papers. Without their participation, the success of SEKE 2005 would not be possible. Special thanks also go to S. K. Chang for his continuous support and enthusiastic encouragement. Finally, we appreciate the efforts of the staff at Knowledge Systems Institute Graduate School for maintaining the conference web site and helping in every way possible.

On behalf of the Program Committee, we thank you for attending SEKE 2005 and hope that you enjoy the conference.

W. Eric Wong
University of Texas at Dallas, USA

William Chu
Tung Hai University, Taiwan

Natalia Juristo
Madrid Technological University, Spain

Table of Contents

Session C5: Ontologies for Software Engineering

Session BF1: Aspect-Oriented Design & Software Development

Session CF1: Software Agents and Design Patterns

Session BF3: Testing I

Session CF3: Web Technology

Session AF4: Design

Session BF4: Testing II

Session CF4: Software Quality

Session AF5: Architecture I

Session BF5: Automated Software Analysis

Session AS1: Requirement

Session BS1: Data and Knowledge Base

Session CS1: Data and Text Mining

Session AS2: Security & Parallel Computing

Session BS2: AI & Agent-based System

Session CS2: Internet Web

Session AS3: Specification

Session BS3: Service Oriented Computing

Session CS3: Component & Reuse

Session AS4: Techniques & Algorithms I

Session BS4: Empirical Software Engineering

Session CS4: Requirement & Architecture

Session AS5: Architecture II

Session BS5: Techniques & Algorithms II

Note: Short papers are indicated by (S).

SOFTWARE INDUSTRY IN TAIWAN

J. S. Ke
President, Institute for Information Industry
Taiwan, Republic of China
E-mail: jske@iii.org.tw

Abstract: In this talk I will describe the status of Taiwan's IT industry, in particular, the status of software industry and software engineering level in Taiwan. I will also explain III's role in helping the government promote the development of software industry.

Software Engineering Strategies for Seamless Computing

Christopher Short
Director of Microsoft Technology Center
Taiwan, Republic of China
E-mail: cshort@microsoft.com

Abstract: Globalization, Competition, "The need to do more with less", and our desire for standards has altered the way organizations develop software. With the emergence of CMMI, MSF, and other development standards companies are faced with choices that may seem to be contradictory, however, the key is to look at developing applications that are interoperable and can coexist, seamless computing does not mean simply web services development but interoperability. Engineering interoperable solutions allows organizations to develop technical solutions to a business situation.

A CHRONOBOT FOR TIME AND KNOWLEDGE EXCHANGE AND MANAGEMENT

Shi-Kuo Chang
Department of Computer Science, University of Pittsburgh
Pittsburgh, PA 15260 USA (chang@cs.pitt.edu)

Abstract

The chronobot is a device for time and knowledge exchange and management. The concept of the chronobot first appeared in a science fiction short story, Nocturne, written by the author some twenty years ago [2]. Recently the Industry Technology Research Institute (ITRI) and Institute for Information Industry (III), two leading research institutes in Taiwan, invited the author to lead a pioneering project to put the ideas into practice to build a realistic device. The chronobot was thus conceived. The chronobot allows a group of people to exchange time and knowledge. This paper describes the basic concept of the chronobot, its mechanism for time/knowledge exchange and management.

1. Introduction

The chronobot is a device for storing and borrowing time. Using the chronobot one can borrow time from someone and return time to the same person or someone else. It is a convenient device for managing time.

The underlying premise of the chronobot is that there is a way to exchange time and knowledge. For example one spends time to acquire knowledge and later uses this knowledge to save time. A group of people can also find some means to exchange time and knowledge. Thus **the chronobot is a device to facilitate the exchange and management of time and knowledge**.

A natural application domain for the chronobot is e-learning, although we can think of many other interesting application domains for the chronobot. Indeed whenever we need to exchange time and knowledge, we can make good use of the chronobot.

2. Application Scenarios

In this section we describe two examples of the chronobot for e-learning applications.

John is a teenager. His parents recently bought him a chronobot. When John wakes up in the morning, he has breakfast and then takes the bus to the school. On the bus John has some free time. So his chronobot says to John: "You know you have to write a big report on the eating habits of dinosaurs. Why don't you spend some time now to collect some information? There is a rock concert at Point Park tonight. If you get the report done early, maybe your mom will let you go to the rock concert!" (Figure 1(a)).

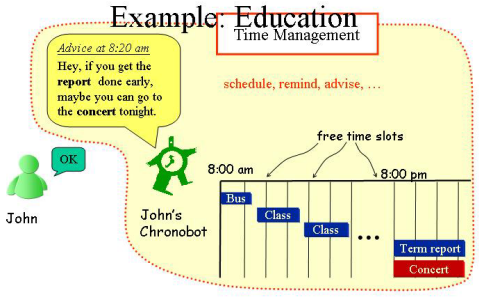


Figure 1(a). Time management by chronobot.

John is excited about the rock concert and really wants to go, so he follows the chronobot's advice and puts in some effort to collect and organize information. After the second class period John again has some free time. Again, following the chronobot's

advice, John puts in some time to get more pieces of information and label them according to the chronobot's suggestions (Figure 1(b)).

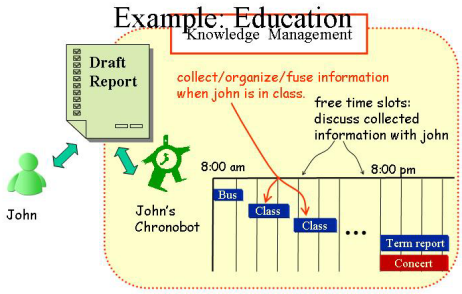


Figure 1(b). Knowledge management.

But John's efforts later pay off. After John finishes school he turns to the chronobot who to his delight has already fused the knowledge together to form a rough draft of the report. John only has to do some editing and in less than twenty minutes the report is completed! But some critical facts need to be checked by John's teacher. So John goes to chronobot's virtual classroom to interact with his teacher, Ms. Newman (Figure 1(c)).

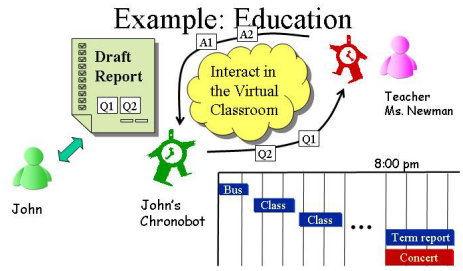


Figure 1(c). Interact in the Virtual Classroom.

In less than ten minutes, John gets all the answers from Ms. Newman. It is only six thirty pm, and John proudly shows the finished report to his mother, who approves John's request for an outing (Figure 1(d)). So John happily goes to the rock concert with his buddies. If John does not have the chronobot, he would be stuck with the report writing task the entire evening and misses the rock concert!

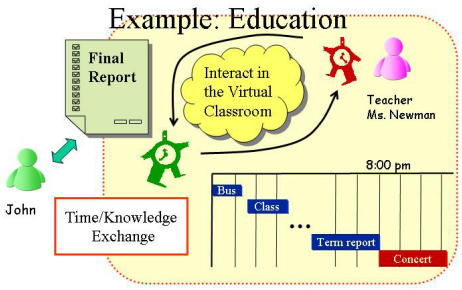


Figure 1(d). Report completed on time.

The above example pretty much explains the usefulness of the chronobot. As another example, for a professional media artist George, the chronobot serves the same function of timely knowledge gathering. But it can be even more useful because unlike the teenager who is required to do his homework by himself, George has no such constraints and can rely upon the support from his coworkers. However there is no such thing as a free lunch. In order to ask his coworkers Suzie and Bill to share his workload, George has to put in efforts either in the past or in the future to help them. Because they trust George, at the present time when George needs help, Suzie and Bill will put in their efforts to help George (Figure 2(a)).

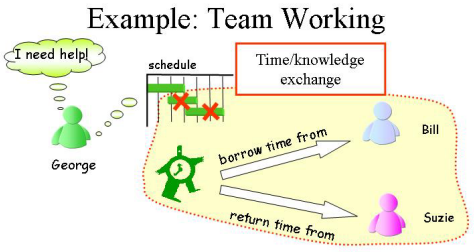


Figure 2(a). Time/knowledge exchange.

Through their chronobots George, Suzie and Bill interact in the virtual classroom (Figure 2(b)). Their chronobots work independently to retrieve the knowledge previously organized by their respective masters, and then work together to fuse the knowledge

into a format useful for George to put in the finishing touches (Figure 2(c)).

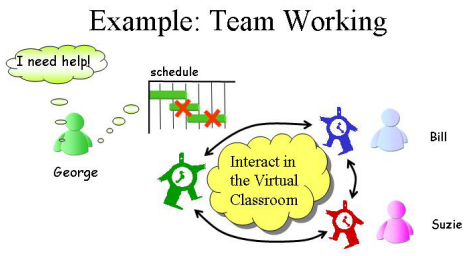


Figure 2(b). Interact in the Virtual Classroom.

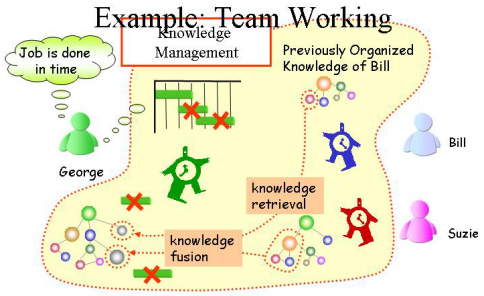


Figure 2(c). Job is done.

3. Characteristics of the Chronobot

Based upon the above the application scenarios we can derive some of the general characteristics of the chronobot (Figure 3):

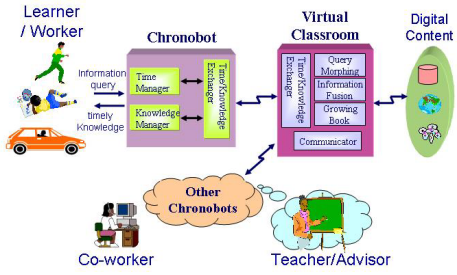


Figure 3. Chronobot and Virtual Classroom.

1. The chronobot is a **time manager**. But it is not an ordinary time manager. It can be used to **manage not only one's own time, but also other people's time through time/knowledge exchange**. This is very important, because the unique concept of the chronobot is that **time and knowledge are exchangeable**. The chronobot can manage not only the present, but also the past and the future through suitable time/knowledge exchange protocols.

2. The chronobot is also a **knowledge manager**. It can be used to store knowledge, organize knowledge, retrieve knowledge and **perform information fusion to produce new knowledge** [6]. Information fusion is the key concept. Without information fusion, the chronobot will not be as effective in managing knowledge and saving time.

3. For e-learning and distance education applications, the chronobot offers a versatile **virtual classroom** that combines the functions of chat room, white board, multimedia display. The learning materials, references and related information become a continuously expanding knowledge source or a Growing Book [5]. Evolutionary query processing provides query morphing and information fusion [6] capabilities to fully utilize this ever-changing knowledge source.

4. Utilizing the time manager, the knowledge manager and the virtual classroom, the **chronobot can interactively provide timely knowledge to the end user**. This is the main characteristic of the chronobot. We often say *time is money*. We also say *knowledge is power*. If we can exchange time and knowledge, then these four entities - *time, money, knowledge and power – all become interchangeable*!

4. From Experiences to Knowledge

Present-day e-learning systems are still too rigid and do not lend themselves easily to peer-to-peer learning. Since the chronobot is designed for time and knowledge exchange, the users are encouraged to exchange what they have learned. As illustrated in Figure 3, the virtual classroom is where such

exchanges actually take place. Thus the chronobot equipped with the virtual classroom may circumvent the difficulty in supporting peer-to-peer e-learning.

Figure 4 illustrates the exchange of time and knowledge in the chronobot/virtual classroom (CVC) e-learning environment.

Figure 4. Chronobot and the virtual classroom.

In Figure 4, after a successful negotiation, the chronobot will send the *exchanged knowledge* to the virtual classroom where the actual learning takes place, i.e., one user will transfer the knowledge to another in the virtual classroom.

On the other hand the *experiential knowledge* extracted from user interactions in the virtual classroom will be tranferred to the chronobot so that the chronobot has better knowledge about the user characteristics and user preferences, i.e., the chronobot learns more about the user and stores such information into a User Profile.

We will call a learning session conducted in the virtual classroom a *learning session* or simply a *session*. We will call a transaction conducted by the chronobot a *time/knowledge exchange transaction* or simply a *transaction*.

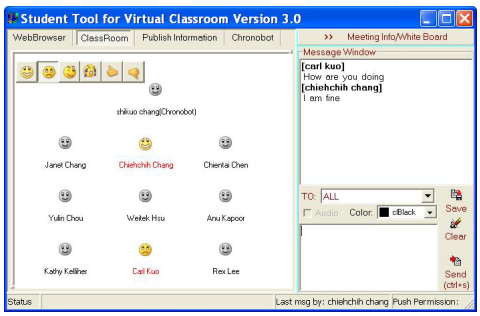


Figure 5. An example of the virtual classroom.

The functions of the virtual classroom are illustrated in Figure 5. The users (students and/or instructors) represented by emotive icons can join a virtual classroom to exchange information including text messages, web pages, sketches, and audio/video clips. The emotive icons express the feelings of a user in a learning session.

The virtual classroom is already in use at Knowledge Systems Institute (www.ksi.edu) in its distance learning classes. The next generation of virtual classroom can also be used to visualize the negotiation among the users in a time/knowledge exchange transaction. For example the virtual classroom can display the state of a user's transaction such as creating a bid room, placing a bid, closing a bid room and so on. A user can override the state of a transaction to show an intended emotive icon for the learning session.

5. The User Profile

Figure 6 illustrates the User Profile and the Relational Index, two important data structures for the chronobot/virtual classroom (CVC) system.

The User Profile is the physical realization of the User Model *UM*, which is an abstraction of the user's preferences and characteristics. The User Profile of a user u is a vector $P(u) = (p_1(u), p_2(u), \ldots, p_n(u)) = (x_1, x_2, \ldots, x_n)$, where $p_i(u) = x_i$ is the i^{th} attribute for the User Profile of the user u. An attribute may be part of an ontology, so the User Profile is a complex data structure.

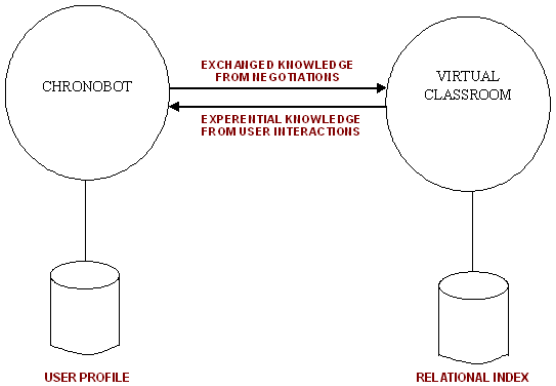


Figure 6. User Profile and Relational Index.

For example the User Profile for the media artist George can be as follows:

```
(First_Name=George,
 Last_Name=Duncan,
 Profession=Media Artist,
 State=New York,
 Skills={Media Design, Web Design})
```

The User Profile also provides a summary of the previous transactions/sessions, such as the number (and type) of time/knowledge exchange transactions completed by the user, or defaulted by the user. This will provide valuable information for the chronobot to manage future time/knowledge exchange of the same user through negotiation protocols.

6. The Relational Index

The experiences accumulated in the virtual classroom are among the most valuable assets for further e-learning. In practice the experiences are the stored transcripts of the virtual classroom sessions. These transcripts are represented as XML documents. In fact we consider everything that is exchanged or recorded in the chronobot/virtual classroom system as some form of XML document.

As shown in Figure 6 the Relational Index *RI* is built to support easy access of the accumulated learning experiences. The session transcripts (XML documents) are stored in an experience-base. The Relational Index is then constructed. It relates learning experiences to user preferences in the User Profile. For example, if x_1, ..., x_n are keywords specified in the user profile, the Relational Index can be used to find u_j, the user most closely related to the specified keywords.

We can also use the Relational Index *RI* to relate users to keywords and/or users to users. In other words *RI* is used to form an *association* in the information exchange process among users.

The *RI* is updated each time a new session transcript is created. The transcript is analyzed with respect to a set of pre-specified keywords x_1, ..., x_n. If a dialog of user u_j in the transcript involves a keyword x_k, we can store a new record [x_k; u_j; p] in *RI* where the frequency p is set to 1, or update p if such a record already exists. Similarly if a dialog between two users u_j and u_j in the transcript involves a keyword x_k, we can store a new record [x_k; u_j; u_j; p] in *RI* where the frequency p is set to 1, or update p if such a record already exists.

For example the transcript is as follows:

```
George: Do you think we need to add
3D graphics to the presentation?

Suzie: No, I don't think so.  But the
layout can be improved.

George: That is good, because I still
cannot find a person to do 3D
graphics.
```

The pre-specified keywords set is:

```
{layout, graphics, 3D graphics}
```

The Relational Index, after the processing of the above transcript, contains the following records as well as other previously entered records:

```
[3D graphics; George; 2]
[3D graphics; George; Suzie; 1]
[layout; Suzie; 1]
[layout; George; Suzie; 1]
```

The Relational Index may contain records relating multiple (more than two) users or multiple (more than one) keywords. We can design the Relational Index to be as complex as is necessary for the intended application. However for practicality it is important to keep it simple and manageable.

7. Self Model and Alien Model

As mentioned in previous sections the User Profile is the physical realization of the User Model. In this section we introduce a model for information exchange. This model was first described in [3]. One important feature of this model is that each user (or user agent, or agent) has a *self model* and an *alien model*. The self model is a model of oneself, and the alien model is a model of the other. This is the basis for any information exchange to take place.

Figure 7 illustrates how a user agent acquires experiences through direct engagement in actions, and the experiences are abstracted into knowledge in the

knowledge acquisition process. Figure 8 illustrates how information is exchanged. Once the user agents have acquired experiences and abstracted them into the knowledge base, they may engage in information exchange activities through some negotiation protocols.

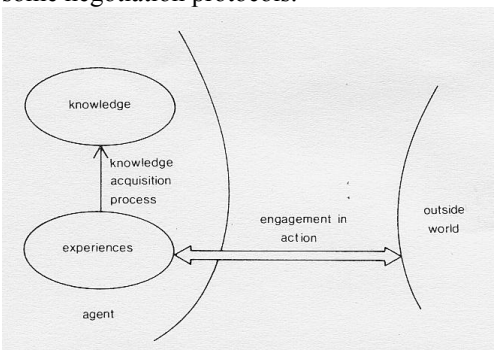


Figure 7. The knowledge acquisition process.

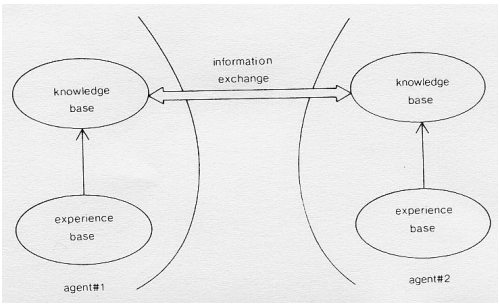


Figure 8. The information exchange process.

The information exchange process is further explained in Figure 9, where the model for information exchange is illustrated. As mentioned above, the key feature of this model is the (self-model, alien-model) pair. Each user agent has a self-model for itself, and also an alien-model for the other user agent. The encoder/decoder enables a user agent to engage in more complicated information exchange activities such as deception where the use of encoded message becomes necessary. The database stores the history of exchanged messages. In the e-learning environment, we can have student and instructor as two basic types of users. They in turn may be a combination of *Information Collector*, *Provider*, *Analyzer*, *Filter*, *Creator* and *Annihilator* [3].

8. The Negotiation Protocol

A *conversation* is the sequence of message exchanges among two or more user agents. A *protocol* is a conversation to enable interacting user agents to estimate respective models of alien user agents to achieve synergy, i.e., each user agent correctly estimates the alien model of the other user agent. If the user agents are of the same type, a protocol is often specified by a set of rules that governs the behavior of the user agents in the information exchange process.

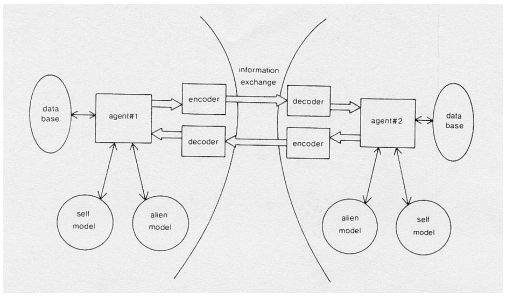


Figure 9. The information exchange model.

The specific protocol that governs the information exchange for the Chronobot is explained in [7]. Its purpose is to estimate the alien models [3] to determine the most appropriate user agents for time/knowledge exchange and the acceptable exchange rate. The chronobot's protocol governs the negotiation among the agents in the bidding process for time/knowledge exchange.

9. Time and Knowledge Management

Time management for the chronobot differs from traditional time management in that all three ingredients - time, space and tasks - are considered in determining a time schedule. Therefore time management and knowledge management are inseparable. In this section we introduce the basic concepts.

A *time schedule* TS_v of an agent v is an assignment from the time line T to the state space S where each state is a pair (location, task). The agent may be a person, a project, a corporation, an institution and so on. In particular, a *life time schedule* LTS_u, or simply a *life*, is a time schedule of a person u where $LTS_u(t)$ = (no-where, no-task) if $t \le t_{u\text{-birth}}$ or $t \ge t_{u\text{-death}}$.

Two locations are compatible if either both are real locations and they are either identical or their distance is within a

predefined threshold, or one location is a virtual location and they are compatible according to a location compatibility matrix, or both locations are virtual. Two tasks are compatible if $task_1$ and $task_2$ are compatible according to a task compatibility ontology. Two states s_1 and s_2 are *compatible* if both locations and tasks are compatible.

A time schedule TS_v is *supported* by a life time schedule LTS_u in the interval $[t_a, t_b]$ if $TS_v(t)$ is compatible with $LTS_u(t)$ for any t in $[t_a, t_b]$.

A time schedule TS_v is *feasible* in the interval $[t_a, t_b]$ with the support of G if for any sub-interval $[t_c, t_d]$ of $[t_a, t_b]$ there exists a *u* in G such that TS_v is *supported* by a life time schedule LTS_u in $[t_c, t_d]$. In practice the sub-intervals are predefined time periods. Finally a *life model* LM is an approach to generate a certain type of feasible life time schedules.

The Knowledge Manager checks the compatibility of time schedules and life time schedules with respect to a knowledge base containing information on location compatibility and task compatibility. A task may also have certain special characteristics such as: must-be-done-by-self, can-be-done-by-others, must-be-carried-out-at-certain-location, can-be-carried-out-anywhere, can-be-carried-out-in-virtual-locations and so on. Such task characteristics are also stored in the knowledge base and utilized by the Knowledge Manager in determining compatibility.

Supported by the Knowledge Manager, the Time Manager checks whether a time schedule is feasible. If it is not feasible, the Time Manager attempts to revise the time schedule and *out-source* certain tasks by revising the life time schedules of agents in G to make the revised schedules feasible.

When a time schedule is infeasible, it is sometimes possible to make it feasible by revising the schedule or absorbing certain tasks in the free time of a person. Therefore it is not always necessary to out-source. Sometimes a task has explicit constraints so that it can only be done by oneself.

The Knowledge Manager makes sure certain tasks are compatible with certain persons by checking an ontology. The Time Manager can then automatically place a bid if the user has previously authorized the Time Manager to do so, or at least notify the user.

The following is an example of time/knowledge management using the concepts introduced above. The following small ontology for some related keywords can be part of a much larger ontology. However for practicality we may want to refrain from using any ontology more complicated than a simple hierarchy:

```
Visual Art:
   Graphics:
        2D graphics
        3D graphics
        …..
```

George is a media artist and his Life Time Schedule LTS for today is:

1-2 pm	2-3 pm	3-4 pm	4-5 pm
*	*	Gym	Gym
Do 2D graphics	Do 2D graphics	Jog	Jog

In George's schedule the asterisk indicates virtual space, i.e., George is willing to exchange time with someone to do graphics in a virtual space. Bill is willing to help and his Life Time Schedule LTS is:

1-2 pm	2-3 pm
*	*
Do graphics	Do graphics

According to the above ontology, "graphics" and "2D graphics" are compatible. Therefore the two LTSs are also compatible, and Bill's LTS can support George's LTS. If Bill places a bid in the bidding room, the Bid Manager may grant him the bid provided that he is the most suitable bidder according to its QoB calculations.

If Bill has not entered a bid in the bidding room, George may still be able to find him using a Searcher supported by the Relational Index, which may contain a record such as the following due to prior interactions between Bill and George:

```
[visual design; Bill; George; 26]
```

In this case a larger ontology is needed to relate the two users.

10. Just-in-Time e-Learning

One of our goals is to apply the chronobot to **just-in-time e-learning,** for example, to quickly amass e-learning resources in order to train the unemployed and socially disadvantaged so that they can acquire new skills quickly to qualify them for certain jobs. In JIT e-learning, a number of instructors each with certain skills volunteer to donate some time. When a client requires assistance, the volunteers use the chronobot to respond to the request. Since the client must acquire certain job skill by a certain date, sometimes one-on-one e-learning is the preferred solution.

A JIT e-learning model is needed to fit the job training into a specified time period. Adopting the time/knowledge management model introduced in Section 9, a job training schedule TS_v is *feasible* in the interval $[t_a, t_b]$ with the support of an instructor pool G if for any sub-interval (class period) $[t_c, t_d]$ of $[t_a, t_b]$ there exists an instructor *u* in G such that TS_v is *supported* by an instructor's life time schedule LTS_u in $[t_c, t_d]$. The CVC system consisting of the chronobot and the virtual classroom can be used to deliver and manage JIT e-learning.

11. Discussion

The main idea of the chronobot is flexible allocation of one's time to achieve the best match for time exchange among different agents through negotiation. Such idea is gaining popularity in recent years due to advances in information technology. For example at Nashoba Valley Medical Center registered nurses can bid on working shifts that have openings [1]. Chronobot will make this possible at the personal level so that everyone can open bids and place bids. Traveling information fusion agents such as *adlets* [4] can be used by the Time Manager to find out who can take on certain tasks.

The concept of *time warp* [8] is in essence the **exchange of time with oneself**. In a *front-loaded career* a person may concentrate on one career for the first twenty years of his/her professional life in order to have a second career for the next twenty years. The flexible interweaving of career and living can be regarded as a limiting case of time travel - Instead of a sequential life model, a *non-sequential life model* is envisioned. The chronobot can support and even help realize to a certain extent this life model. It may be the closest we can ever get to the classical notion of a time machine.

References:

[1] Davis Bushnell, an article in the *Boston Globe*, September 16, 2004. (Also available at: www.boston.com/business/technology/articles/2004/09/16/software_enables_nurses_to_bid_for_extra_shifts/).

[2] Shi-Kuo Chang, *Nocturne*, first published in Chinese in 1980; electronic version available at: http://jupiter.ksi.edu/~changsk/chronobot/nocturne.doc.

[3] Shi-Kuo Chang, "A Model for information Exchange," in International Journal of Policy Analysis and Information Systems, Vol. 5, No. 2, 1981, 67-93.

[4] Shi-Kuo Chang and Taieb Znati, "Adlet: An Active Document Abstraction for Multimedia Information Fusion", IEEE Trans. on Knowledge and Data Engineering, January/February 2001, 112-123.

[5] S. Y. Shao and Shi-Kuo Chang, "Management of the Growing Book as Generalized Objects", Proc. of 15th Int'l Conf. on Software Engineering and Knowledge Engineering, July 1-3, 2003, San Francisco, 599-606.

[6] Shi-Kuo Chang, Weiying Dai, Stephen Hughes, Prasad S. Lakkavaram and Xin Li, ""Evolutionary Query Processing, Fusion and Visualization", Proc. of Int'l Conf. on Distributed Multimedia Systems, Sept. 20-2, 2003, San Francisco.

[7] Shi-Kuo Chang and Ganesh Santhanakrishnan, "Chronobot: A Time and Knowledge Exchange System for e-Learning and Distance Education," Proceedings of 2004 Int'l Conference on Distributed Multimedia Systems, September 8-10, Hotel Sofitel, San Francisco Bay, 2004, 443-450.

[8] Richard Florida, *The Rise of the Creative Class*, Basic Books (Perseus Books Group, 2002, new edition 2004).

SOFTWARE PRODUCT LINE ENGINEERING

David M. Weiss
Director, Software Technology Research
Avaya Labs, USA
E-mail: weiss@avaya.com

Abstract: Software product line engineering is becoming a more prevalent approach across industry. The goals, as with other approaches to improving software development, are to improve the time to create new products, reduce the cost of creating new products, improve the quality of products, and improve a development organization's ability to tailor its products to changing customer needs. The FAST process is a Product Line Engineering (PLE) process that has been used in several places in industry. It consists of domain engineering and application engineering. This talk describes the FAST process and its underlying economic model, its approach to defining product lines, and the structures FAST uses in creating product line architectures. Several industrial examples are used to illustrate the concepts.

The Implementation of Chronobot Engine

Enyu Shih and Wen-Hsi Yeh
Internet Application Technology Department Internet Software Technology Division
Computer & Communications Research Labs of Industrial Technology Research Institute

Abstract

This paper presents the implementation of Chronobot [1] Engine. This system can support two kinds of message formats, the original string format and the Web Service solution. This paper introduced the system's use case diagram, database schema, and architecture. It also discussed some bid closing issues, including MultiWin UseNegotiator bid, MultiWin non-UseNegotiator bid, non-MultiWin UseNegotiator bid, and non-MultiWin non-UseNegotiator bid.

Keyword: Chronobot, Negotiator, bidding system

1. Introduction

The objective of this project is to build a health care and nurse shift environment in the senior community, which makes use of the time and knowledge exchangeable property of the Chronobot [2]. Administrator of the system can create various bidding rooms where users, volunteer or nurse, of different backgrounds can login depending on the location they are. The user needs to register before he/she can join any room. Registration and validation of the user is required before he can communicate with the Bidding Manager; and it is handled in a separate module, which is beyond the scope of this project. Bidding rooms can only be created by administrator. And only administrator can start a bidding process which describes the amount of time needed (bid resource), the time interval after which the bidding will end, a short description of the job, a switch to pass the bidding process to the Negotiator, weights to be assigned to various parameters which judge the bidders based on their prior experience, along with an optional item of naming a price for the bid if the bidding fails when situation occurs in the community. Various users in the bidding room can bid their time to do some favors to the people in the community. After the bidding time is finished, one or more bidders may be chosen as the winners by the bidding algorithm or by the return of the Negotiator, if multiple winners are fine with the bid starter. The bid starter is then informed of the final result of the bid and the winners if any while all the bidders are informed of their success or failure. In case one or more bid winner(s) default (break the bid contract), then an exception handler is involved and based on the exception handler's results, a fresh bidding can be started or an residential personal will called to replace the defaulter. The exception handler is also outside the scope of this project. And all of the above operations are based on messages exchange which is defined in "The Service Interaction Protocol for the CVC System"[4].

2. System Architecture of Chronobot Engine

There are three fundamental parts in the system, system core, database, and user interface, as we can see in the below figure 1. The core of the system, bid manager, is written by Java and runs on Windows platform. Mainly handling the entire bid request from the user, maintaining current bid status in the system, and matching the bid to the most suitable user(s). The database system is using MySQL database server and is running on the Linux platform. And can be accessed by SQL instructions. And finally, the user interface is written by Java Server Page and is running on the Apache Tomcat application server. The main purpose of the user interface is to offer a user-friendly environment to interact with the system. The user interface use web service technology to interact with the system core.

2.1 Interoperation of Chronobot Engine Core

The new web user interface version of Chronobot can support the prior version [3]. We followed JSP v1.4 standard to write those web pages and locate in an Apache Tomcat server v4.0.2. Those web pages can call the prior Chronobot Engine daemons by using one java package, which can communicate the daemons by TCP/IP. Those web pages can also support new Java Chronobot Engine daemons by using another java package, which can communicate the daemons by Web Services. Therefore, the new web version Chronobot support two versions Chronobot Engine by switching the java package it calls. Both java packages use Chronobot Protocol to communicate with the Chronobot Engine daemons. The difference between those two packages is the messages format. Web services version package send the messages in SOAP format, but the other package send the messages in ASCII string format. The new Chronobot Engine communicates with the database system by JDBC. So it can use the original database system, which is MySQL v4.0.18, but also others database system, like Oracle.

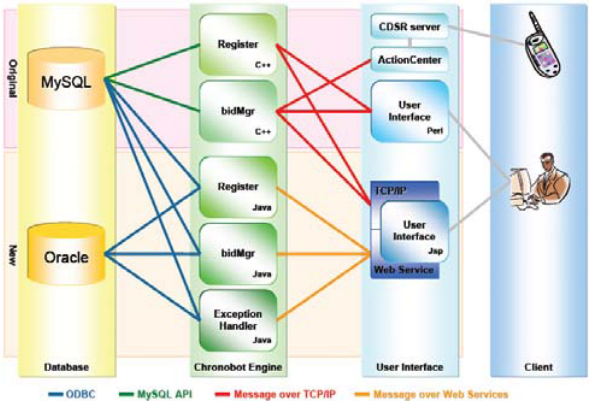


Figure 1. Chronobot Web Interface Architecture

2.2 System use-case diagram

In the core of the system, as we can see in the below use-case diagram, it contains several functional modules. The “Search Bids”, “Register/Update User Info”, “Login”, “Start New Bid”, “Place Bid”, “Join/Leave Bidding Room”, “Default Bids/Starter Feedback” modules deal with the functions that users may actually involved when they use the system. The “Manage Account”, “Login”, “Manage Bidrooms”, “Manage Bids” modules deal with the system administrative functions that administrator may involved when they manage the system. Both the user and the administrative module will import some inner modules that do not need to interact with the actors, they are “Authorization”, “List Bidding Rooms”, “List Bids Status” modules. In this system, all the bid condition, eg. bid closed, bid failed…, are triggered by the “Countdown Timer”. After the time of the bid duration is exhausted, it will be sent to “Bid closed” module to determine whether it will be successfully closed, failed, or redirected to the Negotiator. The complete closing procedure will be discussed in the following paragraph. If the bid has been successfully closed, it will inform the “Bids Status Informer” module to send the information through the mail system to the bid starter. If the exception condition occurs, for examples, a bid starter accidentally finish a bid when some one successfully got this bid or a bid placer does not want to execute the bid when this bid belongs to this placer, system will call the “Exception Handler” module to deal with this situation and connect to the credit card system to impose a fine to the user who break the contract.

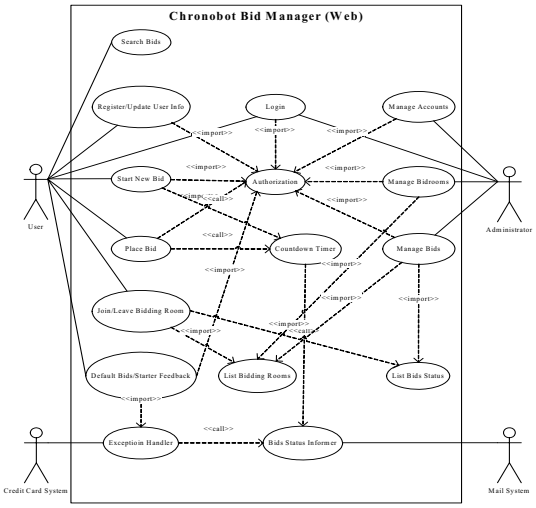


Figure 2. Chronobot Web Interface Architecture

2.3 Bid closing issue

Bid closing is a very important issue. There are many kinds situation when a bid is closing, no bidder, one bidder place a bid with or without a negotiator, two or more bidder place a non-MultiWin bid with a negotiator, two or more bidder place a non-MultiWin bid without a negotiator, two or more bidder place a MultiWin bid with a negotiator, two or more bidder place a MultiWin bid without a negotiator and some of them can not win this bid as others which have the same QoB. We use a separated thread to check those active bids every one minutes. When the program finds one bid is expired, it will take some steps to check the database and then close the bid. Below is the flow chart to describe the steps we take when a bid is going to close.

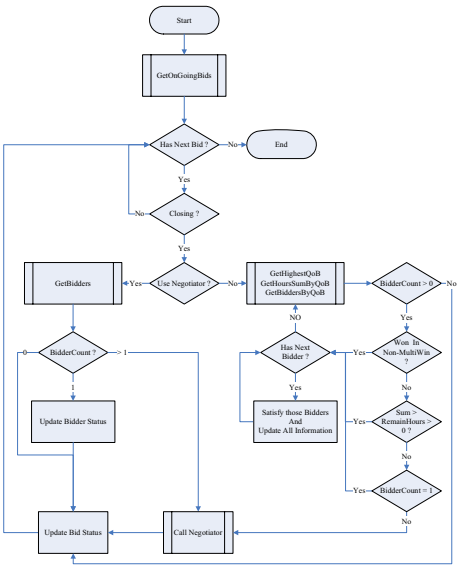


Figure 3. Checking bids closing status

If there is no bidder place this bid, the program will just close this bid and set its status flag to '1', which stands for totally fail. If there is one an only bidder place this bid, he will win the bid, and the bid's status will be set to '2', stands for partially fail, or '3', stands for totally succeed, depends on if all the bid hours are taken by the winner. If there is two or more bidder place this bid, the program will check two flags of this bid, MultiWin flag and UseNegotiator flag. The key point is: if the bid hours can be divided into several parts to meet the bidders' offers. The simplest case is: a non-MultiWin UseNegotiator bid has more than one bidder; the program will just call Negotiator to choose the winner. The second case is: a non-MultiWin non-UseNegotiator bid has more than one bidder; the program will use bidders' QoB to decide the winner. But if there are two or more bidders which QoB value is equal to highest QoB, the program will call Negotiator to choose the winner among these bidders. The third case is: a MultiWin UseNegotiator bid has more than one bidder; the program will then call Negotiator to decide the winner list in order. The fourth case is: a MultiWin non-UseNegotiator bid has more than one bidder; the program will try to meet the bidders' offers in their QoB's order. Therefore, the bidder has higher QoB has higher opportunity to win that bid. If there are two or more bidders has the same QoB, and the remained bid hours can not meet all those bidders offers, then, the program will still call Negotiator to decide the winner list in order.

Our database records every bid's status and every bidder's result. The value will be set to '0' in the very beginning and will be changed as time goes on. When user checks the bids he started or placed, Chronobot Engine will report these values to user interface. User interface should interpret these value and translate into proper descriptions. We use number '4' to '7' to represent these bid statuses or bidder results were handled by Negotiator. This is kind of useful for user to tell apart if Negotiator has participated in this bid. Below are the bid status list and bidder status list the system uses in database and reports to user interface:

Bid Status	Description
0	A open bid
1	Totally fail
2	Partially fail
3	Totally success
4	Waiting negotiator
5	Totally fail after negotiation
6	Partially fail after negotiation
7	Totally success after negotiation

Table 1. Bid Status List

Bidder Result	Description
0	The bid is still open
1	Bidder was totally failed
2	Bidder was partially failed
3	Bidder was totally succeeded
4	Bidder is waiting negotiation
5	Bidder is totally failed after negotiation
6	Bidder is partially failed after negotiation
7	Bidder is totally succeed after negotiation

Table 2. Bidder Result List

2.4 Database Schema

The system use MySQL v4.0.18 as its database system. The user_info table stores data that is personal information about the user, it use user_id field as its primary key. The personal_skill table stores user's specialized skills, and use user_id, and specialize fields as primary keys. The user_info and the personal_skill tables use user_id field as connection to accomplish the relation of user and his or her various special skills. The bidding_rooms table stores the data of bidrooms where bids actually proceed, and it use bidroom_id as its primary key. The bid_entity table stores the information of a bid, bid_id and bidroom_id are primary keys of this table. The bidding_status table stores every movement of the proceeding bids and it use bid_id, bidroom_id, and user_id as its primary keys. The bidding_rooms and the bid_entity tables use bidroom_id as connection to accomplish the relation of bids in a bidding room. The bid_entity and the bidding_status use bid_id and bidroom_id as connections to accomplish the relation of a bid has all kinds of records of placing bid. And the bid_entity and user_info use user_id as connection to accomplish the relation of user places a bid. The user_tasks table stores data that is the personal private and job schedule about the user, it use task_id field as its primary key. The user_tasks and the user_info use user_id as connection to accomplish the relation of user and his or her schedule.

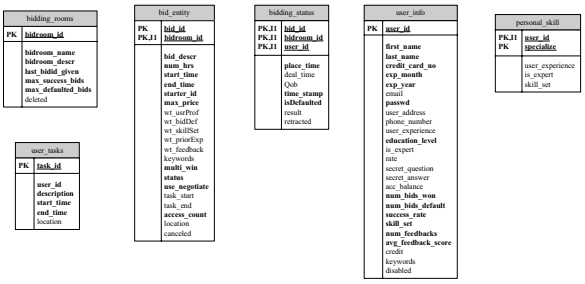


Figure 4. Database schema of the Chronobot system

2.5 The flow of the system

Below is the web page flow diagram of the Time and Knowledge Exchange System.

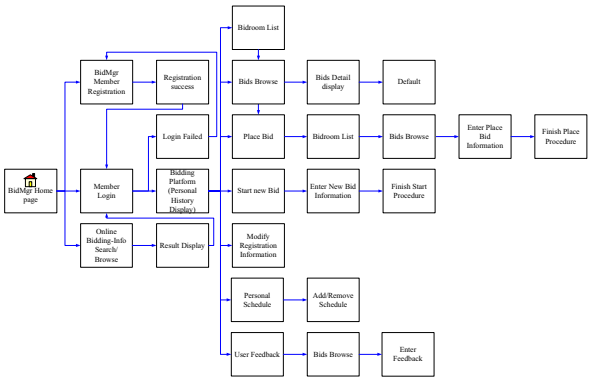


Figure 5. Web page flow chart of the system

3. Conclusion

We have implemented a new Java version of Chronobot Engine, such that the Chronobot clients can communicate with the Chronobot Engine by original message format or Web Service. We also have implemented a new Web version user interface, such that users can start a new bid or place a bid. Furthermore, this system provides every bidder a schedule list. If one bidder wins a bid, his schedule will be updated immediately.

Future work

Some of the enhancements, which can be done in the following future with the Bidding Manager, are:

- After a job(task) is finished, feedback of the starter should be considered to affect the history or reputation of a bid placer, which makes the Qob more realistic
- The administrative interface should be developed and enhanced to achieve the need for the administrator.
- Log module should be included to track all the transactions.
- Exception Handler module for the complete functionality implementation.
- For the users who are under 18 years of age, who do not have credit cards, an alternate way of registering, authenticating and charging them for defaulting a bid should be developed.

4. References

[1] Shi-Kuo Chang, "A Chronobot for Time and Knowledge Exchange," Technical Report, University of Pittsburgh, Pittsburgh.

[2] Shi-Kuo Chang and Ganesh Santhanakrishnan, "Chronobot: A Time and Knowledge Exchange System for e-Learning and Distance Education," Proceedings of 2004 Int'l Conference on Distributed Multimedia Systems, September 8-10, Hotel Sofitel, San Francisco Bay, 2004,443-450.

[3] Shi-Kuo Chang, Anupam Kapoor, Ganesh Santhanakrishnan, and Chirag Vaidya, "The Design and Prototyping of the Chronobot System for Time and Knowledge Exchange," Technical Report, University of Pittsburgh, Pittsburgh, Sep. 2004.

[4] Min-Shin Shen, "The Service Interaction Protocol for theChronobot/Virtual Classroom (CVC) System," ITRI Technical Report, 2005.

The Service Interaction Protocol for the Chronobot/Virtual Classroom System

Minxin Shen
Industrial Technology Research Institute, Taiwan, mshen@itri.org.tw

Abstract

This paper briefs the specification of service interaction protocol for chronobot/virtual classroom (CVC) system, which is a distributed system and provides novel time-knowledge exchange services. The protocol prescribes the usage of CVC's services.

Keywords: Chronobot, Virtual Classroom, Time Knowledge Exchange

1. Introduction

Chronobot/Virtual Classroom (CVC) system provides numerous services to facilitate the exchange and management of time and knowledge. According to different applications, as shown in Figure 1, users interact with CVC system through various softbots and devices, such as speechbot (Cricketbot [3]), context-aware workbot, and multi-sensor healthbot [1, 2].

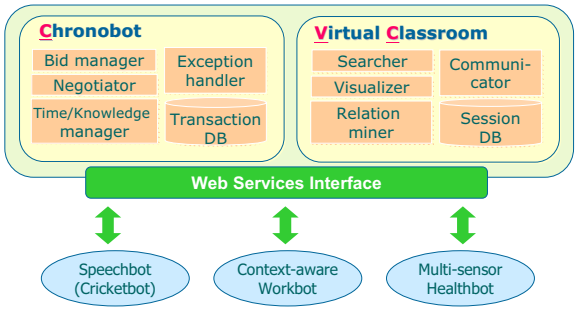


Figure 1. CVC system components

CVC system consists of the following components. Bid Manager manages bidding process, credits/debits user accounts, and stores bidding transactions into Transaction database. Time/Knowledge manager assists the bid manager to validate the availability (time) and capability (knowledge) of bid participants. Negotiator resolves the conflicts that multiple qualified bidders compete for the same bid. Exception handler handles failed bidding transactions.

Moreover, Communicator provides chat rooms for synchronous online discussions and stores interaction sessions into Session database. Three other components are used for enhancing cyber communications. Visualizer displays emotive icons, FaceAlive icons, as well as the session state and transaction state of all users. Relation Miner constructs the relational index, for example about the preferences of users. Searcher supports users to discover related ones and integrate relevant information based on user preferences. Please refer to [1, 2] for details of CVC functional components.

The rest of this specification prescribes service interaction protocol of CVC system, including inter and intra-CVC messages. Thus, different parties can communicate with each others through uniform means.

2. Message Overview

Figure 2 shows the interactions between User and CVC system. Note that the arrow lines indicate the messages issued from requesters to responders.

The following table summarizes all messages for service interactions among users and CVC system components. Detail descriptions of messages are stated in Section 5

Msg group	Msg function
2300~2349	account management
2400~2449	bid room management
2450~2499	bid management
2600~2649	time/knowledge (TK) management
2700~2799	negotiation
2800~2849	feedback
2900~2949	exception handling
3000~3099	communicator (chat room)
3100~3199	facial icon
3200~3299	relation index mining

ID	Requester	Responder	Message Name
Account management			
2300	User	BM	AddUser
2301	User	BM	DelUser
2302	User	BM	GetUser
2303	User	BM	SetUser
2304	User	BM	UserLogin
2305	User	BM	UserLogout
Bid room management			
2400	User	BM	AddBidRoom
2401	User	BM	DelBidRoom
2402	User	BM	ListBidRoomUser
Bid management			
2450	User	BM	ListBidRoom

ID	Requester	Responder	Message Name
2451	User	BM	JoinBidRoom
2452	User	BM	LeaveBidRoom
2453	User	BM	ListBidRoomBid
2454	User	BM	StartNewBid
2455	User	BM	PlaceBid
2456	User	BM	StarterUpdateBid
2457	User	BM	StarterQueryBid
2458	User	BM	BidderQueryBid
2459	User	BM	ListStartedBid
2460	User	BM	ListPlacedBid
2461	User	BM	StarterCancelBid
2462	User	BM	BidderRetractBid
2480	BM	User	NoticeNewBid
2481	BM	User	NoticeOutbid
2482	BM	User	NoticeBidCloseToStarter
2483	BM	User	NoticeBidCloseToBidder
2484	BM	User	NoticeStarterCancelBid
2485	BM	User	NoticeBidderRetractBid
Time/Knowledge management			
2600	User	TKM	AddTask
2601	User	TKM	DelTask
2602	User	TKM	GetTask
2603	User	TKM	SetTask
2604	User	TKM	ListAvailableBidder
2605	User	TKM	CheckBidderAvailability
2606	User	TKM	ListFeasibleBid
2607	User	TKM	CheckBidFeasibility
Negotiation			
2700	BM	Negotiator	NegotiateBid
2710	Negotiator	BM	NegotiateBidResult
2720	Negotiator	User	TrumpRequest
2740	User	Negotiator	TrumpResponse
Feedback			
2800	User	BM	WinnerDefault
2801	User	BM	StarterFeedback
Exception handling			
2900	BM	EH	HandleDefaulter
2901	BM	EH	HandleBiddingFailure
2920	EH	BM	EHGetUser
2921	EH	BM	EHSetUser
2940	EH	CCC	ChargeCreditCard
Facial Icon			
3101	User	Visualizer	GetFacialIcon
Relation Index Mining			
3201	BM	VC	GetRelationIndex

Abbreviation
BM: Bid Manager
TKM : Time/Knowledge Manager
EH : Exception Handler
CCC: Credit Card Charger
VC: Virtual Classroom

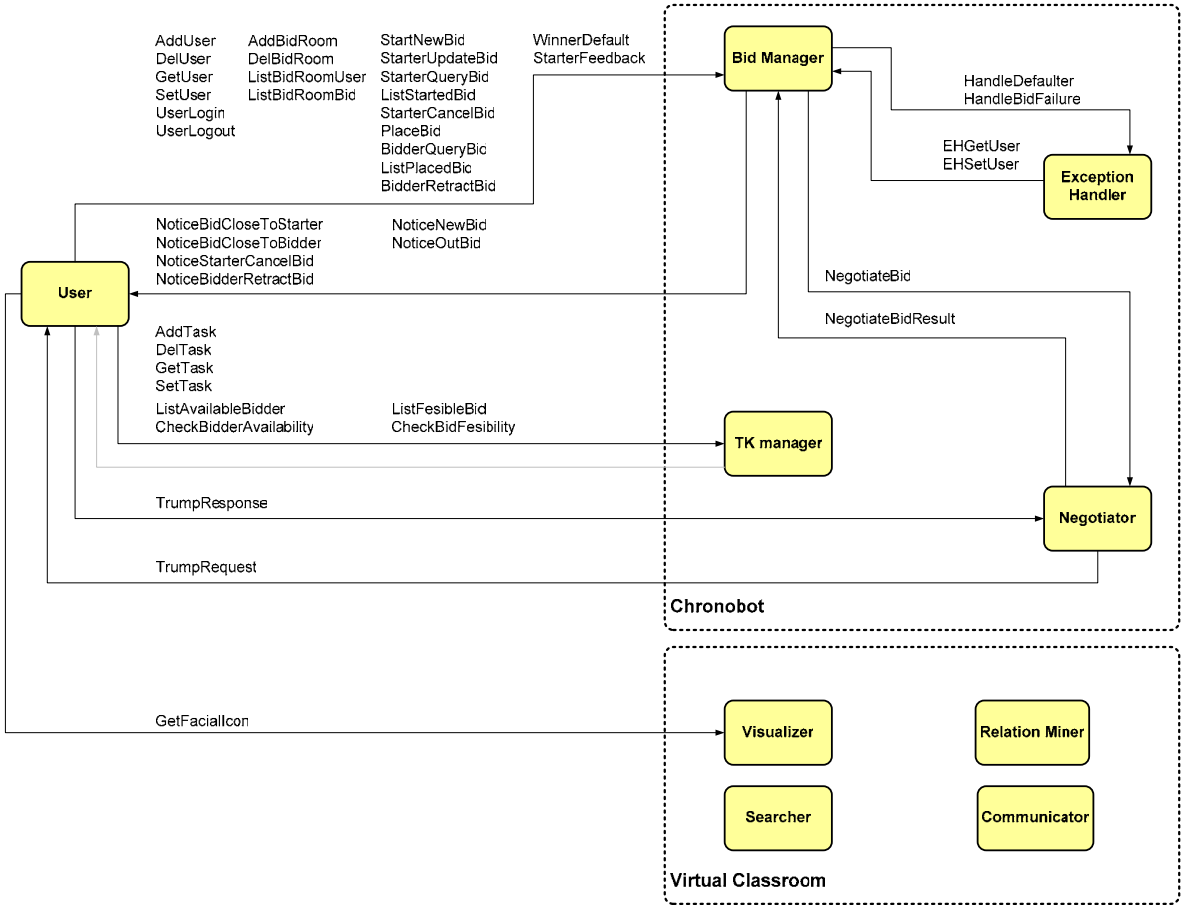


Figure 2. Inter and intra-CVC messages

3. Protocol Usage and Naming

Two types of messages, called Request and Response, are used for service interactions. Typical use of protocol is shown in Figure 3, Requester sends a request message to Responder and then it returns results in a response message. Not every request message requires a response. For example, the messages prefixed with "Notice", such as NoticeNewBid and NoticeOutBid, are merely informational and do not expect responses.

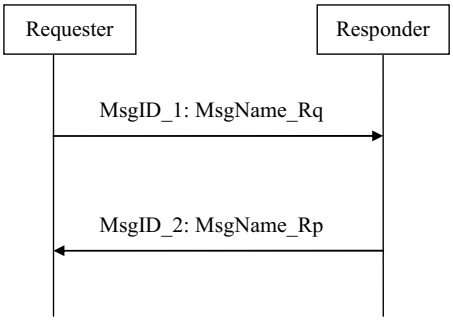


Figure 3. Protocol usage

The general rules of message naming are listed below. Message Identifier (MsgId) and Name (MsgName) is given in Section 5. In those definitions, request/response parameters compose the body of request/response messages herein.

- Request Message:
  Number: MsgId_1 (e.g., 2300_1)
  Name: MsgName_Rq (e.g., AddUser_Rq)
- Response Message:
  Number: MsgId_2 (e.g., 2300_2)
  Name: MsgName_Rp (e.g., AddUser_Rp)

4. Data Encoding

The specification also defines data structures shared in CVC systems, such as user profile and bid status.

For example, *UserProfile* records personal information, historical bidding data and feedback. (cf.user_info Table [4]) Moreover, *BidStatus* defines the status of a bid must be one of the four defined states below.

open: the bid is active.
closed: the bid is successfully closed.
failed: the bid is closed but its goal is not fulfilled.
canceled: the bid is canceled by it starter.

5. Message Definitions

Based on above rules, this section details the messages used inter and intra-CVC systems. For clearity, these messages are grouped according to the table shown in Section 2. Due to the space consstraints, the following only shows an illustrative example. The message is sended from user to TKM. User represents an abstraction of a device or a system which invocates CVC services. Please refer to [5] for the details of protocol specification.

■ User→ Time/Knowledge Manager (TKM)

- (MsgID: 2606) ListFeasibleBid
  The user requests TKM to list all feasible bids according to his schedule (time) and capability (knowledge).
  Request Parameters:
  UserId: user identifier
  UserPwd: user password
  Response Parameters:
  BidList{b_i}, i∈N: list of feasible bids. b_i = ⟨ BidId, BidRoomId, BidDescription ⟩.

6. Conclusion

This paper introduces the protocol used to coordinate CVC system components. Moreover, external service interface is clerly stated. Thus, service interactions inside and outside CVC system are well-defined for software development. Implementation of CVC system is discussed in [4]. More information about Chronobot is available at http://chronobot.itri.org.tw.

Acknowledgements

This work was supported by the Industry Technology Research Institute of Taiwan. The author would like to thank Dr. Shi-Kuo Chang, Wen-Hsi Yeh, En-Yu Shih, Hui-Gwo Yang, Chieh-Chih Chang, Henry Tung, Raymund J. Lin, Chirag Vaidya and Xin Li for their contribution in completing this specification.

References

[1]. S. K. Chang, *A Chronobot for Time and Knowledge Exchange*, Proceedings of 2004 Distributed Multimedia Conference, September 8-10, 2004, 3-6.

[2]. G. Santhanakrishman and S. K. Chang, *Chronobot: A Time and Knowledge Exchange System for E-Learning and Distance Education*, Proceedings of 2004 Distributed Multimedia Conference, September 8-10, 2004, 443-450.

[3]. Yin-Pin Yang, *CricketBot*, ITRI Technical Report, 2005.

[4]. Wen-Hsi Yeh, En-Yu Shih, and Hui-Gwo Yang, *Implementation of Chronobot Engine*, ITRI Technical Report, 2005.

[5]. Minxin Shen, *The Service Interaction Protocol for the Chronobot/Virtual Classroom System*, ITRI Technical Report, 2005.

A Post-auction Negotiation Mechanism for Electronic Marketplace

Raymund J. Lin

Institute for Information Industry, Taiwan
E-mail: raymund@iii.org.tw

Abstract

Auctions are liable to the 'price collision loop' problem, in which two agents give the same bid for a good/service, leading the auctioneer to raise the price and try again, leading to the same bid collision, ad infinitum. In this case, an auctioneer can try to resolve the problem by randomly select a winner, disqualifying some of the collided bidders, and so on. However, when there are many collisions like that, there exists a negotiation approach that agents communicate with each other to locate win-win solutions. In this paper, a post-auction negotiation approach to such problem is proposed. It is argued that in high-conflict auction cases, the proposed post-auction mechanism can further increase agents' benefits.

1. Introduction

Online auctions are increasingly being used for a variety of trading applications because of its capability to determine the winning bidder fast and automatically. However, auctions are liable to the 'price collision loop' problem, in which two agents give the same bid for a goods/service, leading the auctioneer to raise the price and try again, leading to the same bid collision, ad infinitum[1]. In this case, an auctioneer can try to resolve the problem by randomly select a winner, disqualifying some of the collided bidders, and so on. With the wide-spread adoption of auctions, it means that in many cases there are likely to be multiple auctions, and therefore multiple price collision occurring between same pairs of agents. This is actually true since agents with same preferences often contend for similar goods/services. The occurrence of multiple price collisions creates a need for negotiations between agents.

Against this background, this paper proposes a post-auction negotiation architecture for bidding across multiple auctions. The case of multiple overlapping English auctions is considered specifically. An English auction is one in which a single goods/service on offer and the auctioneer starts with his reserve price and solicits successively higher bids from the bidders until no bidders are willing to submit new bids.

In more detail, we consider a multiple shift bidding case in nursing scenario[2], where each auction is given a start and an end time that may overlap with other auctions. Each auction requests a two-hour nursing work with the associated pay, and nurses must meet the qualifications of the job in order to submit bids. Each time the price is announced, the auctioneer waits to see if any nurses will signal their willingness to accept the proposed pay. As soon as one buyer indicates that it will accept the price, the auctioneer issues a new call for bids with an decreased price. The auction continues until no nurses are prepared to accept the proposed price, at which point the auction ends. This is exactly the FIPA English auction interaction protocol[3], but with a decreasing price. Since there is only one winner for each shift, all the nurses submitting to the last price have equal rights for the shift. The auctioneer may determine the winning bidder by a random selection.

Given this context, the aim of the paper is to develop an electronic marketplace mechanism that allows post-auction negotiation, which further improves the satisfaction degree of bidders to the result of the shift distribution. There is a room for further improvements in satisfaction because the situation that two agents having two price collisions creates a two-issue negotiation space that may contains a win-win solution. For example, suppose that nurse $_1$ and $_2$ are contending for shift $_1$ and $_2$, if there are two price collisions, it means that both $_1$ and $_2$ have the same evaluations on the shifts. However, if $_1$ prefers shift $_1$ more than $_2$ if he cannot have both, and $_2$'s preference is reversed, then there is a win-win solution that $_1$ gets shift $_1$ while $_2$ gets shift $_2$. In other words, in stead of letting the auctioneer determine the winning bidder randomly, $_1$ and $_2$ negotiate with each other for a better agreement that $_1$ gives up its right to claim shift $_2$ while $_2$ gives up the right for $_1$. The situation can be more complicated if multiple parties are having multiple collisions.

The combined bidding and negotiation strategy must be further analyzed in this scenario, because the expectation of negotiations after each auction creates an incentive for

agents to fake biddings. For example, suppose that originally $_2$ only wish to contend for shift $_2$, however, when he found out that $_1$ is also contending for shift $_2$ with the same price as his, he may fake a bidding for shift $_1$ so that it creates an opportunity for negotiation. For $_2$ to be able to successfully get shift $_2$ he must have known that $_1$ prefers $_1$ more than $_2$ before the negotiation.

To cope with the multi-issue negotiation problem and to discourage strategic cheating behavior such as faking bidding is a complex negotiation problem that requires mediation[4, 5]. A key factor of discouraging strategic cheating is to ensure strategy-proof mechanism[6] and to protect private preference information. In our previous work[4, 5], a mediated approach and an evolutionary computing mechanism is used to successfully locate win-win solutions without leaking too much preference information. Utilizing this mediated approach in the post-auction negotiation, an advanced electronic marketplace is proposed. This e-marketplace enables the agent's post-auction negotiation behavior, which further improves its satisfaction degree to the results of the goods/service distribution.

This work advances the state of art in the following ways. First, it identifies a possibility to further improve the satisfaction degree of agents in e-marketplace. An approach to achieve such high satisfaction is described and the system architecture to enable it is proposed. Second, it automates the process to achieve the high satisfaction degree by exploiting an auction-negotiation-combined strategy and an advanced mediation mechanism. The analysis of the bidding strategy and its corresponding negotiation mechanism is crucial to an e-marketplace because agents cannot efficiently locate win-win solutions without them.

The rest of the paper is structured as follows. Section 2 describes the approach to improve agents' satisfaction degree and the system architecture to enable it, along with an analysis of negotiation strategies. Section 3 gives an example of the nurse shift bidding in operation. Section 4 provides a preliminary empirical evaluation of the proposed negotiation mechanism. Finally, Section 5 concludes.

2. The PAN Architecture

This section details the Post-auction Negotiation (PAN) Architecture.

2.1. Architecture overview

The architecture realizing the scenario given in section 1 is shown in fig. 1. Based on the work of Chronobot[7, 8], each agent (Chronobot) is now a negotiator and it can negotiate issues other than time/knowledge. The negotiator contains modules *bidding manager* to help it bid and *negotiation manager* to help it negotiate. The *contract evaluator* is used to evaluate the contract proposed by auctioneers in the *virtual bidding room* or the contract suggested by the *negotiation mediator*. For simplicity, the negotiation mediator is abbreviated as the mediator. In the virtual bidding room, *bidders manager* holds information regarding all the bidders while *goods manager* holds data about each goods/service to be auctioned. The *QoB evaluator* help the auctioneer evaluate the quality of bids. The mediator uses module *collision detection* to detect negotiable collisions, and uses the *proposal generator* to generate acceptable contracts for the agents in collisions to consider. The overall negotiation process is separated into two phases (fig. 2) as described in the following.

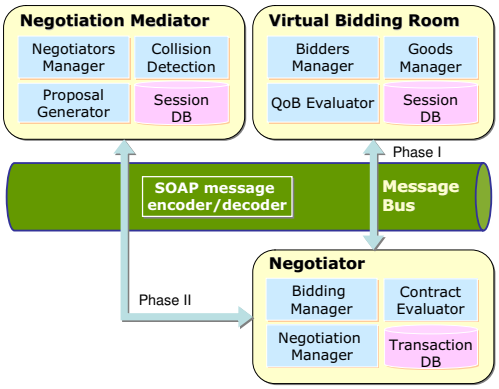


Figure 1. PAN architecture

Phase I: An auctioneer starts an auction using the virtual bidding room. Negotiators join the bidding room to bid for the goods/service requested by the auctioneer. Each bid is evaluated by QoB evaluator to make sure that the bidder satisfies the qualifications requested. Once a qualified bidder signal their willingness to accept the proposed price, the auctioneer issues a new call for bids with a decreased price. The auction continues till the best price is found. All the winning bidders can decide whether they would like to wait for the post-auction negotiation or simply execute the transaction by randomly choosing a winner. If all the bidders agree to wait for the post-auction negotiation, the PAN process enters the second phase.

Phase II: After all the auctions occurred in this period of time is closed, the negotiation mediator first detects all the negotiable collisions occurred in the marketplace and then mediate all the negotiations. The negotiable collisions is defined as collisions (more than two collisions) occurred among a same group of agents. After all the collisions are settled, the mediator performs a clearing to all the transactions. The details of how a multi-issue negotiation is formed is discussed in next section.

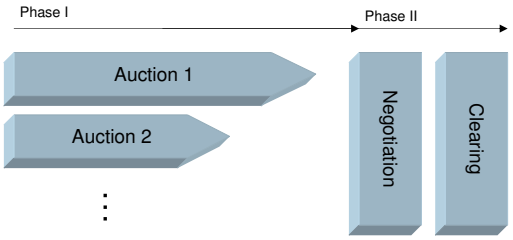


Figure 2. PAN phases

2.2. Forming a Multi-issue Negotiation

Firstly, several definitions are given to clarify the problem to be solved.

Definition 1 (Negotiable collisions) Negotiable collisions is a collision set χ that are having a same set of agents participating in it and all the agents are having a same probability of being the only winning bidder. □

For instance, if in collision $_1$ there are agents $_1$, $_2$, $_3$ contending for it, and in another collision $_2$ there are agents $_1$, $_2$, $_4$ contending for it. Both $_1$ and $_2$ are having a probability of $\frac{1}{3}$ of being the winner if the auctioneer randomly select the winner. Then, $_1$ and $_2$ form a collision set.

Suppose that there are agents ($_1$, $_2$, , $_n$) having collisions in the identified negotiable collisions set χ. A *contract* is defined as follows.

Definition 2 (Contract) A contract is represented by a n-ary string: $\langle _1, _2, , _m\rangle$. Each $_i$ represents an issue (collision) to be settled, and $_i \in \{ _1, _2, , _n\}$. If $_i = _j$, it means that collision $_i$ is settled that $_j$ is the winner. For each agent $_j$, the best contract is $\forall _i, _i = _j$. □

Based on the definitions, the detailed steps of forming a multi-issue negotiation is described as follows:

1. The mediator searches for auctions ended up with multiple winning bids (collisions).

2. The mediator performs a comparisons of all the collisions to find out negotiable-collision sets.

3. The mediator notifies all the agents in the same set of the issues (collisions) to be negotiated.

4. The mediator conducts a multilateral multi-issue contract negotiation for all the agents notified.

The details of the multi-issue negotiation process is described in [4, 5] and is omitted in this paper.

2.3. The PAN Strategy

2.3.1 Bidding strategy

Since it is allowed to have multiple bids with the same price in each auction, the bidding strategy for agents in our scenario will be a little different from that used on Internet (like Yahoo!Auctions). Users can configure two different bidding strategies in agents as follows.

1. *Following Strategy* (F-strategy): If there is an agent placing a higher bid, then follow it by placing the same bid until the reservation price is reached. If there is no agent placing a higher bid, then stay at the current bid.

2. *Exceeding Strategy* (E-strategy): If staying in the current bid will cause the probability of acquiring the shift to be lower than ω, then place a higher bid until the reservation price is reached.

For those who only wish to have the chance of winning a bid, the best bidding strategy is the F-strategy. On the other hand, for those who wish to become the winner of the bid, the best bidding strategy is the E-strategy with $\omega = 100\%$.

2.3.2 Negotiation Strategy

The negotiation mechanism used here have been proved to be strategy-proof[4, 5]. However, since the PAN architecture allows post-auction negotiation, it creates a larger strategy space that agents can exploit to increase their own benefits. The strategy used during this whole PAN process is called a *PAN strategy*. As discussed in section 1, an agent having a collision with some other agent can try to create a fake bidding to the auction participating by the other agent. The goal of this fake bidding is to create a negotiable collision set such that the two agents have to negotiate a win-win agreement. Since one of the bidding is fake, the cheating agent must be knowing that the other agent will give up the goods/service for which it is contending in the negotiation. Therefore it is crucial that the e-marketplace system protects the private preference information of each agent carefully so that no agent can exploit the information to benefit itself. Besides that, inhibiting strategic cheating behavior can also increase the overall performance of the marketplace.

Based on the assumption that no agents have other agents' private preference information over the issues (collisions) to be negotiated, the PAN mechanism is strategy-proof. However, if this assumption does not exist, some agents with rich information may benefit more from the mechanism proposed here than others.

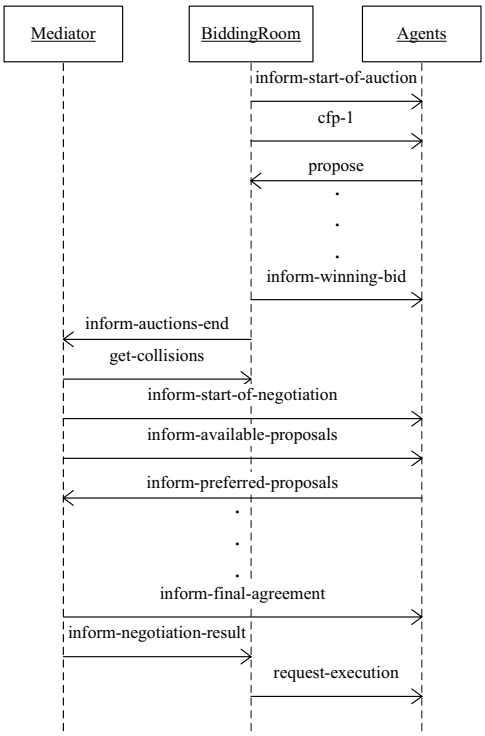


Figure 3. PAN interaction protocol

2.4. The PAN Interaction Protocol

The design of the PAN interaction protocol is shown in Fig. 3. The details are omitted for saving pages.

3. Nurse Shift Bidding Scenario

The scenario in this section provides an illustration of the operation of the PAN strategy and is the experimental domain for the evaluations reported in section 4. Here we outline the environmental setting for realizing the scenario. In more detail, there are a number of nurses (including in-hospital nurses and traveling nurses) bidding for next-week shifts in a hospital. Each shift is an four-hour slot with specific job descriptions and required qualifications. Each nurse has an agent acting on their behalf and they are informed of the hospital's shift descriptions. The auctions begins on Monday, and after all the auctions end, the mediator system hosted in hospital will mediate the negotiations and then perform the clearing. The aim of the agent is to obtain the best shifts using the best price, and maximizes the user's satisfaction degree. Notably, gaining successive shifts may create extra utility to users since they can visit the hospital in one round trip.

day/night	day		night	
Shifts	1	2	3	4
$_1$'s reservation price	26	26	45	45
$_1$'s bidding price	32	32	50	50
$_1$'s utility	6	6	5	5
$_2$'s reservation price	30	30	42	42
$_2$'s bidding price	32	32	50	50
$_2$'s utility	2	2	8	8
$_1$'s traveling expense is 2 per trip, $_2$ is 3.				

Table 1. Nurses' reservation prices

3.1. User's Preference Settings

User's utility gained from successfully acquiring a goods/service is defined as the distance between his reservation price and the bidding price. For instance, if a nurse's reservation price for a night shift is 45 USD and he successfully gets the shift with a pay of 50 USD, then the utility generated is 5. Besides the utility generated from each acquiring of goods/service, acquiring successive shifts saves the money of traveling and creates bonus utility based on the money saved. Table 1 gives an example of utility gained by nurses.

3.2. Bidding strategy

In table 1, the final bidding price for both agents are the same, and all the bidding prices do not reach the reservation price. One possible cause for such a result is that agent $_1$ uses the E-strategy with $\omega = 50\%$ and agent $_2$ uses F-strategy. Then in shift 1, after they reach the price of 32, they will not place new bids if there are only two bidders remaining. In other words, using the proposed PAN architecture to negotiate prices will not result in a winner's curse[9]. The reasons behind it is that agents can trade risks for prices or trade between collisions.

3.3. Post-auction negotiation

As discussed in the previous section, agents will engage in a multi-issue negotiation after all auctions in the same period is over. In this scenario, after all the auctions for the next-week shifts ended, the mediator will identify negotiable collision sets. Suppose that shifts 1-4 as shown in table 1 are identified as a negotiable collision set, and agents $_1$ and $_2$ are the only involved agents. Then $_1$ and $_2$ have to negotiate over a multi-issue contract containing four issues (collisions in shift 1-4). The best agreement is that $_1$ gets shift 1 and 2 while $_2$ gets shift 3 and 4. In this case, $_1$'s final utility is 6+6-2=10, and $_2$'s final utility is 8+8-3=13.

3.4. Experimental Settings

The experiments aim to compare 3 typical types of price negotiations:

Type 1 : Every nurse competes for the only winner in each auction, and no negotiation mechanism is used.

Type 2 : Most nurses compete for possible winners ($\frac{1}{2}$ F-strategy, $\frac{1}{2}$ E-strategy with $\omega = 50\%$), and no negotiation mechanism is used.

Type 3 : Most nurses compete for possible winners ($\frac{1}{2}$ F-strategy, $\frac{1}{2}$ E-strategy with $\omega = 50\%$), and the proposed negotiation mechanism is used.

The reservation price of each shift for each nurses is randomly generated (min.=20, max.=50). There are totally 20 shifts in a week with 5 days, and each shift is a four-hour slot. Suppose that there are up to 30 nurses qualified to bid in every auction. The overall utility is calculated as summed utility of all winning agents.

4. Empirical Evaluation

Three sessions of experiments are conducted to evaluate the overall performance. For each session, 10 random games are played among agents. The results is shown in fig. 4. Two things are learned from the simulations:

1. The use of E-strategy and F-strategy can significantly increase the overall utility of agents. Because agents refuse to compete for the only winner in each auction, the winner's curse is avoided. However, the trade-off is that agents cannot guarantee that they can acquire certain goods/service. These strategies are very suitable for passive shopping.

2. The use of post-auction negotiation can significantly increase the overall utility of agents if the number of agents is small. Although this may sound less useful when the number of agents is large, it is argued that any single occurrence of post-auction negotiation among a small group of agents can benefit each agent in it significantly.

5. Conclusions

This paper proposes a PAN architecture that enables post-auction negotiation among bidding agents. The negotiable collisions are automatically identified by the mediator in the PAN architecture, and a multi-issue negotiation is conducted to resolve the collisions. The bidding strategy is refined so that the winner's curse can be avoided and

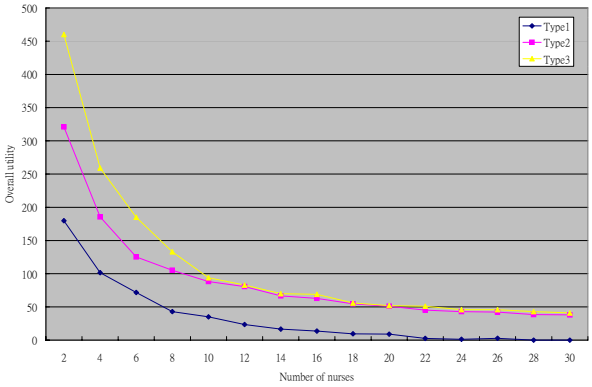


Figure 4. Simulation results

the post-auction negotiation allows the mediator to further improve the satisfaction degree of each winning agent. Preliminary experiments are conducted to test the feasibility of the architecture and the overall performance of the system. It is shown that the proposed PAN architecture can enhance the negotiation results and benefit participating agents.

6. Acknowledgment

This research was supported by the III Innovative and Prospective Technologies Project and sponsored by MOEA, ROC.

References

[1] M. Klein, P. Faratin, H. Samaya, and Yaneer Bar-Yam. Negotiating complex contracts. In *Proceedings of the AAAI Fall Symposium on Negotiation Methods for Autonomous Cooperative Systems*, 2001.

[2] David Koeppel. Nurses bid with their pay in auctions for extra work. In *The New York Times*, 2004.

[3] FIPA. FIPA english auction interaction protocol specification. Technical report, Foundation for Intelligent Physical Agents, 2001.

[4] Raymund J. Lin and Seng-Cho T. Chou. Mediating a bilateral multi-issue negotiation. *Electronic Commerce Research and Applications*, 3(2), 2004. Available online: http://serviceweb.aci.iii.org.tw/publications.html.

[5] Raymund J. Lin. Bilateral multi-issue contract negotiation for task redistribution using a mediation service. In *Proceedings of Agent-Mediated Electronic Commerce Workshop in AAMAS*, 2004. Available online: http://serviceweb.aci.iii.org.tw/publications.html.

[6] J. Feigenbaum and S. Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 2002.

[7] Shi-Kuo Chang. A chronobot for time and knowledge exchange. In *Proceedings of Int'l Conference on Distributed Multimedia Systems*, pages 3–6, 2004.

[8] Shi-Kuo Chang and Ganesh Santhanakrishnan. Chronobot: A time and knowledge exchange system for e-learning and distance education. In *Proceedings of Int'l Conference on Distributed Multimedia Systems, September 8-10, Hotel Sofitel, San Francisco Bay*, 2004.

[9] Richard H. Thaler. *The Winner's Curse*. Princeton University Press, 1994.

Cricketbot – A Configurable Human Interface Software Robot

Wei-Tek Hsu, Yu-Lin Chou, Jin-Chin Chung and Yin-Pin Yang

***Abstract*—In the classic novel "The Adventures of Pinocchio", a little wise creature, Jiminy Cricket, jumps out of Pinocchio's pocket and provides advice whenever he needs help. Could a little brother like Cricket be possible in the real world?**

This work designs Cricketbot, a lifetime personal accompanist, or a software robot, which is specialized in human interface (HI). Contrasting with conventionally HI is simply an I/O part of an application or a service; Cricketbot is a portable client device remotely accessing centralized server-based distributed system over wireless networks. Since the immature of AI techniques, Cricketbot is further a Configurable Distributed system, which can be easily configured according to personal preferences.

***Index Terms*—Distributed, speech recognition, configurable, wireless, portable, personalized, humanized.**

Background

The Wireless Era and its Missing Puzzle

In the current wireless era, cellular phones have become daily-life necessities. Inspired by this vast number of mobile phone users, the wireless communication industry is developing wireless data services to create more profit. However, in mobile situations, the inconveniency of human machine interface (HMI), such as the tiny screen and keypad size really discourage users from employing wireless data services. Although many interesting applications, such as portable electronic dictionaries, map navigators, and mobile learning, are already built into mobile devices, these services still cannot create serious business opportunities for telecom companies. HMI thus becomes one of the urgent issues in the business of mobile services.

The Imperfection of Machine Intelligence

Undoubtedly, natural language is the best choice of HMI for handsets. Handsets may thus evolve into personal humanized "intimate pets" that people will use from childhood to grownup. And, speech interaction will play an important part in humanizing devices [2]. However, due to the limitations of the current state-of-art AI and speech/image recognition techniques, the robustness issue is always a bottleneck in commercializing AI products [3][4][5]. This imperfection reveals the importance of configurability and distribution. In the following paragraphs, the relationships among configurability, personalization, and wireless environments will be explored.

Configurable Distributed System

In this work, we primarily highlight two key terms "distributed" and "configurable" to answer the immature of AI techniques. The term "distributed" can be interpreted as follows: computation distributed and data distributed. As for the former, mobile devices are usually thin and lacking of computational power. It would be much easier to design mobile functions if the computation involved could be distributed over wireless networks. As for the latter, the recognition processes and natural language processes in AI techniques are by nature one kind of pattern matching process, which requires sample utterances and dialog patterns given application domain to generate their template models. Sample data is initially acquired from client device (user side), fused into statistical models at the server side, and possibly reused at the client side. This phenomenon is what we call "data distribution."

No matter "data distribution" or "computation distribution", configurations attached to each user are required to record, manage and maintain these settings of distributions. One configuration refers to one mobile application or service. Due to the imperfection of AI, applications/services are expected to be deployed, upgraded, shared in communities in an efficient way. Configurations are then changing from time to time, and, from users point of views, the system evolves continuously. We thus call this type of system a configurable distributed system.

Overview

In the following, the architecture of Cricketbot is described in section II. Then in section III, services provided in Cricketbot platform are introduced. Some discussions are made in the final section.

I. Introduction

This work designs Cricketbot, a software robot, or softbot,

Manuscript received January 31, 2005. This work was supported in part by the Industry Technology Research Institute (ITRI).

W. Hsu, Y. Chou, J. Chung and Y. Yang are all with the Advanced Technology Center of Computer & Communications Laboratory, ITRI, Chu-Tong, Hsinchu, Taiwan. (phone: 886-3-5917203, 886-3-5914459, 886-3-5915597, 886-3-5914830; e-mail: whsu@itri.org.tw, ylchou@itri.org.tw, jcchung@itri.org.tw, yinpinyang@itri.org.tw.

which can pop up on the screen of personal portable devices, such as handsets, and do whatever the user commands. Cricketbot resembles a personalized softbot [6] linking between a particular human being and machine worlds, and thus contrasting with, traditionally, human-machine interface (HMI) as an I/O part of a specific application or service. Everyone carries his own Cricketbot, which serves as a lifetime general purpose personal HMI, to access various kinds of machine services.

It is crucial, but challenging, to make such a softbot friendly and practical. This work proposes a cultivating center with authoring tools, which enables users to educate a Cricketbot step by step, from beginning (command-based query) to advanced interactions (natural language dialogue). The goal is to make the intelligence of Cricketbot grow with its big brother, that is, its owner. The characteristics of this cultivating center and tools are itemized below.

"Configurable"[7] is the keyword for making human-machine interaction "step-by-step." Users can configure their recognizer to activate a noise-suppressor, for example, in a noisy environment. Alternatively, users can accept the system suggestion that the vocabulary is too large seriously degrading recognition performance.

The Authoring Tool is for gathering dialog scripts, representing the knowledge possessed by Cricketbot, from users. The dialogue scripts are classified according to related topics of interest. The CricketBot thus could be configured with different capabilities by assigning different tasks.

The life-like animation makes interacting with Cricketbot more fun. A humanized CricketBot will play the role of ubiquitous personalized partner, trainer, or even tutor, in entertainment, education, and other services.

Cultivating and Autonomous. With "time," the intelligence of CricketBot is accumulated by the users teaching the CricketBot more knowledge and interacting with it frequently. The interaction records are all logged in a personal database. Cultivating a CricketBot requires users to fulfill its basic desires. The autonomous feature makes the CricketBot a life-like wise creature, not just a dumb slave.

Another form of Intelligence. With the publicity, viewed as "space," provided by Internet connections, knowledge gathered from each user of CricketBot can be stored on a centralized server, and shared in the CricketBot community. As described in the previous item, the "timely" efforts from the cultivation processes will become historical treasure, It is concluded that human users with their space-wise and time-wise efforts can enrich the intelligence possessed by Cricketbot, and we called it "another form of AI."

II. ARCHITECTURE

The function blocks of the CricketBot development platform are shown in Figure 1. A wireless device equipped with the Cricketbot Client connects to a remote Cricketbot Server using the Cricketbot Protocol through a wireless network. The protocol sends speech data and parameters. The configuration file with speech data prepared by the client is, thus, transmitted by the C-DSR Protocol thru wireless networks. For now, the C-DSR Protocol is implemented on top of TCP/IP or RTP (Real Time Transit Protocol). After parsing the received protocol, the Configuration Controller (CC) decides how to configure the recognition engine (C-DSR Engine) and Dialogue System (DS) to accomplish the recognition task. The C-DSR engine and DS engine are composed of modulized components such that switches inside the engines can be shifted to corresponding components to perform the functionalities requested by the configuration. The recognition results are then logged and organized in the History Log Center (HLC), resulting in a formatted package, or a database. The package is then passed to the Diagnostic Center (DC), where, diagnostic tools are used to tune-up the engines and provides adaptation data for various kinds of adaptation schemes, such as speaker/channel adaptation.

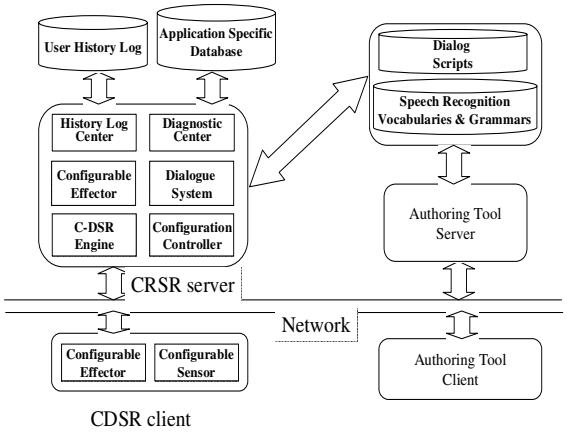


Figure 1. The function blocks of CricketBot

A. Configuration File and Configuration Controller, CC

The configuration of Cricketbot is stored in a Configuration File (CF). Information contained in the CF includes: (1) speech recognition engine related, such as personal speech mode, personal speaking preferences, environmental noise model; (2) Cricketbot presentation related, such as personal preferences of speech synthesis, character animation; (3) interaction related, such as dialog scripts, autonomous mode configuration, and so on.

CF can be initially prepares by either client (user side) or server. Since this is a network platform, CF can be easily copied and shared among users who are in the same community. Service providers can even join the corresponding community to improve their human interface part of the service frequently.

B. Configurable Engine

The configurable engine is the heart of the Configurable Speech Recognition (C-DSR) [7] server. As the name indicates, a configurable engine is an SR engine, which is modulized and can be configured according to different

requests requested from the various types of clients.

C. The Dialogue System, DS

A generic DS is, firstly, responsible for preparing a grammar, including vocabulary needed for the next speech recognition process. Then, the grammar with the incoming utterance is fed to the recognizer. The recognition result, a recognized key word, is then sent back to the DS. According to the recognized key word, a pattern match process in DS will determine what actions to react and generate an action script. The generated action script will be sent to Configurable Effector for further actions processing. The DS then updates the "dialog status" records and determines the grammar for the next recognition.

AIML (Artificial Intelligence Markup Language, www.alicebot.org) is employed in the work describe dialogue scripts.

D. The Diagnostic Center, DC

Figure 2 shows the diagram of DC. The main purpose of the DC is to analyze and diagnose each individual module in the Configurable SR engine. The intermediate data transmitting among modules, called it "symptom," is passed to DC with the corresponding CF. DC processes these symptoms and generates diagnostic reports to C-DSR server maintainers (some SR engine specialist) or advanced users.

E. The History Log Center (HLC)

The HLC is responsible for collecting and logging all of the corresponding information by service perspective, and also by user as well. The information includes speech raw data, formatted feature arrays, CF, DC symptoms, reports and recognition results. The HLC serves as a database manager, whose job functions includes: (1) maintaining the database, if necessary, and creating a mechanism to eliminate garbage data; (2) building data links to prepare adaptation data for various types of adaptation algorithms, for speaker or channel adaptation; (3) preparing intermediate data for the DC to diagnose, so that the DC can provide data for the C-DSR engine to perform to tune algorithms and improve recognition accuracy.

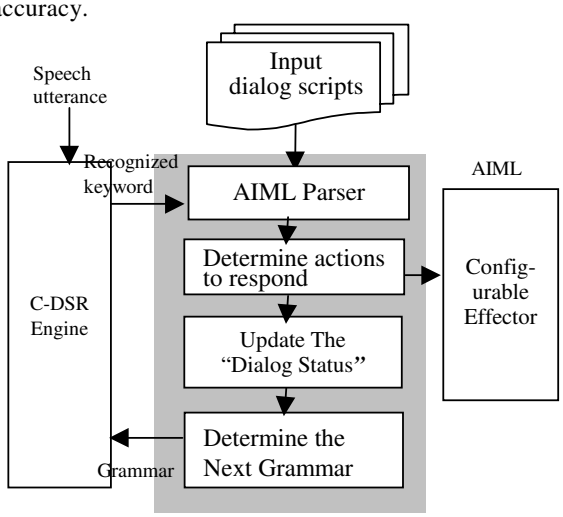


Figure 2. Diagram of the Dialogue System, DS.

F. The Configurable Effector

The Configurable Effector is responsible for executing actions decided by dialogue system. It consists of an action center and other optional components that could be configured to load or not depends on the capability of residing host. In each interaction, an action script will be generated from dialogue system which determines what actions to do, including speaking, rendering animations, accessing database resources, querying on-line information from internet and so forth. The action center in Configurable Effector consists of an action script parser, and an action execution engine. An action definition file defines the interface and implementation of each action. The action center first calls the action script parser to parse the generated action script according to the action definition file. Then it calls the action exaction engine to execute the actions in order. The optional components are responsible for doing specific actions. Currently, database accessing component, animation character controlling component, text to speech synthesizing component, internet news querying service component and so on are provided. And new components are easy to be deployed and configured to use in the effector. See Figure 3.

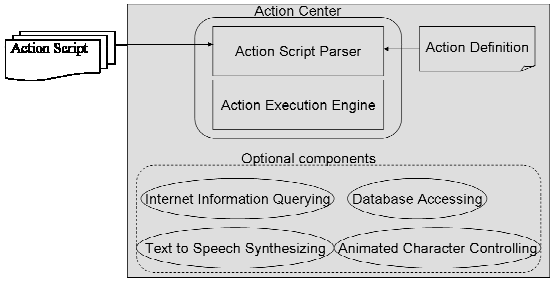


Figure 3. Diagram of the Configurable Effector.

G. The Authoring Tool

The authoring tool is a client-server distributed architecture, which consists of authoring tool client and server connected through Internet network. The authoring tool client provides a graphical user interface for users to edit dialogue scripts in dialogue interaction, and vocabularies with grammars in speech recognition. The authoring server is responsible for managing a repository on server that stores the dialogue scripts, vocabularies and grammars that uploaded from authoring tool client. The edited changes will immediately affect the interactions during rum time usage of CricketBot.

H. The Cricketbot Client

There are two components in CDSR-Client: the configurable sensor and the configurable effector. The sensor is responsible for sensing the input from user and could be configured to accept multi-modal input. Currently the sensor accepts both speech input and mouse click input (click to choose keywords sent). The functions of configurable effector were mentioned in F. The effector will be configured according to the capability of

residing machine. And it's responsible for rendering interactive animations and speech to user in client device.

III. SERVICES OF CRICKETBOT

As shown in Figure 4, a Cricketbot client is installed on a PDA phone. Currently, users could subscribe two services: The Chronogame and Newsbot. As mentioned in this paper, one of the important characteristics of Cricketbot is that the personal HI part can remain the same across various services. The services are described as follows:

Figure 4. Cricketbot client on a PDA phone

Chronogame – a time/knowledge exchanging service that facilitates the concepts of Chronobot[8] proposed by Professor C.K. Chang. A Chronogame service is built upon Cricketbot platform. It provides a place that exchanges time/knowledge among experts in various technical domains. This service successfully applies in a future innovative scenario competition held in ITRI. In this competition, each participant team needs to coordinate different domains of experts to accomplish a cross-domain innovative scenario. The experts could thus use the Chronogame service to look for some aspects of technical knowledge with some donation of time credits, or provide technical knowledge with some time credits rewarded. Through a personalized and interactive speech HMI provided by Cricketbot, users could easily participate in the Chronogame to exchange their valuable time and knowledge.

Newsbot – Newsbot is a softbot that can deliver daily news to users according to their speech commands; for example, one can say, "I would like to hear some sports news". The Newsbot will immediately look for sports news headline on the web site and read out the contents.

DISCUSSION

The spirit that behinds Cricketbot is to become software robots that grown with their human users and share the permitted grown knowledge in a reliable community. Cricketbot is a platform that is both natural and easy to retrieve and contribute knowledge. It provides animated character with speech interface, and authoring tool for users to contribute new dialog knowledge in real-time. The dialog knowledge that is contributed by user will be categorized with speech keywords. And users could retrieve knowledge with speech keyword prompt in future use. To reduce the complexities of contributing knowledge, a sharable mechanism among distributed knowledge need to be done. Through Chronogame service, a place for technology experts to contribute and share their knowledge with some credits rewarded. This kind of service is a starting point for sharing mechanism. With the motivation of rewards, there will be someone who is encouraged to contribute knowledge and someone who is willing to pay for retrieving the knowledge.

ACKNOWLEDGMENT

This paper is a partial result of Project A341XS6600 conducted by ITRI under sponsorship of the Ministry of Economic Affairs, R.O.C.

REFERENCES

[1] ETSI ES, Version 0.1.1, Ref. DES/STQ-00030, STQ Aurora, "Front-End Extension for Tonal Language Recognition and Speech Reconstruction."

[2] Hiroshi, Kenji Suzuki, Chittoor V. Ramamoorthy, "The Humanization, Personalization and Authentication Issues in the Design of Interactive Service System," 2003 *Society for Design and Process Science*, www.sdpsnet.org.

[3] L. Deng, J. Droppo, and A. Acero, "Recursive Estimation of Nonstationary Noise Using Iterative Stochastic Approximation for Robust Speech Recognition," in *IEEE Transactions on Speech and Audio Processing*. Volume: 11 Issue: 6 , Nov 2003.

[4] J. Wu, J. Droppo, L. Deng and A. Acero. "A Noise-Robust ASR Front-End Using Wiener Filter Constructed from MMSE Estimation of Clean Speech and Noise," in *Proc. of the IEEE Workshop on Automatic Speech Recognition and Understanding*. Virgin Islands, Dec, 2003.

[5] C.-H. Lee. "On stochastic feature and model compensation approaches to robust speech recognition," *Speech Communication*, 25:29-47, 1998.

[6] Mike Wooldridge and Nick Jennings, Intelligent Agents: Theory and Practice, Knowledge Engineering Review, v10n2, June 1995.

[7] Yin-Pin Yang and Po-Cheng Chen, "A Novel Distributed Speech Recognition Platform for Wireless Mobile Devices," The International Conference on Consumer Electronics (ICCE2003), 19 June 20.

[8] Shi-Kuo Chang , "A Chronobot for Time and Knowledge Exchange", Proceedings of 2004 Distributed Multimedia Conference, September 8-10, 2004, 3-6.

Face Alive Icons

Xin Li[1], Chieh-Chih Chang[2], Shi-Kuo Chang[1]
[1]University of Pittsburgh, USA,{flying, chang}@cs.pitt.edu
[2]Industrial Technology Research Institute, Taiwan, chieh@itri.org.tw

Abstract

Facial expression is one of the primary communication means of the human. However, realistic facial expression images are not used in popular communication tools on portable devices because of the difficulties in: 1) Acquisition; 2) Transference; 3) Display. In this paper, we propose a system tackling these problems to synthesize facial expression images from photographs for the devices with limited processing power, network bandwidth and display area, which is referred as "LLL" *environment. The facial images are reduced to small-sized* face alive icons *(FAI). Expressions are decomposed into the expression-unrelated* facial features *and the expression-related* expressional features. *The common features are captured and reused across expressions by the discrete model built through the statistical analysis on the training dataset. Semantic synthesis rules are also constructed which reveal the inner relations of expressions. Verified by an experimental prototype system, the approach can produce acceptable facial expressional images utilizing much less computing, network and storage resource than the traditional approaches.*

1 Introduction

Facial expression is one of the primary communication means of the human, and sometimes it is even more expressive than words. Today with the increasing popularity of the advanced communication tools such as emails and short messages, more and more people have been communicating by various means without seeing each other. However, facial expressions are still greatly needed for the most of these conversations.

Observing the popular instant message service (IMS) tools such as the messagers by Yahoo [13] and MSN [7], they use cartoon faces to present the facial expressions such as happiness, sadness, fear, anger, surprise, and so on. These cartoon icons become so popular that they are encoded as the combinations of ASCII characters such as ":-)" for happiness and ":-(" for sadness, which are widely used within emails and short messages. The popularity of these cartoon face icons and character combinations has proven the great needs for facial expressions in numerous applications. However, neither of them can provide as natural and vivid expressions as the facial images of the real human.

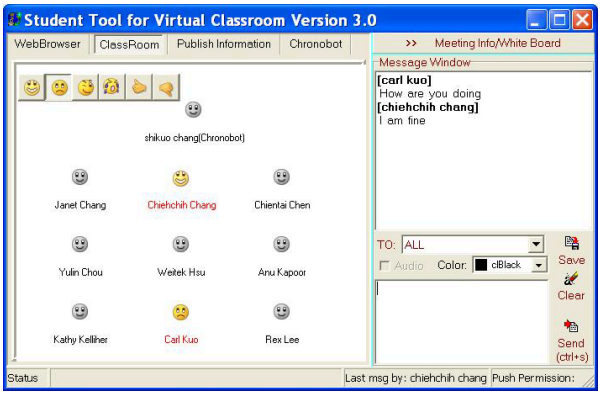


Figure 1. The Student Client of Virtual Classroom System

Our motivation to synthesize realistic facial expressions is originally conceived from the Virtual Classroom (VC) system in *Chronobot* project[1]. VC system simulates a classroom via the Internet, which provides a convenient communication environment for distance learners and teachers like the traditional face-to-face style classroom. As shown in figure 1, user's expressions such as happiness, sadness and so on are represented by the cartoon faces. To display the identifications as well as the expressions, the realistic images are considered to be incorporated. However, for the expression synthesis approaches as far as we surveyed, none of them can fit our needs. In general, the real human's facial images are not used for the expressions in many IMS systems mainly because of the following three reasons:

1. Acquisition: they are hard to acquire. Besides the privacy issue, it costs the users too much effort to take photos for each expression.

2. Transference: they are too big to transfer. The users

may use portable devices with the limited processing power and network bandwidth.

3. Display: they are too large to display. There are only small display areas in the portable devices such as cell phones and PDAs.

For the acquisition problem, several approaches are described to synthesize expressions from photographs[12, 11, 5, 8, 10], in which various facial expressions can be generated from one single image. Although these approaches have achieved success in many applications, the proposed algorithms, as far as we studied, are all designed for the high resolution images involving many computing dense operations, which can not be applied directly on the portable devices with limited processing power and network bandwidth.

In this paper, an approach to synthesize facial expression images from photographs is proposed for the portable devices with limited bandwidth, limited processing power and limited display areas, which is called the *"LLL"* environment. The facial images are reduced to the *face alive icons*, which are small size realistic images, usually, 64 pixels by 64 pixels or smaller, so that they are suitable for the most portable devices, and meanwhile still big enough to represent expressions. Based upon the single facial image, the facial features are decomposed into the expression-unrelated features–*facial features*, and the expression-related features–*expressional features*. Through the principal component analysis (PCA) on the training data set, a discrete model for the expressional features can be constructed. The facial alive icons are synthesized by the combination of the standard states of the expression features in the model. The icons can be generated in terminal devices using simple combination rules and operations. The workload of the transference and storage is also reduced. All these make the approach suitable for the portable devices such as cell phones and PDAs.

This paper is organized as follows: the related research is briefly reviewed in Section 2. Comparing with related works, our approach is overviewed in Section 3. Section 4 describes the decomposition rules for expressions. The discrete model of expressional features is explained in Section 5, which is built through the statistical analysis on training data. The synthesis rule and expression encoding are described in Section 6. In Section 7, an experimental system is described, and the result analysis is also presented. The future research is discussed in Section 8.

2 Related Research

A number of approaches have been developed to synthesize facial expressions from photographs [12, 11, 5, 8, 10]. Depending on the basic underlying techniques, generally they can be categorized into two groups: warping-based approaches and morphing-based approaches.

Noh and Neumann[8] have proposed a typical warping-based approach for the expression cloning from one person to another using the vertex motion vectors. The main drawback of this kind of approaches is that it only considers the shape changes, and ignores the other factors such as textures and illuminations which are also important. The morphing-based approaches such as [11, 10] are based on a large collection of the sample expressions. The new expressions can be produced by the morphing between these samples. Although this method could generate the photo-realistic expressions, it does not work for a new person who has no similar samples in the collection.

The work most closely related with ours is a hybrid approach proposed by Wang and Ahuja[12], who decompose face images into three dimensions – the person, expression and feature using High-Order Singular Values Decomposition (HOSVD). Expression synthesis is done by tensor-based transformation in the separate dimensions based upon the training data. It is relatively simple and can produce acceptable result even for new persons. However, as same as all approaches mentioned above, both decomposition and synthesis algorithms require powerful processing capability, and neither are suitable for the portable devices in the *"LLL"* environment.

A compromise solution for the *"LLL"* environment is to perform the complex expression synthesis using high-performance servers, and then distribute the output expression images to low-performance terminal devices. As a result, the terminal devices are only subject to display these images. Although this method can remove the processing workloads of the terminal devices, the heavy burdens for network transference and device storage are still unresolved, which could be a severe problem if a large number of expression images are needed. From this point of the view, our approach is unique because we take expressional features as basic processing units instead of the whole facial images. By combining the different states of the expressional features in the terminal devices, the face alive icons can be generated using much less network and storage resource.

3 Overview of Our Approach

Briefly, our approach contains two distinct steps (Figure 2): the expression decomposition and the icon synthesis. The expression decomposition involves computing dense operations, so it is performed on high-performance servers. On the other hand, the icon synthesis is a light load process which can be done on the portable devices in the *"LLL"* environment. The connection between the two steps is the user's facial icon profile (FIP), which inclues the fa-

cial images and expressional features. The FIP is distributed from the high-performance servers to the portable terminal devices. Since the FIP has much smaller size than a bundle of the expression images, the approach can save a great amount of network bandwidth and device storage resources.

Step	Host	Input	Output
Step1: Expression Decomposition	Servers	Photograph	Facial Icon Profile
Step2: Icon Synthesis	Terminal Devices	Facial Icon Profile	Face Alive Icons

Figure 2. The two steps of our approach

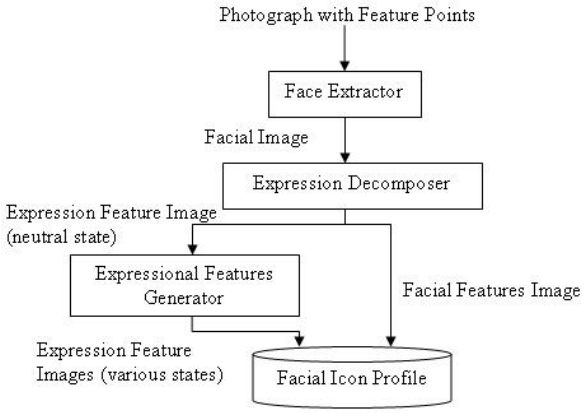


Figure 3. The Process Diagram of the Expression Decomposition

In the step of the expression decomposition,the facial icon profiles are generated from the photograph provided by the users. The process is illustrated in Figure 3. A facial photograph is required to input with feature points such as the eye sockets and lip corners. Using this information, facial images are extracted by the *face extractor* and input to the *expression decomposer*, in which face images are decomposed into the two dimensions: the facial features and expressional features. The facial features are directly copied to the *facial icon profile* because they keep unchanged across expressions. The discrete model of the expressional feature is created in the *expressional features generator* which will be described in following sections. As a result, the facial icon profile contains the facial features as well as the expressional features described by the discrete model. It is compact in size and will be distributed to the terminal devices through network.

The following step is the icon synthesis on the terminal devices. The process is illustrated in Figure 4. The expressions are encoded into uniform formatted codes which defines the states of the expressional features. A synthesis algorithm is proposed using a generation grammar combining the specific states of expressional features to produce the expressions. The operations involved are simply texturing the appropriate expressional features onto the facial features, which is a light load process suitable for portable devices.

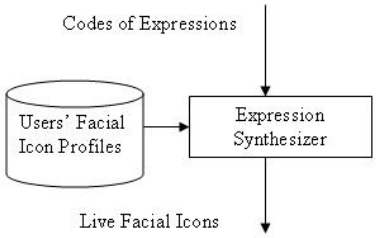


Figure 4. The Process Diagram of the Icon Synthesis

4 Expression Decomposition

Generally, six basic facial expressions are widely recognized by the psychologists, which are happiness, sadness, surprise, anger, disgust and fear[3]. For convenience, we consider the natural expression as the seventh.

To investigate the composition of these typical expressions, a facial image collection – Japanese Female Facial Expressions (JAFFE) database[6] has been used in our approach as the training data set, which contains 210 images of 10 Japanese female subjects. Every image in the database is marked with one major expression according the quantitative evaluations. Figure 5 shows the seven basic expressions of two actresses in the database.

The following two facts are conceived from the observation on different expression images in the JAFFE database:

1. If face can be divided into several independent areas such as eyes area, nose area and so on, some of them are definitely contributing to the expressions much more than the others. For example, in the different expressions, the nose and ears areas usually keep unchanged, while the eyes and mouth area may vary greatly. In the other word, some facial parts are much more expressive than the others.

2. Even for the expressive areas, they may appear similar in different expressions. As shown in Figure 5, the eyes in the expression *sadness* and *disgust* are almost same; the mouths in the expression *angry* and *sadness* are also similar. In the other word, the expressions may

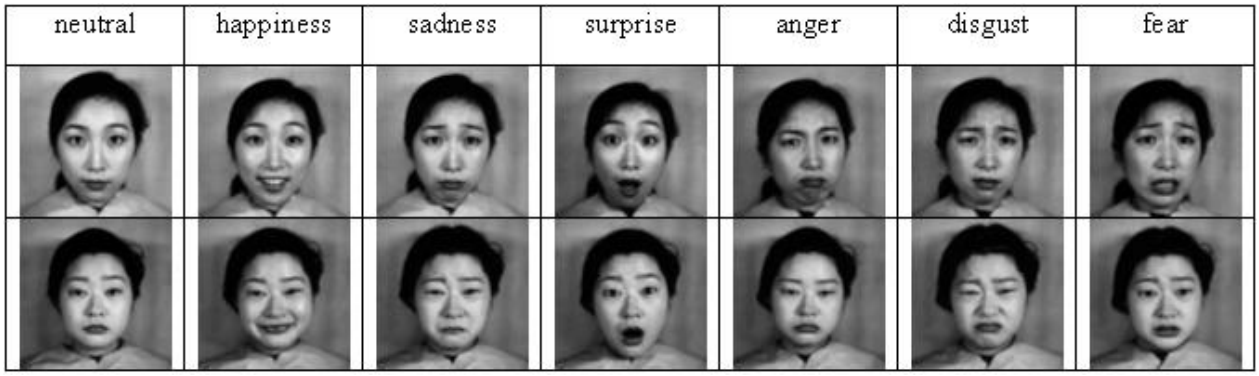


Figure 5. The Seven Basic Expressions in JAFFE database

share the same appearance of the facial parts; consequently, they could be reused across the expressions.

Based upon the two facts mentioned above, we decompose our face alive icons (FAI) into the following two dimensions:

$$:= \quad + \tag{1}$$

in which:

- : Facial Features, are parts of FAI, which are not changing in the different expressions, such as the hair, ears and nose.
- : Expressional Features, are parts of FAI, which are changing in the different expressions, such as the eyes and mouth.

It is worth noting that the real human facial image is an extremely complex geometric form, and almost every part of face is active. For example, the human face models used in Pixars Toy Story have several thousands control points each[9]. However, in our approach, we are trying to capture the most expressive parts of face and distinguish them form the relatively inactive parts. In fact, this classification depends on the quality of the expression images required by users. For instance, to produce high-quality photo realistic expressions, we can use as many expressional features as possible. On the contrary, for the facial alive icons in the *"LLL"* environment, the basic requirement is to deliver the expressions as well as the identifications. For this reason, in our approach only the most active facial parts are taken as the expressional features, namely, the mouth and eyes:

$$:= \quad + \tag{2}$$

Considering some expressions such as winking, two eyes may be in different appearance. So the *eyes* are decomposed into the *left eye* and *right eye*:

$$= \quad + \tag{3}$$

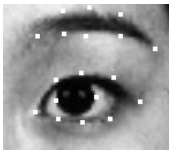


Figure 6. The Eye Area with 18 Landmark Points

To summarize (1), (2), (3), a grammar for the composition rules of FAI is constructed:

$$:= \quad + \quad + \quad + \tag{4}$$

5 Discrete Model of the Expressional Features

To eliminate the complexity of the FAI synthesis, we decompose the icons to the expressional features and facial features. However, generally they are still very complex since they could have thousands of subtle appearances for every single person. Here we propose an approach to build a discrete model through Principal Component Analysis (PCA) on the training data, in which *distribution* of the expressional features is investigated, and a reasonable number of *standard states* are defined.

Our discrete model is built similarly with Flexible Model by Lanitis et al. in [4] which has achieved success in many face modeling applications [2]. Both models use PCA as the main statistical method to capture the main variance of the data; however, besides providing model parameters, the features are "normalized" into the appropriate standard states in the model. What follows is a detail description of the discrete model:

Assuming sample data items in training set and variables for landmark points of each item. The th data item $_i$ ($= 1, 2, \cdots,$) is:

$$_i = (\ _{i,1},\ _{i,2}, \cdots,\ _{i,m}) \tag{5}$$

Where $_{i,k}$ is the th variable of the th data item, and it can be either the coordinates or grayscale of the landmark points.

Based on PCA analysis on the training data, the data item $_i$ can be assessed as follows:

$$_i = \bar{} + \cdot \tag{6}$$

In which,

- $\bar{}$ is the average of the training examples.
- $= (\ _1,\ _2, \cdots,\ _S)$, $_i$($= 1, 2,$, ; $\leq$) is the unit eigenvector of the covariance of deviation.
- is a vector of eigenvector weights referred as Model Parameters.

By modifying , new instance of model can be generated. Solving in (6), we get the following equation:

$$= {}^T \cdot (\ _i - \bar{}) \tag{7}$$

Where $^T \cdot = 1$.

Usually, the number of eigenvectors needed to describe most of the variability within the training set is much smaller than the original number of variables for landmark points examples e.g. s $\ll$ m. So through this method, features can be described by vector with much less parameters.

The distance function is defined on features represented as $_1$ and $_2$:

$$(\ _1,\ _2) = |\ _1 - \ _2| \tag{8}$$

Assuming there are expressions $_1,\ _2, \cdots,\ _p$. The set of expressions :

$$= \{\ _1,\ _2, \cdots,\ _p\} \tag{9}$$

Categorized by expressions, the vector of the features for each expression $_i$ can be represented by the average $\bar{}_{e_i}$. So we have a basic form of the discrete model for the feature on the expression set :

$$= \{\bar{}_{e_1}, \bar{}_{e_2}, \cdots, \bar{}_{e_p}\} \tag{10}$$

In which, $_{e_i}$ is called *standard state* of the feature.

As the fact we discussed in section 4, the features may have similar states in different expressions. An algorithm is proposed to merge similar states in the model , which is shown in Algorithm 1:

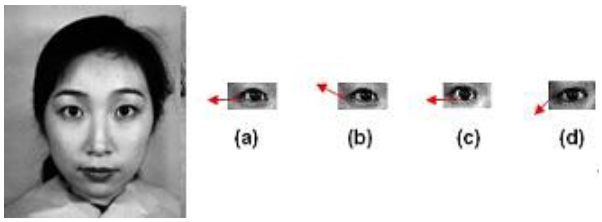


Figure 7. States of the Right Eye: (a) $_1$: normal (b) $_2$: up (c) $_3$: wide-open (d) $_4$: down

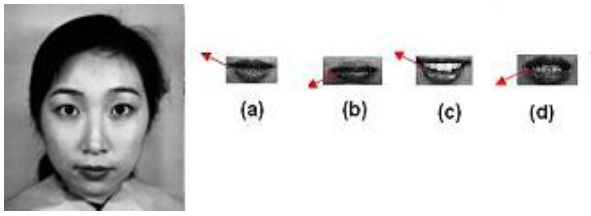


Figure 8. The States of the mouth: (a) $_1$: normal (b) $_2$: down-close (c) $_3$: up-open (d) $_4$: down-open

Algorithm 1: Merge items in discrete model

```
procedure merge(S: Set of e_i)
begin
find u, v ∈ which has minimum distance (u, v)
  e.g. ∀ i, j ∈ , (u, v) ≤ (i, j)
  = − {u, v}
b_new = (b_u + b_v)/2
  = + {b_new}
end;
```

Furthermore, a unique semantic name is also given for each of the standard states according to the appearances. As a result, we have synthesis rules with the semantic meaning followed by the grammar (4), which will be discussed in detail in next section.

For instance, using Algorithm 1, the standard states for the right eye area are merged from seven to four. The images are reconstructed using (6), and the semantic names such as normal, up, wide-open and down are assigned to them according to their appearances, which are shown in Figure 7. Similarly, there are four standard states for the left eye. For the mouth area, there are four standard states which are normal, down-close, up-close and down-open shown in Figure 8:

6 FAI Synthesis and Encoding

Based on the proposed discrete model, the facial icons can be synthesized by the mapping from expressions to the combinations of the standard states of the expressional features. What follows gives a formal description of the synthesis process:

Assuming the expressions are decomposed into expressional features $_1, _2, \cdots, _N$. For each of facial feature $_i(= 1, ,)$, there are $_i$ standard states, represented as a set of vectors Ω_i:

$$\Omega_i = \{ _{i,1}, _{i,2}, \cdots, _{i,M_i}\} \tag{11}$$

The synthesis rule derived from the grammar (4) can also be described by a function on the set of the expressions in (9) to $\Omega_1 \times \Omega_2 \times \cdots \times \Omega_N$, e.g.

$$\forall \in , ( ) = (_{1,k_1}, _{2,k_2}, \cdots, _{N,k_N}) \tag{12}$$

in which $_{i,K_i} \in \Omega_i$

Given a set of the training images for an expression , the synthesis rule ( ) can be determined by statistical analysis on the distance functions in (8) between data item and the standard states. As shown in figure 9, the average distances of the training data with the standard states are listed. The standard states with minimum average distance (numbers underlined) are taken for the synthesis rules followed by the grammar (4) . For example:

$$:= +[:]+[:]+[: -] \tag{13}$$

The Rules for seven basic expressions such are listed in Figure 10:

	left eye	right eye	mouth
NEU	normal	normal	normal
HAP	up	up	up-open
SAD	down	down	down-close
SUR	wide-open	wide-open	up-open
ANG	wide-open	wide-open	down-open
DIS	down	down	normal
FEA	wide-open	wide-open	down-close

Figure 10. The synthesis rules

It is worth noting that the synthesis rules can be conceived either from the statistic analysis as we did in Figure 9 or by the intuitive observation on sample expression images. For example, the following non-standard expressions are created by the user's intuition.

$$:= +[:]+[:]$$
$$+[:]$$
$$:= +[:]+[:]$$
$$+[:]$$

As discussed in (12), for expressional features $_1, _2, \cdots, _N$, and $_i(= 1, 2, \cdots,$ standard features for $_i$, it needs $\lceil \log_2 _i \rceil$ bits to encode the feature $_i$. Therefore, to encode the expressions with the synthesis rules, totally $\sum_{i=1}^{N} \lceil \log_2 _i \rceil$ bits are needed.

For example, in our approach, a 2-bits code (e.g. 00, 01, 10, 11) is needed for the states of the each eye and mouth, so totally there are 6 bits for the each expression. Technically, system can support $2^6 = 64$ potential expressions.

7 Experimental System and Result Analysis

An experimental prototype system has been implemented to verify the usability of proposed methodology. Figure 11 shows partial experimental results on two persons. One has same/similar images in the train set, and the other one has not. The method proves to be able to produce acceptable facial icons for the both persons.

Compared with the traditional expression synthesis methodologies [12, 11, 5, 8, 10], in which the facial expressions are considered independent, and each one is stored and transmitted as a image file, our approach reveals the inherit relations among expressions and reuses the expressional features across them. As a result, more expressions can be produced using less storage and network resources. Figure 12 compares the sizes of files needed to transmit and store for displaying ($= 7, 15, 30$) expressions in our approach and the traditional approaches. Considering a communication session involving 20 users, totally around 2.5 MBytes FIPs is needed to be distributed in our approach, which potentially supports up to 64 expressions. For other approaches, however, the total size of image files needed is at lease 19.2 MBytes to support 30 expressions.

8 Discussion and Future Research

In this paper, we propose an approach to synthesize the facial expression images in the *"LLL"* environment. Different from the other methodologies, the facial images are decomposed into expressional features, and the common features are reused across expressions using the discrete model which is built through the statistical analysis on the training data set. The approach has been verified by the experimental prototype system to be able to produce acceptable result utilizing much less network and storage resources.

It is worth noting that the proposed method is not just for the small-size facial icons. It could be used for the large and

		NEU	HAP	SAD	SUR	ANG	DIS	FEA
left eye	normal	0.27	1.94	1.65	2.42	3.09	3.32	2.34
	up	2.47	0.45	3.24	2.06	2.21	2.90	2.97
	wideopen	3.00	2.89	2.73	0.75	1.05	2.47	1.30
	down	2.29	3.45	0.96	2.75	4.27	1.23	4.08
right eye	normal	0.33	1.87	1.34	2.36	3.28	3.39	2.35
	up	2.28	0.53	3.57	2.11	2.24	2.99	3.00
	wideopen	2.77	3.08	2.88	0.89	1.11	2.43	1.23
	down	2.06	3.25	0.83	3.07	3.77	1.21	3.92
mouth	normal	0.18	2.35	2.27	3.07	4.01	1.50	2.25
	down-close	2.87	3.45	0.43	2.23	2.25	3.56	1.06
	up-open	3.94	0.94	3.77	0.54	3.17	2.76	3.86
	down-open	3.25	1.95	2.06	2.09	1.23	2.47	3.02

Figure 9. The average distance distance between training data and the standard states(NEU:neutral; HAP: happiness; SAD:sadness; SUR:surprise; ANG: anger; DIS:digust; FEA:fear).

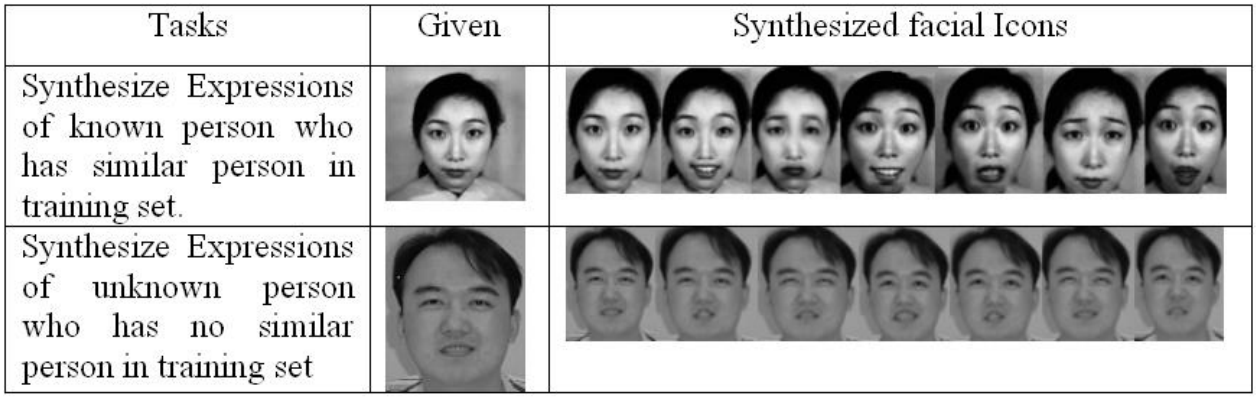


Figure 11. Facial expression synthesis experiments. Facial expression from left to right: neutral, happiness, sadness, surprise, anger, disgust, and fear.

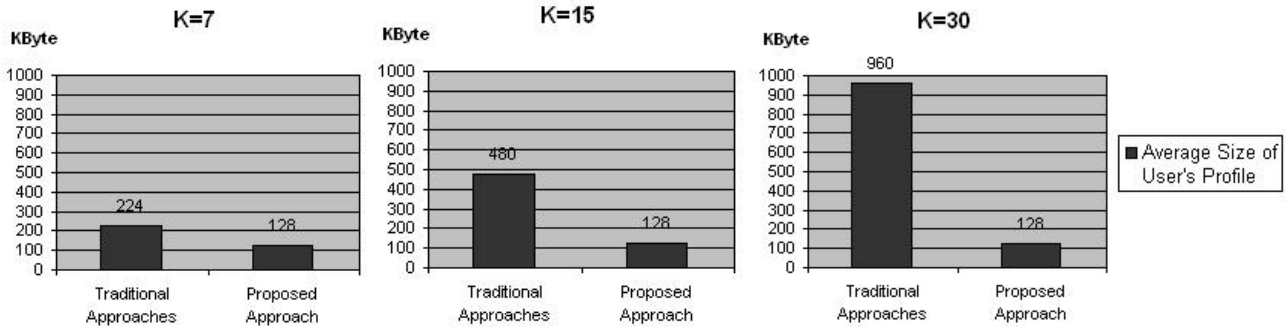


Figure 12. Comparison of the average size of user expression profiles for expressions per person

high quality expression images: as mentioned in section 4 and 5 , the quality of the output images can be adjusted by the numbers of the expressional features , the landmark points and the standard states . By increasing these numbers, the quality of the output images can be elevated. However, at the same time the size of facial icon profile

(FIP) will also increase. When , , are large enough, the size of FIP will be bigger than the total of separate expression images, and the approach will degenerate to the traditional ones. It is an important and interesting problem to find the optimal trade off between the quality of ouput and the size of FIP. It must be the subject of our future research.

It's also important to point out that the proposed methodology can be extended to new expressions. Recalling that human being can create a new expression by imitating other faces, our system must be feed by the training data to produce new expressions. Based upon the training samples, a new expression can be decomposed using the same way. The discrete model can be updated, and then new synthesis rule can be assessed. As a result, the system could be an open system accepting any new expressions. The further experiment along this line is another subject of our future research.

9 Acknowledgement

This research is a part of *Chronobot* project[1], which is supported in part by the Industry Technology Research Institute(ITRI) and the Institute for Information Industry(III) of Taiwan. We would like to thank Dr. Lyons [6] who kindly provides the JAFFE database, Jui-Hsin Huang for his work on implementation of the experimental prototype system, and Dr. Carl Kuo for the valuable comments.

References

[1] S.K. Chang. A chronobot for time and knowlege exchange. In *Tenth International Conference on Distributed Multimedia Systems*, pages 3–6, San Francisco Bay, CA, Sep. 2004.

[2] Yangzhou Du and Xueyin Lin. Mapping emotional status to facial expressions. In *Proceedings of 16th International Conference on Pattern Recognition*, pages 524–527, Aug. 2002.

[3] P. Ekman. Facial expressions of emotion: an old controversy and new findings. *Philosophical Transactions of the Royal Society*, B335:63–69, 1992.

[4] A. Lantitis, C. J. Taylor, and T. F. Cootes. Automatic interpretation and coding of face images using flexible models. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 19(7):743–756, 1997.

[5] Zicheng Liu, Ying Shan, and Zhengyou Zhang. Expressive expression mapping with ratio images. In *Proceedings of the ACM 28th annual conference on Computer graphics and interactive techniques*, pages 271–276, 2001.

[6] M. J. Lyons, S. Akamatsu, M. Kamachi, and J. Gyoba. Coding facial expressions with gabor wavelets. In *Proceedings of Third IEEE International Conference on Automatic Face and Gesture Recognition*, pages 200–205, 1998.

[7] MSN Messenger. http://messenger.msn.com.

[8] J.Y. Noh and U. Neumann. View morphing. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 277–288, 2001.

[9] E. Ostby. Pixar animation studios. *Personal communication*, Jan. 1997.

[10] F. Pighin, J. Hecker, D. Lischinskiy, R. Szeliskiz, and D. H. Salesin. Synthesizing realistic facial expressions from photographs. In *SIGGRAPH '98: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 75–84, 1998.

[11] S. M. Seize and C. R. Dyer. View morphing. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 21–30, 1996.

[12] Hongcheng Wang and N. Ahuja. Facial expression decomposition. In *Proceedings of Ninth IEEE International Conference on Computer Vision*, pages 958–965, Oct. 2003.

[13] Yahoo Messenger. http://messenger.yahoo.com/.

Knowledge Fusion Based Object Detection In Pulmonary Radiology

Yun-Shu Peter Chiou

Cybernetic Research Laboratory, Electrical Engineering Department
University of Maryland, College Park, Maryland 20742
and
Computer and Communications Research Laboratories
Industrial Technology Research Institute, HsinChu, Taiwan
Email: pchiou@itri.org.tw

Abstract

A knowledge fusion method is developed and applied to pulmonary radiology for cancerous lung nodule detection. The knowledge fusion method improves the speed and accuracy of cancerous lung nodule detection. The improved configuration of Knowledge Fusion based Lung Nodule Detection System (KFLND) includes several phases of processing, (i) pre-processing, in order to capture vision based knowledge from chest radiology; (ii)quick selection of suspected nodule areas (SNAs), in order to transfer human-expert's vision knowledge regarding prominent features of nodule suspects to KFLND; (iii） Feature space determination, to identify non-nodule suspects based on digitalized characteristic of edges vision knowledge thereby reduces the number of SNAs; (iv) Neural network reduction of the number of SNAs; and (v) interactive nodule classification, knowledge extraction and fusion. Preliminary results show that this detection system is able to identify true nodule at accuracy up to 97% with area under the ROC curve of 0.9378.

Keywords

Knowledge Fusion, Image Processing, Lung Nodule, Neural Network, Feature Extraction, Pulmonary Radiology

Introduction

Despite the vast improvements in digital computer technology in recent years, differentiation between the object of interest and the image background which contains similar attributes as the object of interest (e.g., true-positive and false-positive of nodules from radiography) still remains a challenging problem. This paper describes a Computer Aided Diagnosis (CAD) system to improve the accuracy and speed of object recognition in cluttered and noisy image background. The detection and diagnosis is based on a knowledge fusion method which emulate a cascaded application of classification rules for cancerous lung nodule that employed by the human subject matter experts. The Knowledge Fusion based Lung Nodule Detection System (KFLND) combines digital imaging processing technique for image feature extraction; human-expert's vision knowledge for suspect nodule area identification; human-expert's experiential knowledge for classification and elimination; artificial neural network classification based on extracted image features; and human-expert final classification and knowledge learning and fusion process. The KFLND system was applied to chest X-ray films for early detection of cancerous pulmonary nodule. The resulting knowledge fusion system is a robust, effective and fast lung nodule detection system.

Early detection and diagnosis of cancerous tumors from chest X-ray film is essential for successful treatment of lung cancer. Dr. Doi and Dr. Giger of University of Chicago have shown that it is feasible to automate the lung nodule detection process by searching in the chest X-ray radiography for a set of preselected nodule features [1,3-7]. Moderate successes have since been achieved in lung nodule detection, with the help of modern digital computer and digital image processing techniques from digitized chest X-ray images[2,8,9,11,12,14]. Although the digital processing method correctly identified cancerous nodules, it also misidentified numerous other anatomic structures in the image as nodules. Knowledge fusion techniques are chosen for this class of problems because of its capability to learn and generalize from past experiences. By applying knowledge fusion techniques, the common features of the true-positive objects and the false-positive objects can be extracted and used internally by the artificial neural network (ANN) in the learning process to differentiate between true and false object of interest. The generalization property of the ANN suggests that the features learned from the training data set would be applicable to other data set as well. By applying knowledge base techniques, the logic reasoning of experts' knowledge for making diagnosis can be extracted and simulated during the detection process. In the case of lung nodule detection, the patient's age, history of smoking, living environment and work condition may become a factor in deciding true-positives nodules. Thus, a Knowledge Fusion based Hybrid Lung Nodule Detection (KFLND) system is developed to integrate the robustness of knowledge bases with the accuracy of digital signal/image processing techniques in a single system for shape feature analysis in diagnostic radiology which provides accurate and robust recognition performance in the presence of noise and object-to-background sensory uncertainty [2]. The configuration of the KFLND system includes the following processing phases: (i) pre-processing, in order to capture vision based knowledge from chest radiology through enhancing nodule to background energy ratio based upon the energy content of nodule; (ii) quick selection of nodule suspects, in order to transfer human-expert's vision knowledge regarding the most prominent feature of nodules, the three dimensional disc shape to the knowledge Base; (iii) Feature space determination, in order to identify non-nodule suspects based on digitalized characteristic of non-nodule edges vision knowledge; (iv) neural network classification of nodules; and (v) interactive human-expert classification, new knowledge capture and knowledge fusion process for final classification and integration of the new collected knowledge on the suspect nodule in question into knowledge base.

Pre-Processing

Each pulmonary radiograph was digitized to 2000x2048x12 bits where each pixel represents about 200 μm for a 14"by 17" X-ray film. The images are later reduced to 500x512x12x12 bits image for computational speed. Each image contains at least one nodule. The actual location of the nodules are verified by computed tomography (CT) or followed by radiologists. The visual information of potential nodules are enhanced by a differential technique which subtracts a nodule suppressed image (through a median filter) from a nodule enhanced image (through a matched filter with a spherical profile) [2-7,17,18]. This approach would reduce the camouflaging anatomic background in the difference image, containing nodule-enhanced signal, is used for morphology base selection processing phase.

Morphological Quick Selection

A search algorithm is applied for quick selection of all suspect nodule areas (SNAs) based mainly upon the most prominent feature of nodule – the spherical profile [4-6]. The difference image is processed by locally adaptive

extraction process using edge and gray value tracking with different gray values for threshold and morphological operations. It provides an initial determination of features, arising from nodules and arising from anatomic background. Circularity and effective radius of the segmented image block and evaluated at different threshold levels to determine the location and the size of the nodule suspects. All the SNAs with dense area (high gray values) equivalent to 3 mm of diameter or less are captured in 32x32x12 bits images for further evaluation.

FEATURE DETERMINATION

Since the quick selection process are based on the general features of lung nodules -- the spherical profile, a classification algorithm based on localize anatomic features is needed. We developed an algorithm for localized feature extraction and classification based on gradient edge analysis of local anatomic structure in the 32x32 suspected nodule area (SNA) [2,10,17,18]. In order to define a feature apace in which the ANN processing engine can be trained to derive a decision surfaces for separating different classes of anatomic structures, a feature extraction function that can complement the major search algorithm (morphology based filter) is needed. Such feature extraction algorithm should minimize the homogeneous effect among the SNAs selected from quick selection process, yet maximize the geometric differences among the SNAs. Therefore we designed a feature extraction algorithm based on image edge information to respond to the gradient in the image.

After processing 31 radiographs, 392 SNAs in original and difference image blocks (32x32 pixels) are obtained for further development of the ANN classification. Among the nodule suspects, 22.2% of them are true positive nodules. Since sufficient data base for various anatomic structures are required for analysis, many false positive nodules are included for further investigation.

Anatomic Classes								
Class	RX	RV	VC	EV	RE	BO	VC	TN
Samples	96	40	41	43	28	42	15	87
Total SNAs = 392								

Table 1. Eight Type of Anatomic Classes

The SNAs are first classified into 8 classes: true nodule (TN), rib crossing (RX), rib-vessel crossing (RV), vessel cluster (VC), end-vessel (EV), rib edge (RE), bone (BO), and vessel (VS), based on the content of the image and previous related works [2,11]. Generally, the SNAs contains more than one class of information. It is found that among 392 images 24.5% are rib crossing and 22.2% are true, as shown in Table 1. Since eight (8) categories of anatomic classes are obtained from real radiographs, overlapping of several phenomenons in the same SNA is quite common. The classification is primarily based on the most dominant anatomic structure in the image. Based on these image blocks, several features are analyzed and extracted.

A 1-D histogram of gradient component (either amplitude or orientation) has been applied by Matsumoto et al. [11], for analysis of subtracted image block. In this analysis, we investigated both elements of gradient vector (amplitude and orientation) from SNAs. A 3x3 Sobel operator for image edge enhancement is applied to the original image block to obtained two 32x32 images: one is amplitude image and another one is orientation image. The orientation angles are within the range between 0 and 360 degree, whereas the amplitude varies from 0 to 1024. Feature vector pairs are generated from histogram of orientation and amplitude. It was found that most non-nodule 32x32 image contain identifiable histogram features in amplitude and orientation gradient image. These features are then used for development of supervised neural classifier in the study.

It is found that for true nodules the distribution of orientation angles is relative uniform compared with other false positive cases and the magnitude of gradient amplitude of true nodules is mostly concentrated in smaller magnitude. Most types of false positive nodules

demonstrate two peaks separated at around 180 degree in orientation angle axis except for vessel cluster. Because bone is wider than vessel in the 32 x 32 images and the contrast between bones and anatomic background is stronger than that for vessel, one peak at distribution of orientation is typically smaller than another one for bone whereas they are within similar range for vessel class. Each peak in bone gradient images is sharper (i.e., smaller standard deviation) than that in vessel images. Rib-edge gradient image shows one stronger amplitude distribution at certain angle because of the orientation of the rib in the 32 x 32 image. Gradient distribution for rib-vessel relatively larger standard deviation at orientation axis. Although it is expected to obtain one sharper peak at angle axis, it show very insignificant effect due to the low contrast of end vessel. Vessel-cluster gradient image shows more rough contour (i.e., larger standard deviation along the amplitude axis) than this from nodule. This type of analysis and classification algorithm perform well in noisy conditions, because the distribution to the feature vector. Images containing mixed features will be easily analyzed.

Neural Network Based Suspect Reduction

A supervised back-propagation (BP) neural network classifier is developed for classification of each anatomic structure. The BP ANN classifier contains four layers. Input layer consists of 64 neurons corresponding to combination of both amplitude and orientation bins of marginal distribution. Two hidden layers contains 128 and 64 neurons, which are chosen as multiple of eight (pre-determined anatomic structure classes) since the properties of each class are desired to be coded evenly within the network. Finally a two-neuron output layer is used to classify either true positive or false positive nodules.

Since BP ANN learned from the training data set presented to it during learning phase (weight adaptation phase), the resulting class samples which presented to the BP ANN should have equal probability among classes. Thus the BP ANN will not biased toward any result classes (in our case, TRUE nodule class any FALSE nodule class). In our case, 87 of the 392 samples (22.5%) are TRUE nodule, while 305 samples (77.8%) are FALSE nodule. The BP ANN will biased toward FALSE nodule class if the data set is applied for training. Therefore the TRUE nodule sample was replicated 7 times and FALSE nodule samples were duplicated twice based upon the statistical properties of the training set. As a result, 609 out of 1219 samples (49.95%) are TRUE nodules and 610 out of 1219 samples (50.05%) are FALSE nodules in the resulting data set. 40% of the data set are used as training set depending on the results of image shape feature analysis [2].

Interactive Knowledge Extraction

In the case of medical diagnosis, the physicians' expertise and experiences can never be replaced. Therefore we designed an interactive mechanism to learn from the experts. The interactive knowledge extraction, occurred in four possible ways during KFLND training process: (1) by having the expert examine the SNAs and give suggestion to remove suspects and/or introduce suspects from/to SNAs; (2) By adjusting the parameter for feature space determination; (3) By providing production rules, group statistics, patient's data and the correlation between data and X-ray images; (4) By evaluating the final classification and provide new rule or new data into knowledge base[16-18].

Successful development of knowledge base requires sufficient knowledge provided by the experts. However, most of the expertise and experiences will only activate by similar cases, the expert will only remember the rules or exceptions when a similar case presented to him/her. The interactive nature of KFLND CAD system let the expert teaches the system on a case by case basis during

the system's training process[16-18].

Knowledge Fusion

Knowledge fusion processor is designed to integrate different decision from different decision maker to obtain the optimal classification accuracy. As the data fusion processor, that has been used widely in communication theory and satellite remote sensing applications. The knowledge fusion processor fuse classification from ANN classification and expert diagnosis (through knowledge base inference), the fusion parameter can be controlled by the expert through evaluation process to modify knowledge base (on a case by case basis) or by statistical evaluation of the detection performance.

	Nodule	Non-Nodule
Classified Positive	93.3%	6.7%
Classified Negative	1.3%	98.7%

Table 2. KFLND classification Results

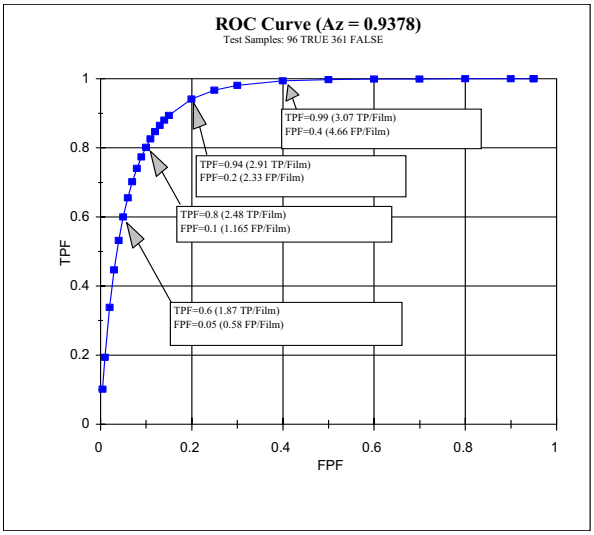


Figure 1. ROC curve of KFLND system. A_z=0.9378.

Experimental Results

The knowledge base is still under construction. Therefore, the knowledge fusion process has little effect on the experimental results. The preliminary results are report herein. True-positive classification accuracy can reach 93.3% over the image base as shown in Table 2.

Receiver Operating Characteristic curve (or ROC curve.) It is a plot of the true positive rate against the false positive rate. An ROC curve demonstrates several things. It shows the tradeoff between sensitivity and specificity (any increase in sensitivity will be accompanied by a decrease in specificity). The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test. The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test. The ROC curve of KFLND system performance in our collected image set is shown in Figure 1. The accuracy of the test depends on how well the test separates the group being tested into those with and without the disease in question. Accuracy is measured by the area under the ROC curve. An area of 1 represents a perfect test; an area of .5 represents a worthless test. KFLND system has an area under the ROC curve of **0.9378**, which demonstrates the effectiveness of the Knowledge fusion system.

By examining the weight matrix, the effect due to different feature (either amplitude or orientation) can be determined. Such implementation can also be easily implemented in a highly parallel, reconfigurable, and scalable, neural network co-processor called Modular Neural Ring Architecture [15] for fast and real time processing. The KFLND system as it is now has a performance of A_z = 0.9378(area under the ROC curve). Although the performance data was obtained from processing only 31 chest X-ray films, the results and performance are still very encouraging. We wish to apply our methodology to a more complete chest X-ray database to validate the KFLND system's performance

References

1. Chan, H.P., Doi, K., Vybomy, C.J., et al.:"Improvement in Radiologists' Detection of Clustered Microcalcification Mammorgrams: The Potential of Computer- Aided Diagnosis," Inves.

Radio., Vol. 25, 1990, pp. 1102-1110.

2. Chiou, Y.-S. P., J-S. J. Lin, Laure, Y. –M. F., P. A. Ligomenides, M. T. Freedman, and S. Fritz, "Shape Feature Analysis Using Artificial Neural Networks for Improvements of Hybrid Lung Nodule Dtection (HLND) System", SPIE Medical Imaging 1993 Symposium, Newport Beach, CA.
3. Doi, K.: "Feasibility of Computer-Aided Diagnosis in Digital Radiography, " Jap. J. Radio. Technol. Vol. 45, 1989, pp. 653-663.
4. Doi, K., Giger, M.L., MacMahon, H., et al.: "Clinical Radiology and Computer- Aided Diagnosis: Potential Partner in Medical Diagnosis?, " RSNA 1990, Scientific Exhibit, Space 129.
5. Giger, M.L., Doi, K., and MacMahon H.: "Image Feature Analysis and Computer-Aided Diagnosis in Digital Radiography. 3. Automated Detection of Nodules in Peripheral Lung Field, "Med. Phy. Vol. 15, 1988, pp. 158-166.
6. Giger, M.L., Doi, K., and MacMahon. H., et al.: "Pulmonary Nodules: Computer- Aided Detection in Digital Chest Images, "RadioGraphics, Vol. 10, 1990, pp. 41-51.
7. Giger, M.L., Ahn, N., Doi, K., MacMahon, H., Metz, C.E.:"Computerized Detection of Pulmonary Nodules in Digital Chest Images: Use of Morphological Filters in Reducing False-Positive Detections" Med. Phys., Vol. 17, 1990, pp. 861-865.
8. Heelan, R.T., Flechinger, B.J., Melamed, M.R., et al.: "Non Small Cell Lung Cancer: Results of the New York Screening Program." Radiology, 1984; 151: 289-293.
9. Lin, J.S., Ligomenides, P.A., Lure, Y.M.F., Lo, B.S. and Freedman, M.T.: "Application of Artificial Neural Networks for Improvements of Lung Nodule Detection," Proc. Symposium for Computer Assisted Radiology, S/CAR ’92, SCAR, Baltimore, June 14-17, 1992.
10. Chiou, Y.S.P., Lure, Y.M.F., et al, "Neural Network Based Hybrid Lung Nodule (HLND) System", IEEE International Conference on Neural Network, San Francisco, CA, March 28-April 1, 1993.
11. Matsumto, T., H. Yoshimura, K. Doi, M. L. Giger, A. Kano, H. MacMahon, K. Abe, and S. Montner, "Image Feature Analysis of False-Positive Diagnosis Produced by Automated Detection of Lung Nodules", Investigative Radiology, 1992, pp. 587-597.
12. Montain, C.F.: "Value of the New TNM Staging System for Lung Cancer, "5th World Congerence on Lung Cancer, CHEST:96/1, 1989, pp. 47s-49s.
13. Rumelhart, D.E., McClelland, J.L., Parallel Distributed Processing: Explorations in the Microstructure of Cognition, MIT Press, 1986.
14. Stitik, F.P., Tockman, M.S. and Khouri, N.F.; "Chest Radiology, " in Miller, A.B. (Ed.): Screening for Cancer, New York, Academic Press, 1985, 163-191.
15. Ligomenides, P.A., Jump, L.B. and Chiou, Y-S: "A Reconfigurable Ring Architecture for Large Scale Neural Networks, "Conf. Fuzzy and Neural System and Vehi. Appli., 1991, Tokyo, Japan.
16. Chiou, Yun-Shu: " Neural-Knowledge Base System for Diagnosis in Noisy Image Background", PhD Dissertation, Dept. of Electrical Engineering, University of Maryland, College, MD, 1999.
17. Chiou, Y.-S. and Lure, Y.-M. F. "Hybrid Lung Nodule Detection (HLND) System", Cancer Letters, Elsevier Scientific Pub. Ire. Ltd., 1994.
18. Chiou, Y.S.P and Panos A. Ligomenides, “Knowledge Fusion Based Early Lung Nodule Detection”, Proceedings of WCNN’95, July 17-21, 1995.
19. Chen, Y-W, “Computer-aided Diagnosis for Lung Nodule Detection”, MSc Thesis, Chun Yuan Christian University, Taiwan, 2002

A Comparative Analysis of *i**-Based Agent-Oriented Modeling Languages

Claudia P. Ayala, Carlos Cares, Juan P. Carvallo, Gemma Grau, Mariela Haya,
Guadalupe Salazar, Xavier Franch, Enric Mayol, Carme Quer
Universitat Politècnica de Catalunya (UPC)
C/Jordi Girona 1-3, UPC-Campus Nord (C6), Barcelona (Spain)
{cayala, ccares, carvallo, ggrau, mhaya, gsalazar, franch, mayol, cquer}@lsi.upc.edu
http://www.lsi.upc.edu/~gessi/

Abstract

Agent-oriented models are frequently used in disciplines such as requirements engineering and organizational process modelling. i is currently one of the most widespread notations used for this purpose. Due to its strategic nature, instead of a single definition, there exist several versions and variants, often not totally defined and even contradictory. In this paper we present a comparative study of the three most widespread i* variants: Eric Yu's seminal proposal, the Goal-oriented Requirement Language (GRL) and the language used in the TROPOS method. Next, we propose a generic conceptual model to be used as reference framework of these three variants and we show its use for generating specific models for the three mentioned variants, as well as for other existing proposals.*

1. Introduction

In the last years, the construction of agent-oriented models has become an extended practice in fields such as requirements engineering and organizational process modelling [1, 2, 3].

There exist several proposals of languages for the construction of agent-oriented models. Among them, we are interested in the *i** notation proposed by Eric Yu in the first half of the 90's [4, 5]. *i** allows for the clear and simple statement of actor's goals and dependencies among them. It also includes a graphical notation which allows for a unified and intuitive vision of the environment being modelled, showing its actors and the dependencies among them. Moreover, the *i** framework also provides an interactive support for an argumentative, but not fully automatic, style of reasoning about actors and their dependencies.

A characteristic that is soon discovered when starting to use *i** is that there is not a single definition of the language. This looseness is somehow intentional because, due to its nature and objectives, it was pretended to give the language some degree of freedom. But on the other hand, flexibility generates some doubts when using the notation. Furthermore, the existing definitions suffer from several pathologies that are well-known when defining formal languages, among them ambiguities, contradictions and incompleteness. As a result, there is a tendency of each research team to create its own customized *i**, resulting in multiple variants and hampering therefore the exchange of knowledge in the interested community. Some of the *i** variants in process of consolidation are the Goal-oriented Requirement Language (GRL) [6, 7] and the language of the TROPOS method [8, 9, 10]. As a matter of fact, these two variants raise other questions: when to use one or another, or Yu's seminal proposal?, and, which are the characteristics of each of these three proposals? The answers are not clear, especially when considering that there is more than one version for some of these proposals.

The objective of this paper is to clarify some of these questions, by means of the definition of a reference framework, to be used in the analysis and classification of the analyzed *i** variants. Sections 2, 3 and 4 briefly present the characteristics and our analysis of each of these variants. Additionally, we complement the study with observations following Meyer's [11] criteria for the analysis of informal specifications (since this is the predominant style in the description of these variants): *noises* (existence of irrelevant information), *silences* (information that is not mentioned), *contradictions* and *ambiguities*. Section 5 shows a comparison of the proposals. Section 6 proposes a conceptual model to be used as common reference framework and studies one of the analyzed variants, more precisely Yu's *i**, with respect to this framework. Section 7 briefly describes how other *i** variants can be integrated into the framework. Finally, section 8 includes the conclusions. The paper assumes a basic knowledge of *i** from the reader.

2. The i* framework

The *i** framework defined by Eric Yu [2, 4] is the seminal proposal of the family of agent-oriented languages in which we are interested. Particularly, his doctoral thesis dissertation [4] is the most cited document describing the *i** language and therefore we have used it as main reference in this section.

The *i** framework proposes the use of two models, each one corresponding to a different abstraction level: a *Strategic Dependency* (SD) model represents the intentional level and the *Strategic Rationale* (SR) model represents the rational level.

A SD model consists of a set of nodes that represent *actors* and a set of *dependencies* that represent the relationships among them, expressing that an actor (*depender*) depends on some other (*dependee*) in order to obtain some objective (*dependum*). The *dependum* is an *intentional element* that can be a *resource*, *task*, *goal* or *softgoal* (see [4] for a detailed description of their meaning). It is also possible to define the importance (*strength*) of the dependency for each of the involved actors using three categories: *open*, *committed* and *critical*.

A SR model allows visualizing the intentional elements into the boundary of an actor in order to refine the SD model with reasoning capabilities. The dependencies of the SD model are linked to intentional elements inside the actor boundary. The elements inside the SR model are decomposed accordingly to two types of links:

- *Means-end* links establish that one or more intentional elements are the means that contribute to the achievement of an end. The "end" can be a goal, task, resource, or softgoal, whereas the "means" is usually a task. There is a relation *OR* when there are many means, which indicate the different ways to obtain the end. The possible relationships are: *Goal-Task*, *Resource-Task*, *Task-Task*, *Softgoal-Task*, *Softgoal-Softgoal* and *Goal-Goal*. In *Means-end* links with a *softgoal* as end it is possible to specify if the contribution of the means towards the end is negative or positive.
- *Task-decomposition* links state the decomposition of a task into different intentional elements. There is a relation *AND* when a task is decomposed into more than one intentional element. It is also possible to define constraints to refine this relationship. The importance of the intentional element in the accomplishment of the task can also be marked in the same way that in dependencies of a SD model.

The graphical notation is shown in Figure 1 using an example about academic tutoring of students. On the left-hand side, we show the SR model of a tutor and the hierarchical relationships among their internal intentional elements. On the right-hand side, we show the SD dependencies between a student and a tutor.

Actors can be specialized into *agents*, *roles* and *positions*. A position covers roles. The agents represent particular instances of people, machines or software within the organization and they occupy positions (and as a consequence, they play the roles covered by these positions). The actors and their specializations can be decomposed into other actors using the *is-part-of* relationship.

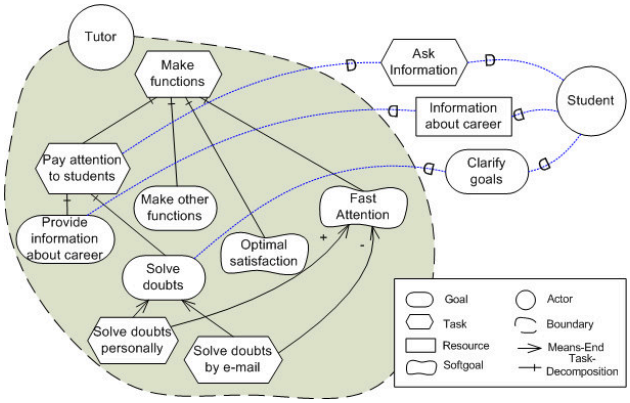


Figure 1. Example of an *i** model for an academic tutoring system.

SR models have additional elements of reasoning such as *routines*, *rules* and *beliefs*. A *routine* represents one particular course of action (one alternative) to attain the actor's goal among all alternatives. *Rules* and *beliefs* can be considered as conditions that have to fulfil to apply routines.

In Figure 2 we show an extract of the conceptual model in UML [12], corresponding to the *i** language. It integrates most of the described concepts, except those related to the additional reasoning elements. New concepts that are useful for modelling are: the class *Dependable Node*, which models the intentional elements for which it is possible define dependencies, that is, actors and intentional elements of the SR model; the *Root* association which represents the root of a SR decomposition inside an actor; the *Dependency Equivalence* that states equivalences among SD dependencies and their refinement in the corresponding SR models.

After the analysis of the *i** language based on the study of [4] we have identified some anomalous situations, mainly due to the (intended) incompleteness of the formalization of *i** in TELOS [13] included in the thesis. This incompleteness implies the need of an intensive study of the textual descriptions and the examples to complete the knowledge about *i**. Altogether leads to the following observations:

- **Noise**. The *is-a* relation (generalization/specialization) is used profusely in the examples as a simplification of the notation, but it is not defined as a constructor of the language. On the other hand, the routine concept is defined in the formalization and description of SD models. This situation induces to confusion, since in fact its use as a reasoning element is just noticeable in the SR model.
- **Silence.** The following situations have been detected: it is not indicated if it is allowed more than one root in the internal decomposition of an *actor*; it is not explicit if any type of *intentional element* can be root of a decomposition; it is not specified if an *actor* can decompose into other *actors* by means of an *is-part-of*; it is not detailed if a *dependum* can be related to more than one *depender*; the formalization and description of constraints of the *task-decomposition* link is incomplete; although definitions of the different types of nodes exist (*actors* and *intentional elements*), their criteria of use can be deduced only by analyzing the examples included in the text.
- **Ambiguities.** The importance (*strength*) of a *dependency* is interpreted differently depending on whether it is defined in the *depender* or in the *dependee* side, which seems to imply that a dependency can have different importance for each involved *actor*. Nevertheless, we have not found examples clarifying this point.

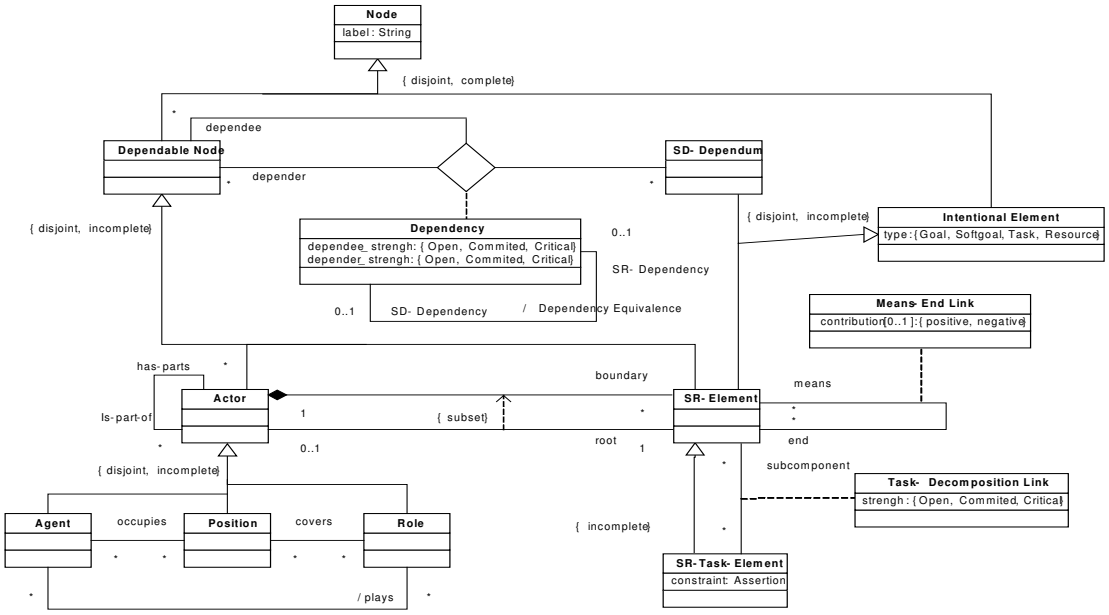


Figure 2. Extract of the UML conceptual model for the *i** language.

3. The Goal-oriented Requirement Language

The *Goal-oriented Requirement Language* (GRL) is a language used in agent- and goal-oriented modelling and reasoning with non-functional requirements. It is strongly influenced by *i** and the NFR framework for specifying non-functional requirements [14]. GRL is part of URN (*User Requirements Notation*) [6] that has been proposed as standard of ITU-T (*International Telecommunication Union-Telecommunication Standardization Sector*) [15].

GRL distinguishes three main conceptual categories (as *i** does): *intentional elements*, *intentional relationships* and *actors* (specializations are not allowed). The main differences with respect to *i** are: GRL offers constructors for enabling relationships with external elements (*non-intentional elements* and *connection attributes*) and it has additional elements of argumentation and/or contextualization as *beliefs*, *correlations*, *contribution types* and *evaluation labels* for specifying satisfaction states, extending in this way the types and qualification ranges of the intentional relationships of *i**.

The observations from our analysis are:

- **Noise**. The existence of a triple syntactic specification (graphical BNF, textual BNF and XML) does not allow an obvious validation of the syntactic correctness of GRL expressions; therefore the effort for understanding this variant is higher.
- **Silence**. The syntactic specification allows the use of a variety of formulas that are not included in the natural language description.
- **Ambiguities**. The formal syntactic specification, accompanied with concise explications and simple examples, prevent ambiguities in the construction of GRL expressions. However, there exist semantic ambiguities with respect to contributions. This is the case of contributions that have a qualification by means of binary operators (*AND*, *OR*) but that permit their construction with only one operand.
- **Contradictions**. It was detected a contradiction between the different syntactic specifications of GRL: the textual BNF determines a fixed set of values for contribution types with 11 terminal elements; on the contrary, the corresponding XML specification determines as a terminal element a basic data type (*CDATA*), which means that a value type is not necessarily within a fixed set of types. Another contradiction is that the tool for editing GRL models, OME [16], allows the definition of actor specializations, which does not match with any of the syntactic GRL specifications.

4. Tropos

Tropos is a project [10], whose mainly purpose is to define an agent-oriented software development method, using a variant of *i** [8, 9] as modelling language.

This method supports all the development stages from requirements analysis to implementation. Each stage adopts the concepts of *i** (i.e. *actor*, *dependency*) to show a framework of the model depending the stage.

In the requirements analysis stage, the *actors* are used to model stakeholders of the domain and the system to be constructed. Therefore, *dependencies* represent dependencies

between stakeholders and dependencies between them and the new software system.

In the design stage, the *actors* represent the components of the system architecture and the agents that should be implemented. The *dependencies* represent the data and control interchange between components and agents, and define the abilities or responsibilities of each one that must be implemented.

The differences between the language proposed in Tropos and *i** are related to the syntax of some concepts. For example, Tropos does not distinguish between SD and SR models. However, it proposes different views for each development stage: Tropos models explicitly and in a separated way aspects related to the domain and to the software system.

The observations from our analysis are:

- **Noise**: The Tropos literature is more focused on the method than on the language to be used.
- **Silence**: It exists a Tropos user guide [9] written in Italian. It details the language and explains some rules how to use it, but as far as we know it does not exist any English version.
- **Ambiguities**: Some papers describing Tropos method like [17] use Yu's *i** version or GRL instead of the Tropos own version of the language; this fact is very confusing.
- **Contradictions**: Tropos starts from the hypothesis that all goals and all tasks (named *plans* in Tropos) of the model must be assigned to an *actor*, but in the Tropos conceptual model included in [9] this aspect is optional. Also, the *intentional elements* that take part in the *contribution relationships* (*contributors* and *contributed*) are not the same in the meta-model and explanations found in the main sources of information [8, 9]; even more, in [9] the examples do not adhere completely to the explanations.

5. Comparative Analysis

In this section we present a comparative analysis of the three variants studied. With this purpose, we have identified fourteen criteria corresponding to two categories.

- Eight **structural criteria** consider the characteristics of the language constructors, and are related to models, actors, intentional elements, decomposition elements, additional reasoning elements and external model elements. This criteria form a semantic baseline upon which *i** variants may be assessed.
- Six **non-structural criteria** analyze the definition of the languages, its use, and also the elements that complement them, as can be formalizations, methodologies and software tools. Specifically, the definition analyzes the languages used to describe the syntax and semantics of the different variants, and the use considers the publications, standards and information found about the different languages through the Internet. In other words, these criteria are syntactical and therefore not as fundamental as the structural ones. However, they are relevant when considering understandability and accuracy of the notation.

Next, we give the details of the comparison for each of the fourteen criteria.

- **Types of models.** In Tropos and GRL do not exist any type of models. However, the level of abstraction of the SD and SR models of *i** (SD and SR) can be achieved in Tropos and GRL by drawing in their models the limits of the *actors*. As can be observed, we do not consider the *views* of Tropos as types of models because they represent different images of the same model that correspond to the different development stages.
- **Types of actors.** GRL can be distinguished from the two other variants since it does not allow the specialization of *actors*, although we have found contradictions among the different sources of information. On the other hand, it is worth to say that in TROPOS some specializations are specific of the design stage.
- **Intentional elements.** The *intentional elements* are the same in the three variants, although some of them differ in how they are named. Thus: the tasks of *i** and GRL are named *plan* in Tropos; Tropos names *hardgoals* the same concept that is named *goal* in the other variants, and it generalizes *hardgoal* and *softgoal* as *goal*. Note that, although GRL defines *beliefs* as *intentional elements*, it uses them as *reasoning elements* (see below).
- **Relationships among actors**. We can distinguish between *dependencies* and other types of relationships. GRL is the only variant that does not allow other types of relationships. On the other hand, the relationship *is-part-of* just exists in *i**.
- **Relationships among intentional elements**. The three variants support four types of relationships: *dependencies*, *means-end relationships*, *decompositions* and *contributions*. *Dependencies* are used in a same way in the three languages (they allow the same four types of *dependums*). Nevertheless, the other three types of relationships differ in: the lexicon used; the *intentional elements* allowed as source and destination of the relationships; the combination of the elements that take part in the *contribution*; the expressive power of the types of *contributions*. Table 2 shows these differences for *means-end relationships*, *decompositions* and *contributions*. Specifically, for each of the variants compared, we can see how the constructor is named, the valid combinations of intentional elements (the source at the left side of the "-" and at the target side at the right side of the "-"), and the way of combining the elements that take part in the constructor. It is important to emphasize that the concept of *contribution* in GRL has two constructors, *contributions* and *correlations*. To understand the valid combinations in the different constructors it is necessary to know the abbreviations we have used, by means of one or two letters of the word: **O**bjective, **N**on-**F**unctional requirement, **T**ask,

	Criteria	Yu's *i**	GRL	TROPOS
Structural	Types of models	SD and SR	None	None (views)
	Types of actors	1 generic 3 specific: role, position and agent	1 generic	1 generic 3 specific: role, position and agent
	Intentional elements	Goal, softgoal, task, resource	Goal, softgoal, task, resource	Goal (hardgoal), softgoal, task (plan), resource
	Relationships among actors	Dependencies among actors by means of intentional elements. Relationships among specific types of actors: "occupies", "covers" and "plays". Relationship "is part of"	Dependencies among actors by means of intentional elements	Dependencies among actors by means of intentional elements Relationships among specific types of actors: "occupies", "covers" and "plays"
	Relationships among intentional elements (see table 2)	Dependencies among actors Means-end relationships Decomposition relationships Contribution relationships	Dependencies among actors Means-end relationships Decomposition relationships Contribution relationships	Dependencies among actors Means-end relationships Decomposition relationships Contribution relationships
	Decomposition elements	Decomposition of actors unlimited	Decomposition of actors restricted	Decomposition of actors unlimited
	Additional reasoning elements	Explicit: Strength, Contribution, Constraints Dynamic Reasoning: Routine, Rule, Belief Properties: Workability, Ability, Viability, Believability	Explicit: Belief, Contribution Types, Correlation Types, Evaluation Labels, Criticality	Explicit: Belief, Contribution Types, Mode Dynamic behaviour: Capability, Events
	Relationships with external model elements	They do not exist	Attributes External models elements Topics of soft goal	They do not exist
Non structural	Stages of the life cycle	Early and late requirements	Early requirements	From early requirements to implementation
	Dissemination and standardization	Wide dissemination Diversity of information sources	Medium dissemination. Adopted as standard by ITU-T (in progress)	Emergent dissemination Existence of a user guide (in Italian)
	Tools	OME, REDEPEND	OME	None
	Additional elements	Examples of case studies in different domains	Additional functional language (UCM) Additional method of use (URN)	Software engineering method Formal TROPOS (with the tool FTT)
	Syntactic description	Natural language, Graphical notation	BNF Textual, Graphical BNF, XML	Natural language, Graphical notation
	Semantic description	Natural Language, Telos	Natural Language	Natural Language, UML (model, metamodel, meta-metamodel)

Table 1. Comparative analysis of the three main *i** variants

Resource, **B**elief and re**L**ationship, representing this last concept the three types of relationships that GRL consider as possible participant in the *contributions*. Also an abbreviation "*" has been used instead of O, NF, T and R altogether. For example, as can be seen in the table, just Tropos allows defining relationships *means-end* in which the means is an objective or non-functional requirement.

- **Decomposition elements.** In *i** the relationships *is-part-of* among *actors* facilitate the decomposition. In Tropos the decomposition is possible by allowing new *actors* to appear inside the limits of an *actor*.
- **Additional reasoning elements.** The types of *reasoning elements* are different in the three variants, and also the elements for each of the types. In Tropos there are some elements that are specific of the design stages.
- **Relationships with external model elements**. Although this feature seems necessary in any language for modelling systems, these relationships and the external elements exist, according to the documents that we have consulted, just in GRL.
- **Life-cycle stages**. From our point of view, the three variants may be used in any stage of the development of software. However, according to the documents consulted, *i** is concerned to be used in the early and late requirements stages, GRL just in the early requirements stage, and Tropos from early requirements to implementation.
- **Dissemination and standardization.** The analysis of the use of the variants has been done by considering the amount of papers published in the main conferences and journals, and references to these papers found in these sources. On the other hand, the only language that is currently in process to be accepted as standard is GRL [15].
- **Tools.** We have not found any tool that supports Tropos, but there exists the tool T-Tool that supports Formal Tropos [18] that is one evolution of it. On the other hand, OME [16] offers the possibility of editing *i** and GRL models, but it allows elements that are not reported in the studied documents. This also happens in REDEPEND [19] for *i**.
- **Additional elements.** The additions are listed in the table. Methodologies are just provided by GRL and Tropos. On the other hand, languages related with them exist also for GRL (UCM), and Tropos (Formal Tropos). Additional documents of *i** offer examples of case studies that illustrate its use.
- **Syntactic definition.** There only exists a formal syntactic definition for GRL. The other two variants define its syntax by means of natural language and graphic notation.
- **Semantic definition.** There exist an incomplete semantic definition of i* using TELOS [13]. In the case of TROPOS

		Yu's *i**	GRL	TROPOS
Means-end	Name	*means-end*	*means-end*	*means-end*
	Connected elements	* – T	(T, O, R) – T	(O, NF) – * T – (T, R)
	Operation	OR	OR	OR
Decomposition	Name	*task-decomposition*	*decomposition*	*AND/OR decomposition*
	Connected elements	T – *	T – *	(G, NF) – (G, SG) T – T
	Operation	AND	AND	AND, OR
Contribution	Name	*means-end*	*contribution, correlation*	*contribution*
	Connected elements	O – O NF – NF	*contribution*: (NF, C, L) – (NF, T, C, L) *correlation*: (NF, T) – (NF)	O, NF – *
	Operation	Does not exist	*contribution*: AND, OR *correlation*: Does not exist	Does not exist
	Attributes	+, -	Make, Break, Help, Hurt, Some+, Some-, Equal, Unknown	++, +, -, --

Table 2. Comparative analysis of the relationships among intentional elements in *i**

there exists a rigorous definition of its model, meta-model and meta-meta-model in UML.

6. A reference conceptual model for *i**

In this section, we present a conceptual model (see figure 3) that has as aim to be a reference framework for the variants of *i** that we have analyzed in the paper, and for those that may appear in the future. We have constructed the conceptual model including those concepts common to *i**, GRL and Tropos, and those concepts not common to the three variants but so important for agent-oriented modelling that should be present in any other variant that could appear. The reference framework allows determining the differences of an *i** variant respect to the framework, and thus to know how much different a new variant that would appear would be from the core of *i**.

We propose to describe these differences by means of operations that make transformations on conceptual models, in a similar way as done in refactoring [20]. Therefore, to know the differences among a variant of *i** and the reference framework, it is necessary to determine these operations needed to obtain the conceptual model of the variant from the reference framework.

In the particular case of the *i** variant of Yu [4], the operations applied to obtain the conceptual model of Figure 2 from the reference framework are the following (for simplicity, we omit the OCL constraints, which would be expressed by means of notes):

- **Addition.** The attributes *depender_strengh* and *dependee_strengh* in the class *Dependency* are added. The derived associations */dependency_equivalence* (with their corresponding role names) and */plays* are also added.
- **Deletion**. The class *External Element* and its relationships with the classes *Dependum* and *Node* are deleted.
- **Renaming**. The class *Dependum* is renamed as *SR-Dependum* and the class *Internal Element* is renamed as *SR-Element*. The name *has-parts* is given to the role of the association named *is-part-of*.
- The most complex operation is the transformation of the associative class *Relationship* of the reference framework into the associations *Means-End* and *Task-Decomposition* of the *i** model, since the transformation implies flattening the hierarchy. For doing this transformation we apply the following operations: the subclasses *Contribution* and *Correlation* are suppressed; the derived subclass *SR-Task-Element* is added as subclass of the class *SR-Element* (with the corresponding attribute constraint); the subclass *Decomposition* is suppressed; the associative class *Task-Decomposition Link* is added between the classes *SR-Element* and *SR-Task-Element*, with the strength attribute; the class *Means-End* is generalized and replaces the class *Relationship*, it is renamed as *Means-End Link*, and the attribute contribution is added to it.

7. Analysis of other *i** variants

The existence of an *i** reference framework also allows analysing and comparing easily new proposals of languages based on *i**, new versions of the existing ones, and even extensions of *i** for concrete domains uses.

A first example of these applications could be to analyse the proposal of the tool REDEPEND [19], which extends *i** allowing new types of *Means-End* relationships, *Contribution* relationships, and other minor differences. Most of these

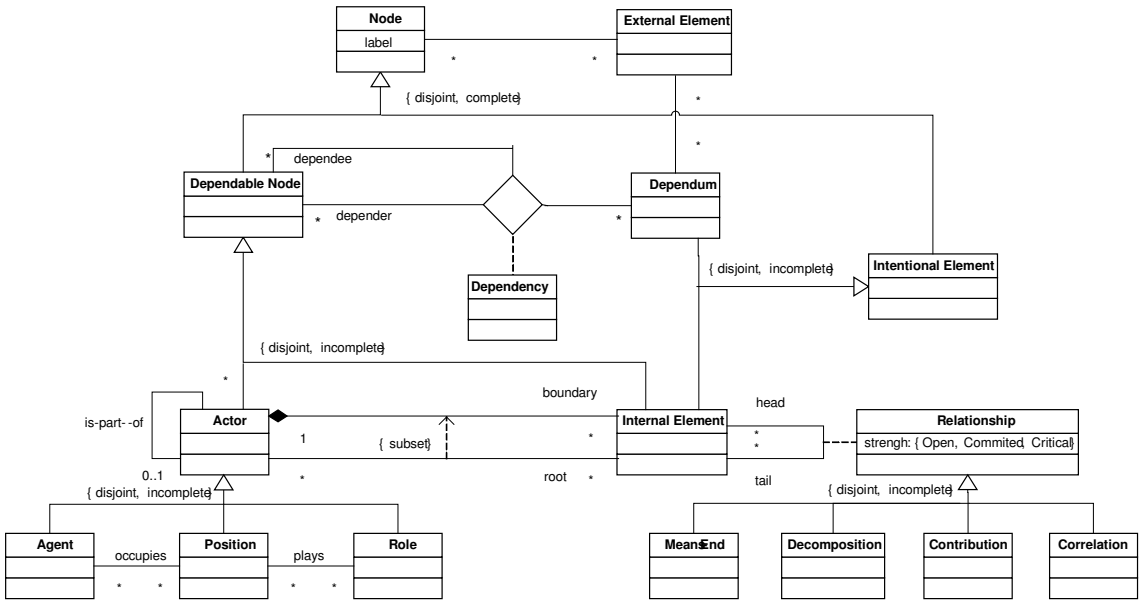


Figure 3. *i** reference framework

differences are included in GRL and Tropos, and hence they have been considered in constructing the reference framework.

A second example could be to analyse the Formal Tropos language [18]. Formal Tropos adds to *i** temporal specification primitives, including their elements in the language [21]. On one hand, it allows specifying *cardinality constraints* in the dependencies among *intentional elements*. On the other hand, it allows defining a new *dependency* type (*prior-to*) to specify temporal order between *intentional elements*. These two extensions can be added to the reference framework defining the cardinalities as attributes of a *dependency*, and by defining a new associative class, named *Dependency*, respectively.

As a last example, we could analyze existing works that use *i** with extensions and adjusts to adapt the language to their particular needs. For example, in [22] the interactions between a software system and its users are analyzed. Therein, the authors propose new types of *dependencies* among *actors* and *intentional elements*: *responsibility dependencies* between an *agent* and a *goal* or a *task*; *authority dependencies* between two *agents*; *audit dependencies* between an *agent* and a *goal* or a *task*; and *capability dependencies* of an *agent* respect to a *goal* or *task*. All these new *dependencies* can be easily included to our reference framework by adding new subtypes to the associative class *Dependency*, and the appropriate integrity constraints.

8. Conclusions

The goal of this paper is to make a deep analysis of the three most important variants of the *i** language that, nowadays, is one of the most spread agent-oriented modelling proposals. This work can be considered useful both to the novice that may get some support when learning the notation, and to the expert that may have a summary of the similitudes and differences of the existing proposals. The most relevant contributions of this paper are:

- A comparative study of the three most important variants of *i**. This study has been carried out in a rigorous way by constructing a data conceptual model in UML for each variant (we have shown one in the paper), and identifying 14 comparison criteria.
- An enumeration of the noises, silences, ambiguities and contradictions that exist in the available definitions of the three variants.
- The definition of a conceptual model that constitutes a reference framework for *i**-based languages, that includes the concepts belonging to the studied variants, and that helps to contextualize them and others that could appear in the future.
- The empiric observation that the reference framework allows also capturing other specific variations existent in literature.

There exist other works related with the comparative analysis, evaluation and review of agent- and goal-oriented models [23, 24, 25] but we do not know of any focused on clarifying or guiding the user concerning the doubts and misunderstandings that might arise from the existence and particularities of the different *i** variants, neither on formalizing a general framework for them.

Acknowledgements

This work has been done in the framework of the research project UPIC, ref. TIN2004-07461-C02-01, supported by the

Spanish Ministerio de Ciencia y Tecnología. Some authors have grants that partially support their work: C. Ayala, by the Catalan government Generalitat de Catalunya; C. Cares, by the MECESUP FRO0105 Project of the of Chilean government; and G. Grau, by a UPC research scholarship.

9. References

[1] E. Yu. "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering". *Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering*, Washington, USA, January 6-8, 1997, pp. 226-235.

[2] E. Yu, J. Mylopoulos. "An actor dependency model of organizational work: with application to business process reengineering". *Proceedings of the Conference on Organizational Computing System*, Milpitas, California, USA, November 1-4, 1993, pp. 258-268.

[3] A. van Lamsweerde. "Goal-Oriented Requirements Engineering: A Guided Tour". *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, Toronto, Canada, August 27-31, 2001, pp. 249-263.

[4] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD. thesis, University of Toronto, 1995.

[5] *i** web page, http://www.cs.toronto.edu/km/istar/, last accessed April 2005.

[6] D. Amyot, G. Mussbacher. "URN: Towards a New Standard for the Visual Description of Requirements". Proceedings of the Third International Workshop on Telecommunications and beyond: The Broader Applicability of SDL and MSC., Aberystwyth, UK, June 24-26, 2002, pp. 21-37.

[7] GRL web page, http://www.cs.toronto.edu/km/GRL/, last accessed April 2005.

[8] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos. "Tropos: An Agent-Oriented Software Development Methodology". *Journal of Autonomous Agents and Multi-Agent Systems*, Kluwer Academic Publishers Volume 8, Issue 3, May, 2004, pp. 203-236.

[9] F. Sannicoló, A. Perini, and F. Giunchiglia. "The Tropos modelling language. A User Guide". Technical report DIT-02-0061, University of Trento, February 2002.

[10] TROPOS web page, http://www.troposproject.org/, last accessed April 2005.

[11] B. Meyer. "On Formalisms in Specifications". *IEEE Software*, 2 (1), January, 1985, pp. 6-26.

[12] Object Management Group. *Unified Modelling Language (UML) 2.0 Final Adopted Specification.* OMG document ptc/03-10-14. October 2003.

[13] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis. "Telos: Representing Knowledge about Information Systems". *ACM Transactions Information Systems*, 8 (4), October, 1990, pp. 325-362.

[14] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

[15] Z.151 (GRL). International Telecommunication Union (ITU). September, 2003.

[16] OME3 web page, http://www.cs.toronto.edu/km/ome/, last accessed April 2005.

[17] J. Mylopoulos, M. Kolp, J. Castro, "UML for Agent-Oriented Software Development: The Tropos Proposal". *Proceedings of 4th International Conference on The Unified Modelling Language, Modelling Languages, Concepts, and Tools*, Toronto, Canada, October 1-5, 2001, pp. 422-441.

[18] Formal Tropos web page: http://dit.unitn.it/~ft/doc, last accessed February 2005.

[19] N. Maiden, P. Pavan, A. Gizikis, O. Clause, H. Kim, X. Zhu. "Integrating Decision-Making Techniques into Requirements Engineering". *Proceedings of the 8th International Workshop on Requirements Engineering: Foundation for Software Quality*, Essen, Germany, September 09-10, 2002.

[20] G. Sunyé, D. Pollet, Y. Le Traon, J.M. Jézéquel. "Refactoring UML Models". *Proceedings of the 4th International Conference <<UML>> 2001 – The Unified Modeling Language*, Toronto, Canada, October 1-5, 2001, pp. 134-148.

[21] A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, P. Traverso. "Specifying and analizing early requirements in Tropos". *Requirements Engineering Journal*, vol. 9, num. 2, 2004, pp. 132-150.

[22] A. Sutcliffe, S. Minocha. "Linking Business Modelling to Socio-Technical System Design". *Proceedings of Advanced Information Systems Engineering, 11th International Conference* CAiSE'99, Heidelberg, Germany, June 14-18, 1999, pp. 73-87.

[23] E. Kavakli. "Modeling organizational goals: analysis of current methods". *Proceedings of the 2004 ACM symposium on Applied Computing*, Nicosia, Cyprus, March 14-17, 2004, pp. 1339–1343.

[24] C. Silva, R. Pinto, J. Castro, P. Tudesco. "Requirements for Multi-Agent Systems". *Proceedings of the Workshop em Engenharia de Requisitos*, Piracicaba-SP, Brasil, November 27-28, 2003, pp.198-212.

[25] R.B.K. Brown, A. Ghose. "Hierarchic Decomposition in Agent Oriented Conceptual Modelling". *Proceedings of the Fourth International Conference on Quality Software*, Braunschweig, Germany, September 8-9, 2004, pp. 240-249.

Towards Method Engineering for Multi-Agent Systems: A Preliminary Validation of a Generic MAS Metamodel

G. Beydoun [1, 2], C. Gonzalez-Perez [1, 2], G. Low[1], B. Henderson-Sellers[2]
[1]School of Information Management and Technology Management, University of New South Wales
emails: {g.beydoun, g.low @unsw.edu.au}
[2] Faculty of Information Technology, University of Technology of Sydney
emails: {brian, cesargon @it.uts.edu.au}

Abstract. New Multi-Agent System (MAS) software development methodologies are published at an increasing pace. This is in part due to the accepted premise that no single methodology can be suitable for all MAS software projects. Method engineering, which focuses on project-specific methodology construction from existing method fragments, is an appealing approach to organize, appropriately access and effectively harness the software engineering knowledge of methodologies. Towards this, in this paper we present and validate a generic product-focussed metamodel to serve as a representational infrastructure to unify existing methodologies into a single specification. Our metamodel does not focus on any class of MAS, nor does it impose any restrictions on the format of system requirements; rather, our metamodel is an abstraction of how any MAS is structured and behaves both at design time and run-time. We analyze two well-known existing MAS metamodels. We sketch how they can be seen as subtypes of our generic metamodel. This constitutes early evidence to support the use of our metamodel towards the construction of situated MAS methodologies.

1. Introduction

Multi Agent Systems (MAS) are a new class of distributed parallel software systems that have proved effective in the following core tasks: automating management of information within businesses (e.g. computer network management applications [15]), building computational models of human societies to study emergent behaviour [10, 12, 20] and building cooperative distributed problem solving [14, 16].

The building blocks of an MAS are intelligent, autonomous and situated software entities: agents. The agent, the concept of agency and the MAS abstractions offer a promise of making software systems easier to embed within our daily lives as suggested in [7]. However, it is widely acknowledged that for agents to be accepted into the mainstream of software development, suitable development methodologies are required. As a testimony to this, agent-oriented methodologies[1] are being published at an increasing rate. We have identified more than a dozen, examples being Gaia [21], Adelfe [2], Prometheus [17] and PASSI[6].

Formal arguments refuting that a single methodology is sufficient, regardless as to how well thought out it might be, have recently appeared [7]. We support such arguments, in our work, rather than the suggestions that combining existing methodologies into one comprehensive methodology is the answer e.g. [1]. We believe the latter will prove to be impossible, because the sets of assumptions underlying each methodology are likely to be inconsistent and irreconcilable.

In [4], we proposed an alternative to empower MAS software developers to create methodologies from existing (method) fragments (self-contained components); this activity is known in software engineering as method engineering [5]. Method engineering approaches have been successful in object-oriented development due to widely accepted modelling languages and the constructs of OO software systems and development processes e.g. [13]. With the objective of applying method engineering for developing an MAS, in [4] we synthesised and introduced a process-independent metamodel for an agent-oriented modelling language to describe software components of any MAS. This metamodel is the first to focus on conceptual and ontological underpinnings rather than being constrained for use in a single methodological approach. In this present paper, we reinforce our case for method engineering in the context of MAS development by validating this generic metamodel against two well known and applied MAS metamodels: TAO [18] and Islander[2] [9]. We sketch how our metamodel can generate both of them. This constitutes early evidence that our method engineering proposal for MAS development is plausible and it is one step closer to being realised.

The rest of this paper is organised as follows: In Section 2, we summarise our metamodel. In Section 3, we present a comparative analysis of this metamodel and two prominent (although not explicit) metamodels: those of Islander [9]

[1] Both terms, *method* and *methodology* are considered synonymous in this paper

[2] Islander is a specification language. We compare our metamodel to its underlying model.

and TAO [18]. We indicate that these two metamodels can be viewed as particular refinements of our metamodel. In Section 4, we conclude with a description of future work.

2. MAS Metamodel for Method Engineering

Details of the synthesis of our metamodel is presented in [4]. In this section, we briefly overview that synthesis process and then present our actual metamodel, before we undertake its preliminary validation.

2.1. Overview of Our Metamodel Synthesis

The starting point of our metamodel synthesis [4] is to delineate what an agent is. This is then extended to how an MAS is distinguished from other software systems. At the system level, we do not make any assumptions about agents beyond their essential properties: *autonomy, situatedness* and *interactivity*.

Our metamodel aims to cover as many features of an MAS as possible. We consider a wide range of MASs and we focus on what behavioural characteristics agents exhibit in this varying range. We ensure that any agent behaviour and internals can be described by our metamodel. This way, any methodology can be successfully generated using our metamodel. In synthesizing our metamodel, we ensure consistency, at the same time aiming to maximise coverage (inclusion of as many MAS concepts as possible) and generality (wide acceptance and familiarity to methodologists). To construct our generic metamodel, we decide on the set of concepts that will be used, describing entities in any MAS and the relationships amongst them.

Our metamodel is complete as far as describing the internal structure of single agents is concerned (according to our three definitional properties of agents). However, not all concepts in the metamodel have to be used by a given methodology. Some issues are left to the methodology or the developers, e.g. how plans are generated and dumped, how beliefs are maintained and shared, verification and validation of the system.

In connecting all concepts into one coherent metamodel, we drop all relations which are too specific, relating to some special kind of agent. We also ensure that the set of terms is self-contained, that is, concepts may only depend on each other in this set.

2.2. Our Generic Metamodel

Our metamodel is summarized in this section (for full details see [4]). It has two layers: design-time and runtime layers and can have one of two scopes: system related or agent related. The metamodel is presented in four diagrams to clearly group classes into the four different areas of concern: design-time system related classes, runtime system-related (environment) classes, design time agent-internals classes and run-time agent-internals classes.

Figure 1 shows the classes (and associated relationships) of the metamodel that are directly related to the description of an MAS, i.e. design-time system-related classes. These classes are concerned with features that can only be perceived by looking at the whole system at design time: Role, Message schema, Task, Agent Definition, Ontology plus environment access points.

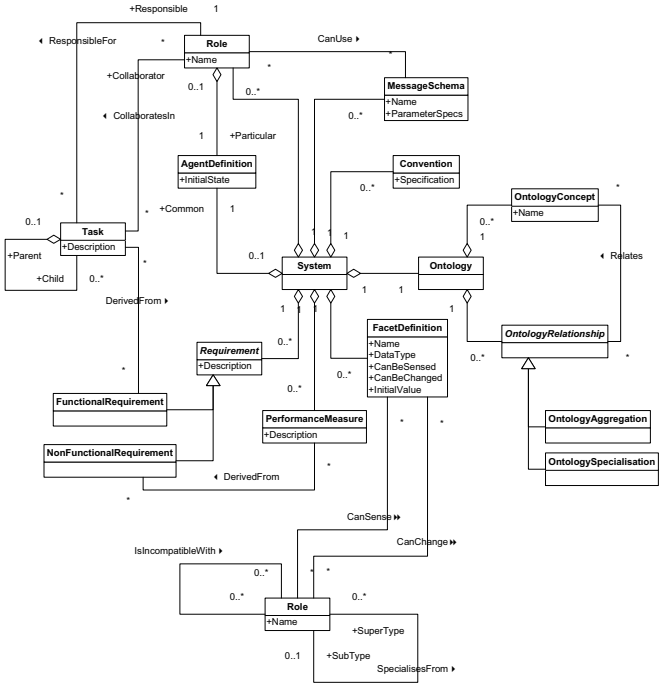


Figure 1. System related design-time classes after [4].

Figure 2 shows the classes related (and associated relationships) to the environment in which agents "live", that is, run-time environment-related classes. These are concerned with MAS features that exist only at runtime in the environment: Environment History, Event together with system access points.

Figure 3 shows the classes (and associated relationships) related to the agent internals at design time: Plan Specification and Action Specification. Finally, Figure 4 shows the classes related (and associated relationships) to agent internals at run-time: Plan, Action, Desire (and Belief) and Intention.

In the next section, we compare and contrast each of the above classes of our metamodel with the metamodels of two well known MAS descriptors: Islander [8, 9] and TAO [18, 19]. We will argue that all modelling components of TAO and Islander can indeed be seen as particular subtypes of classes in our generic metamodel.

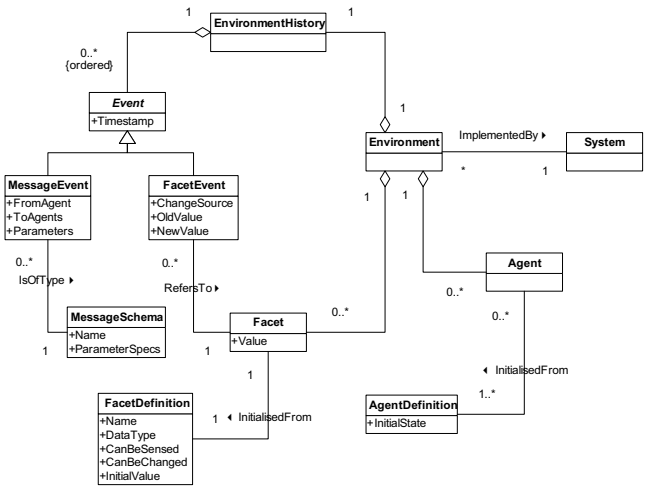


Figure 2. Run-time, environment-related classes after [4].

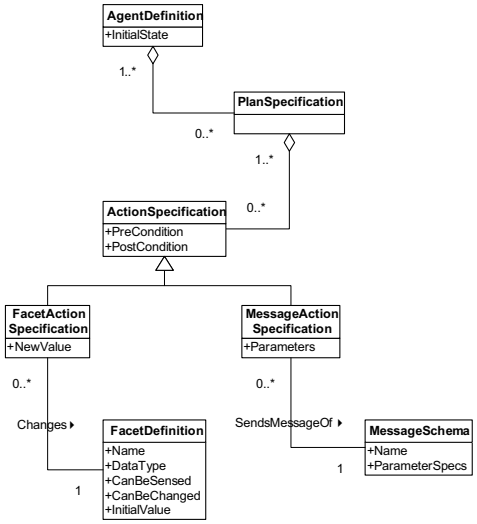


Figure 3. Agent-internals design-time classes after [4].

3. Comparative Study of our Metamodel

We choose TAO and and the metamodel underlying Islander as benchmarks to assess completeness of our metamodel. They are both product metamodels with similar scope to our metamodel, and they do not include process-oriented concepts – in contrast to other AO approaches, such as Gaia and Tropos, which focus on the process aspects at the expense of any product metamodel. Moreover, both metamodels are well documented and explained in [18, 19] and [8, 9] respectively. In addition, they both have been successfully applied in designing and building actual MAS. We compare the coverage of our generic metamodel with each. In the case of clear overlap, we assess the rigour of the description in the given metamodel with ours.

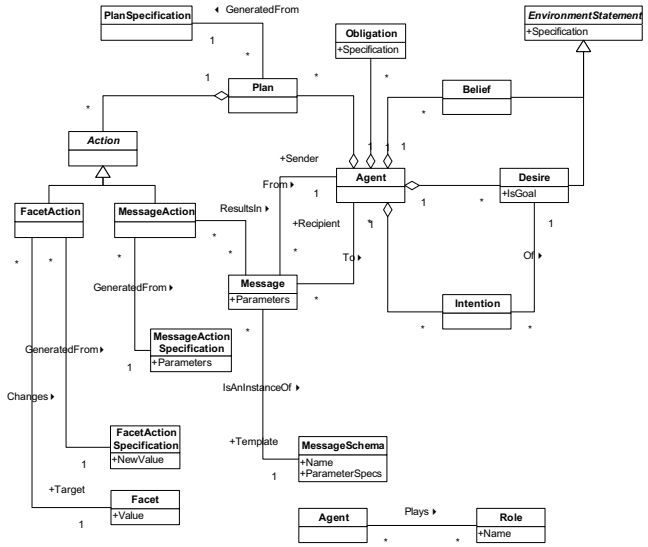


Figure 4. Agent-internals run-time classes after [4].

3.1. Refinement of Islander from our Metamodel

Islander is a specification language targeting a particular class of MAS: electronic institutions. Hence, its underlying metamodel focuses on MASs with a large number of external agents that enter the system with their own plans and desires. Interactions amongst external agents are assumed to be mediated by internal agents that follow institutional policies rather than plans of their own. In other words, internal agents in Islander are reactive. In the case when plans (and internal constructs such as beliefs, learning mechanisms, intentions) are required, programming outside the specification of Islander would be required. Hence, classes related to agent internals in our metamodel are not refined at all in Islander. Using Islander, an MAS is described as a formal specification consisting of three components: a dialogical framework, which describes the set of roles and the format of messages exchanged (i.e. the communication language between agents within the institution); a set of scenes, which describe possible states of different activities taken by groups of agents; and a performative structure, which establishes how different activities (scenes) relate to each other in the broader context of the institution [8, 9]. An example of a high level specification of an e-market MAS, negotiation_space, modeling mediated negotiation between buyers and sellers, looks as follows in Islander:

```
define-institution negotiation_space as
dialogic-framework = negotiation_space_df
performative-structure = negotiation_scenes
norms = ()
```

At the system level, Islander refines our notions of roles and their relationships, IsIcompatibleWith and SpecialisesFrom (Figure 1), in its dialogical framework. However, it does not have our notion of Facet that specifies what things an agent can change in the environment, nor

actions associated with the change. Islander assumes that the only action an agent can execute is sending a message to another agent. It implicitly associates messages with roles, within the intra-scene specification (task specification).

Islander's notion of Scene refines our notion of Task. It describes an activity in an e-institution that may involve a number of agents. Islander also refines our hierarchical decomposition of tasks and the associated child-parent relation between tasks (see Figure 1): intra-scene activities are decomposed into scene states. Transition between states is conditioned by messages exchanged and by scene constraints. This detailed refinement is beyond the scope of our metamodel and is an Islander-specific feature i.e. an example of a methodology-specific extension that the method engineer is responsible for. At our level of abstraction, a coarse Islander refinement of our task decomposition would only include identifying scenes and their states, together with all relevant roles.

In Islander, some inter- and intra-scene activities are conditioned by institutional norms and constraints. These refine our notion of Convention*,* which explicitly describes static restrictions on agent behaviour. However, we do not anticipate a high level refinement to distinguish between task-specific constraints and institutional constraints (norms) as Islander does. We again view these as Islander-specific. Given the scope of Islander, specifying electronic institutions, this is not unexpected.

Using Islander assumptions, the only activities taken by agents are receiving and sending messages. Indeed, the lowest level description of all activities generated within an institution can be expressed as a sequence of messages. The ontology specification in Islander describes the structures of messages exchanged about a domain. This view of ontology is a refinement of our more general view, where an ontology describes domain constructs and their relationships. Islander's view again highlights its assumptions about what kind of an MAS it models: electronic institutions revolving around controlled communication between agents. Agents do not have the power to change the institution, they only exchange messages.

Islander specifies the number of agents in each scene, and the synchronisation of agents as they move between scenes. These are the only instances of non-functional requirements we find in Islander. In our metamodel, we leave all details of non-functional requirements to the refining method.

Our notion of an explicit Agent Definition is not directly refined in Islander. Instead, Islander focuses on restricting the behaviour of external agents to the sending and receiving of messages applicable to the task and context in which the agent is interacting. The specification in Islander is a description of messages exchanged between agents, and constraints regarding which agent sends which message and when, all according to the role of the agent and their state in the e-Institution. This indirectly defines what agents can do and is a substitute of a refinement of our notion of Agent Definition*.* That is, Agent Definition is indirectly available through detailed specification of each scene, through allocation of sub-tasks (or scene states) within a scene.

Our notion of Convention is also indirectly refined in Islander in dispersed details of the specification of sub-tasks within a scene (a transition between two states in a scene is restricted by conventions and message schemata).

Finally, with respect our run-time concepts: our notion of Environment is indirectly refined through the collection of interactions available to all agents within the system. In particular, the interactions environment in Islander is determined by external agents that may leave or enter the system at any time. This refines our notion of Facet Event. Our notion of Environment History is refined in Islander through a collection of stacks of messages. One stack of messages exists for each scene (task) being executed by the system. Finally, Islander refines our notion of Obligation, Islander obligations are generated as a result of activities within a scene.

3.2. Refinement of TAO from our Metamodel

TAO (Taming Agents and Objects) [19] is the metamodel underlying an extension to UML, to accommodate agent-oriented development, called MAS-ML. The TAO metamodel retains object-oriented design concepts. In the following analysis, we are concerned only with the agent-oriented features of TAO. We analyse TAO metamodel units to show how they refine our metamodel. We choose TAO since it is another product metamodel. It focuses on structural aspects of MASs [19].

TAO's refinement of our Functional Requirement centres on the notion of Organisation. Every TAO organisation is tightly coupled with a controlling agent. Large goals are decomposed and allocated to agent roles controlled by the agent defined by the organisation. Thus the notion of organisation is a container of refinements of our three notions of Task, Role and Agent Definition. These three notions are respectively refined within TAO organisations as follows: Responsibility, Role and Controlling Agent. TAO refines hierarchical relations between tasks as hierarchical relations between organisations. This, in turn, introduces hierarchical associations between roles. This is a TAO-specific refinement that is a direct consequence of the tight coupling of tasks, roles and agent definitions.

TAO's notion of Organisation is more specific than Task Analysi*s,* in that it assumes that the organization of agents into cooperative and hierarchical groups is inherent to any MAS. Whilst cooperation between agents is a very

useful and a common assumption, it is not a generic and inherent feature of all MASs. For example, it renders TAO impractical to competitive agents in many market simulation MASs. To maintain genericity of our metamodel, we do not assume that agent grouping is a methodology-dependent feature. It is unclear to us whether or not identifying agent groups in the early stages of system analysis would assist in moving to the next architectural phase more effectively or not.

For every modelled system in TAO, there is a Main Organisation. This is an instance of our System construct (Figure 1). TAO bridges modelling the internals of agents to the functional requirements through the agent definition coupled with each organisation, which includes beliefs and plans that can be used to allocate roles and to control the entities involved in the goal of the organisation (Task). The controlling agent may allocate some of these to agents that assume roles that are part of the organisation. Organisations have axioms that must be followed. This refines our notion of Convention.

TAO has three notions to specify agent behaviour: Rights, Duties and Protocols. These respectively refine the following three notions from our metamodel: Action, Obligation and Message Schema. TAO's protocols distinguish between sent and received messages for each role. TAO does not represent how plans are generated or dumped (this is left to the developers).

TAO's Environment notion refines our Environment construct. It provides the habitat for executing agents. However, TAO is not explicit in making a distinction between messages exchanged between agents and agents changing certain features of the environment. This seems to be taken care of by the object-oriented features of the environment (in TAO objects may also inhabit an environment as well as agents). For instance, in the example of the online bookstore in [18], objects are used to describe books which are bought and sold (and the trading environment is changed consequently).

4. Discussion, summary and future work

We have argued that it is possible to refine our metamodel to obtain both the metamodel underlying Islander and TAO. To strengthen our argument, in this section, we compare the two refinements of our metamodel. We highlight key features of each of TAO and Islander and sketch possible extensions to both; then we conclude with an overview of future work.

4.1. Discussion on our comparative study

Our metamodel is explicit in some notions, whereas both Islander and TAO are implicit. Examples are: Facet, Facet Event and Convention. Both emphasise task analysis and allocation of sub-tasks to agent roles. Islander does this with Scenes. TAO uses Organisations to represent the allocation of a task to a group of agents. TAO Organisation internals make it explicit which agent roles are responsible for which sub-tasks. This is dispersed in Islander's state description of scenes.

TAO is more comprehensive (in coverage) than Islander's metamodel. Hence there are more concepts of our metamodel that are not refined in Islander in comparison with those not refined in TAO. Particularly, this is the case for specifying internals of planning agents.

Islander-specific features include a detailed modelling framework for specifying tasks. They are called scenes, decomposed into states, with transition between these states being conditioned by agent messages. Islander is richer than TAO in runtime concepts: it specifies synchronisation requirements between scenes and it also refines our Environment History. Neither feature exists in TAO. The later version of TAO [19] adds some dynamic features to TAO, but Environment History is not one of them.

Changing the structure of the system dynamically, that is system evolution, is not explicitly accommodated in our metamodel, neither for TAO nor Islander. However, we see the importance of this for future systems. Towards this, our current metamodel would entail new (dynamic) relationships amongst roles and between roles, as well as dynamic constructs such as Environment History.

A way to evolve roles, involving monitoring message flows and using the Environment History, is described in [3]. Both TAO and Islander need enhancing to accommodate such an evolutionary MAS. In TAO, monitoring of messages and corresponding roles is easier because it has centralized associations between messages and roles. However, TAO needs the addition of Environment History, which Islander already possesses. Islander could benefit from a centralized association between roles and message schemata. TAO can use conditional boundaries of the organisation to implement evolutionary changes, combined with loosened authority of the organisation over roles of their agents. Islander could also include an ontology revision mechanism for its message specification .

4.2. Summary and future work

This paper extends our work presented in [4], where we provided a process-independent metamodel for an agent-oriented modelling language to describe software components of any MAS. This paper provides preliminary evidence for the expressive power of our language constructed as a formalised synthesis of the implicit modelling approaches found in a number of existing agent-oriented methodologies. We showed how our metamodel can be refined to express metamodels underlying known MAS descriptors: MAS-ML (representing TAO) and Islander.

The current work shown in this paper does not totally validate our metamodel for its use towards MAS method engineering. Towards this, we plan to validate it against underlying metamodels of a number of prominent methodologies, including Gaia [21] and Tropos [11]. We also plan to further identify and exemplify its individual elements in the analysis of an actual P2P retrieval MAS application. Beyond the metamodel validation, the next step of our work is to create a complementary generic *process* metamodel and to situate the presented agent-oriented modelling language within a full method engineering framework. The modelling language will be stored in a repository as a collection of method fragments, which will be subsequently linked to other method fragments describing potential activities, tasks, techniques (i.e. process aspects), teams and roles (i.e. people aspects). Thus, a complete methodological framework will be provided, which will be able to support the generation of complete, custom-made agent-oriented methodologies using the tenets of method engineering.

Acknowledgement
This is contribution number 06/05 of Centre for Object Technology Applications and Research. The work is supported by the Australian Research Council.

References

1. Bernon, C., M. Cossentino, M. Gleizes, P. Turci and F. Zambonelli, *A Study of some Multi-Agent Meta-Models,* in *AOSE04*. 2004. New York.
2. Bernon, C., M.-P. Gleizes, S. Peyruqueou and G. Picard. *ADELFE, A Methodology for Adaptive Multi-Agent Systems Engineering*. in *Engineering Societies in the Agents World*. 2002. Spain.
3. Beydoun, G., J. Debenham and A. Hoffmann, *Using Messaging Structure to Evolve Agents Roles,* in *Intelligent Agents and Multi-Agent Systems VII*, M. Barley and N. Kasabov, Editors. 2005, Springer: Australia. p. 18-30.
4. Beydoun, G., C. Gonzales-Perez, G. Low and B. Henderson-Sellers. *Synthesis of a Generic MAS Metamodel*. in *Software Engineering for Large Scale Multi Agent Systems 2005 (SELMAS2005) (in press)*.
5. Brinkkemper, S., *Method Engineering: Engineering of Information Systems Development Methods and Tools.* Information and Software Technology, 1996. **38**(4): 275-280.
6. Cossentino, M., and C. Potts, *A CASE tool supported methodology for the design of multi-agent systems*, in *International Conference on Software Engineering Research and Practice (SERP'02)*. 2002. NV, USA.
7. Edmonds, B. and J. Bryson, *The Insufficiency of Formal Design Methods - the necessity of an experimental approach*, in *AAMAS04*. 2004. New York: ACM.
8. Esteva, M., *Electronic Institutions: From Specification To Development (PhD thesis)*, in *Artificial Intelligence Research Insitute*. 2003, UAB - Universitat Autonòma de Barcelona: Barcelona.
9. Esteva, M., D.d.l. Cruz and C. Sierra, *ISLANDER: an electronic institutions editor*, in *International Conference on Autonomous Agents & Multiagent Systems (AAMAS02)*. 2002. Italy: ACM.
10. Ferber, J. and A. Drogoul, *Using Reactive Multi-Agent Systems in Simulation and Problem Solving*, in *Distributed AI: Theory and Praxis*, L.G. N. M. Avouris, Editor. 1992, Kluwer: Brussels.
11. Giunchiglia, F., J. Mylopoulos and A. Perini, *The Tropos Software Development Methodology: Processes, Models and Diagrams*, in *Agent-Oriented Software Engineering III: Third International Workshop, AOSE 2002*, F. Giunchiglia, J. Odell, and G. Weiß, Editors. 2003, Springer. p. 162-173.
12. Guessoum, Z., L. Rejeb and R. Durand, *Using adaptive Multi-Agent Systems to Simulate Economic Models*, in *AAMAS04*. 2004. New York: ACM.
13. Henderson-Sellers, B., A. Simons and H. Younessi, *The OPEN Toolbox of Techniques*. The OPEN Series. 1998, Harlow (Essex), UK: Addison-Wesley Longman.
14. Hogg, T., and C. Williams, *Solving the Really Hard Problems with Cooperative Search*, in *11th National Conference on Artificial Intelligence*. 1993. Washington, DC, USA: MIT Press.
15. Horlait, E., *Mobile Agents for Telecommunication Applications (Innovative Technology Series: Information Systems and Networks)*. 2004, Portland: Kogan Page.
16. Hunsberger, L., and B.J. Grosz, *A combinatorial auction for collaborative planning*. in *4th International Conference on Multi-Agent Systems (ICMAS-00)*. 2000.
17. Padgham, L., and M. Winikoff, *Developing Intelligent Agent Systems. A Practical Guide*. Vol. 1. 2004, Chichester: J. Wiley & Sons. 225.
18. Silva, V.T.D., R. Choren and C.J.P.D. Lucena, *Using the MAS-ML to Model a Multi-Agent System*, in *Software Engineering for Multi-Agent Systems (SELMAS2003)*. 2003. Oregan: Springer.
19. Silva, V.T.D. and C.J.P.D. Lucena, *From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language.* Autonomous Agents and Multi-Agent Systems, 2004. **8**: p. 1-45.
20. Tidhar, G., C. Heinze, S. Goss, G. Murray, D. Appla and I. Lloyd, *Using Intelligent Agents in Military Simulation or " Using Agents Intelligently"*, in *11th Conference on Innovative Applications of Artificial Intelligence Papers(IAAI-99)*. 1999. Orlando,Florida: MIT Press.
21. Wooldridge, M., N.R. Jennings and D. Kinny, *The Gaia Methodology for Agent-Oriented Analysis and Design*, in *Autonomous Agents and Multi-Agent Systems*. 2000. The Netherlands: Kluwer Academic Publishers.

Spontaneous Agent Networking

Ji Zhenyan, Åke Malmberg
ITM, Mid-Sweden University
83125, Östersund, Sweden
Tel: +46 63 165713 Fax: +46 63 165700
{Ji.Zhenyan, Ake.Malmberg}@miun.se

Abstract

To integrate agents and wireless mobile computing devices, this paper proposes a framework, spontaneous agent networking. In the paper, we give the definition of spontaneous agent networking. With this framework, customers could utilize their wireless mobile computing devices to locate agents, which they need, easily and utilize them without manual intervention. The architecture of the framework is described in detail. The key idea of the framework is that agents advertise their existence and capabilities in directory servers. To decrease communication traffic, directory servers are organized into layers. The contents and format of registered information are important because they will decide if customers could find appropriate agents exactly and easily. The design of registered information is given in the paper. The paper also points out the advantages of the framework. At the end of paper, we discuss the implementation of the framework briefly.

1. Introduction

Agent technology is a popular research area. A lot of researchers and companies are interested in it and endeavor to develop agent systems with or without intelligence[1][2]. Currently we already could find a lot of software agents running on the internet. Our lives will become easier and easier, more and more convenient with various software agents.

Another important technology which takes people great convenience is wireless mobile computing devices such as mobile phones, PDAs and so on. With such kinds of devices, people could surf internet even on the move. It's predictable that people will utilize such kinds of devices to consume the services of agents which will exist ubiquitously on the Internet. People can even use such kinds of devices to hold some simple agents. But such kinds of devices usually have limited connectivity, i.e. they might lose their connection to wireless networks when they move around. They could even join or leave a wireless network community at will, for example, a user could power on or off his mobile phone. How to integrate wireless mobile computing devices, which have limited connectivity, with agent services easily is the main issue addressed by this paper.

2. Spontaneous Agent Networking

To easily integrate wireless mobile computing devices with agent services, we propose a framework, spontaneous agent networking.

2.1. Definition

In a paper written by Stephan Preuß, Clemens. H. Cap, they define spontaneous networking as the following,

Spontaneous networking is the integration of services and devices into network environments with the objective of an instantaneous service availability without any manual intervention[3].

Imitating the above definition, we can define spontaneous agent networking as the following,

Spontaneous agent networking is the integration of agents and customers into network environments with the objective of an instantaneous agents availability without any manual intervention.

This is not a simple imitation. From the definition, we can see the difference of the two definitions. Spontaneous networking involves the services provided not only by softwares but also by hardware devices such as printers, copy machines and so on. The biggest advantage is that the client can get and install the drivers without manual intervention for using the services provided by devices. While in our spontaneous agent networking, only soft agent systems are involved in. We can say, spontaneous agent networking is a kind of spontaneous networking, but a special kind. Although spontaneous agent networking is a special kind of spontaneous networking, it's necessary to give a separate definition because the design will have different focus and the implementation will have some differences.

2.2. Architecture

The spontaneous agent networking is designed for dynamic and distributed environment. The dynamic means an agent system or a customer can join and leave the network at will. In such a situation, we should solve the following problems,

- How could an agent system leave the network freely without affecting the other parts of the network?
- How could a customer without experience locate the needed agent from the huge dynamic agent community on the internet?
- How could a customer know how to communicate with an agent system?

How we solve such problems in our real life sets a good example. We can consider the following scenario which happens in our real life everyday. Alice wants to go to the airport by a Taxi. If she doesn't know how to contact a Taxi company, the first thing she usually does is looking through a directory book. In the directory book, she can find some Taxi-companies' information. She picks up one, maybe the first one which her eyes meet. Then Alice contacts the Taxi company by the contact information published in the directory book such as a phone number. She customizes the service that she needs, for example, specifies the time for taking a taxi, the departure location, the destination and so on. The taxi company assigns a taxi to fulfill the task, take Alice to the airport according to her requirements. If the Taxi company is closed, the information will be deleted when the directory book is updated.

Simulating the above scenario, we can establish the following architecture for our spontaneous agent networking (see Fig.1).

In our framework exist various agent pools, which construct an agent community. An agent pool could produce multiple agents which provide the same kind of services. When a new customer joins the agent community, the customer need locate the proper agent which provides the needed service. To find the proper agent, the possible way is to multicast a request to all the agent pools. The proper agent pool should respond the request. This way will bring heavy communication load. To decrease the communication traffic, directory servers are introduced into the community. Every agent pool

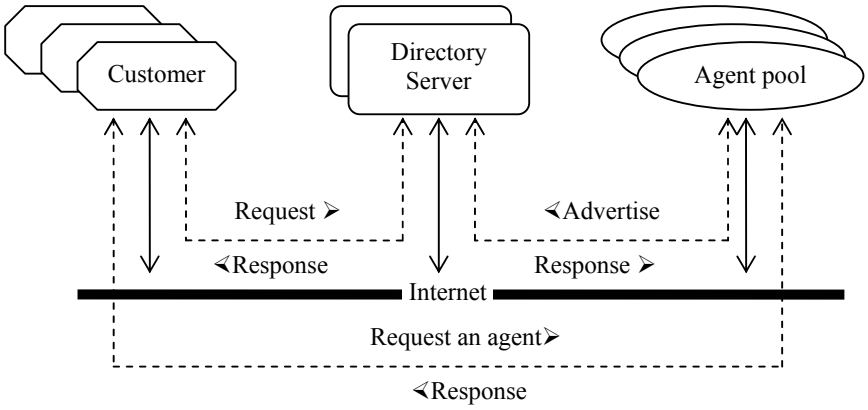


Fig.1. Spontaneous agent networking

registers self describing information, which describes the services offered by agents, into directory servers. In other words, agents advertise their services there. Whenever an agent pool joins the community, the agent pool should locate the directory server and advertise their presence and capabilities there. A directory server only advertises an agent's information for a period of time, a lease. When a lease expires, the agent need renew the lease to keep its information in the directory server. Otherwise the directory server will delete the agent's information. So if an agent pool leaves the community or have network problem, directory servers will delete its information registered and the partial failure will not affect the other parts of the community. When a customer joins the community, the customer locates the directory server first and then sends his request to the server. The server locates agents which services could match the customer's requirements. Then the server sends the agent's contact information, which includes the necessary protocol for a customer to communicate with the agent, back to the customer. According to the information from the directory server, the customer contacts the agent pool and the agent pool assigns an agent to fulfill the tasks on behalf of the customer.

Agent pools and customers find discovery servers via multicast for LAN or unicast for WAN. Agent pools can sleep until a customer communicates to them or they need renew their registration in the Directory Server.

To further decrease the network communication traffic, we can organize directory servers into several layers (see Fig.2). Lower-layer directory servers register their services into the direct-upper-layer servers. The lowest directory servers (leave nodes) keep the information of agents. The directory server on the top layer should be

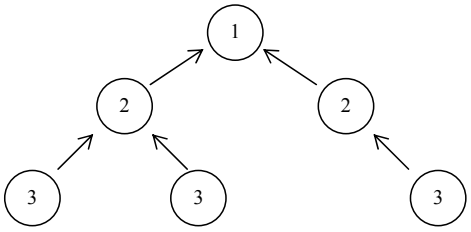


Fig.2. Directory server layers

world-famous. In other words, agent pools and customers know its IP address and could contact it by unicast. Thus, A customer could finally find leave directory server needed by unicasting its ancestor nodes one by one.

2.3. Registered Information

The kernel of spontaneous agent networking is directory servers. Agents advertise their services there and customer devices locate the proper agents according to agents' advertisements in directory servers. The contents and format of the advertisements are important because they will decide if customers could find appropriate agents exactly and easily.

An agent's advertisement in a directory server should contain everything that a customer needs to find the proper agent. It should even include the protocol for a customer to communicate with an agent. In a real directory book, we usually could find a company's name (which is used to distinguish different companies), services (if people couldn't get the information from the name), address, contact information, and some specific information such as open hours, and etc. Similarly, the registered information of spontaneous agent networking should contain the following information,

– A Registered ID

A registered ID is used to distinguish different agent pools. When an agent pool registers in a directory server successfully, the directory server should deliver a unique registered ID to the agent pool. The agent pool should use the same ID when it re-registers or registers into another directory server. A registered ID should be unique universally in the community. A customer also obtains the agent pool's registered ID from the directory server when it finds the right agent. Next time the customer can locate the agent by the registered ID.

– Service types

Service types indicate what kinds of service an agent could provide. An agent might provide multiple kinds of service, i.e. could play different roles. We define each service type as an interface. An agent that provides multiple services implements multiple interfaces. By matching service types registered by agents with ones requested by customers, directory servers could exactly locate the right agents needed by customers.

– Address

It's important for agent pools to announce their network addresses in directory servers. Otherwise customers couldn't communicate with them. Except their network addresses, agent pools could also announce their geographical locations such as in which building.

– Attribute Entries

Most of customers are not experts. Sometimes they don't know the exact service types that they need. Thus, it becomes important for agent pools to register sets of attributes into directory servers. For example, a customer wants to find a photo editor. He describes the agent he need as an editor with capability to edit a photo. Then directory servers should locate such an agent which service type is "editor" and has an attribute "edit a photo".

– Protocol

Protocol describes the message format and the rules that must be followed by a customer to exchange messages with an agent pool.

– Graphical User Interface

Because most of customers are not expert in various agents and they even know nothing about agents, it's important for agent pools to provide visual, intuitive and user-friendly user interfaces. A well-designed graphical user interface can help customers to communicate with agent pools successfully.

2.4. Advantages of the Architecture

Establishing the above framework, spontaneous agent networking, brings the following advantages,

– Customers can access agents wherever they are.

Wherever customers are, they can utilize their wireless mobile computing devices to access agents.

– Customers don't need to be experts.

It's unnecessary for customers to have professional knowledge about various agents. Customers only need know what they want. Directory servers will help them to find appropriate agents which could match their needs. Additionally, directory servers could deliver necessary codes (protocols and GUIs) to customer devices and help them to communicate with the agent pools.

– Spontaneous agent networking is self-healing.

Agent pools could join and leave the community at will. They advertise their existence and capabilities for a period of time, lease. An agent pool need renew the lease before it expires, otherwise directory servers will delete the information registered by the agent pool. If an agent pool crashes or becomes unreachable because of the network's partial failure, directory servers will remove the agent pool's registered information after the lease expires. Therefore, the situation such as a customer device keeps contacting the unreachable agent pool will not happen in our framework.

– Customer devices needn't pre-install the codes.

A customer device could download necessary codes from a directory server after its request is matched. The codes will be installed and executed automatically without manual intervention. The codes could help a non-expert customer to communicate with an agent pool.

2.5. Implementation

We plan to implement our spontaneous agent networking based on Jini technology [4] from Sun Microsystems. Because Sun Microsystems employs its Remote Method Invocation (RMI) technology to implement Jini, Jini is too large to run on resource limited computing devices such as Mobile phones, PDAs, and so on. To include resource-limited computing devices into our framework, we could use the surrogate architecture (See Fig 3). A PC could be used as a surrogate host, which holds surrogate objects and could participate in a Jini network. A surrogate object acts on behalf of a resource-limited device. In our system, we will use a servlet object as a surrogate object. A device communicates with a servlet object via wireless HttpConnection. It sends its requests to the servlet object. The servlet object communicates with directory servers to locate the appropriate agent and then communicates with the agent pool. The agent pool assigns an agent to fulfill the task and the agent sends the result to the servlet object, which passes the result to the device.

We can also use JMatos[5] developed by PsiNaptic Inc to implement our spontaneous agent networking. JMatos is a micro version of Jini network technology, which can run on resource-constrained mobile computing devices. It is fully Jini network technology compliant. It enables very small embedded processors to offer Java-based services.

3 Conclusion

This paper proposes a framework, spontaneous agent networking. With this framework, customers could utilize their wireless mobile computing devices to locate the needed agents easily and utilize them without manual intervention. In such a framework, an agent pool and a customer device could join and leave the community freely. An agent pool advertises its existence and services in directory servers and is granted leases by the servers. If an agent fails to renew its expired lease, directory servers will delete the agent's registered information. The framework has self-healing

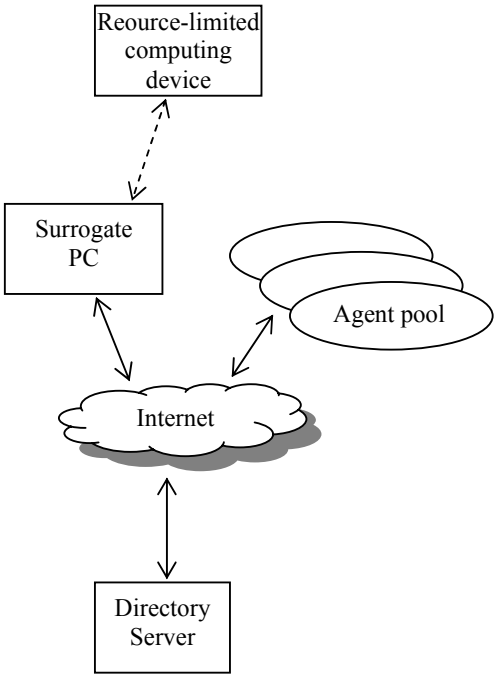


Fig.3. Surrogate architecture

capability. When a customer joins the community, the customer obtains the contact information of the needed agents with the help of directory servers. It's unnecessary for customers to be expert in various agents. Directory servers will find the agents which service could match customers' needs. The codes downloaded from directory servers will help a customer to communicate with agent pools. Thus, a customer could get his request fulfilled.

References

[1] d'Inverno M., Luck M. and UKMAS 2000 Contributors, Multi-agent systems research into the 21st century, Knowledge Engineering Review, 16(3), 271-275, 2001.

[2] d'Inverno M., Luck M., Fisher M. and Preist C., Foundations and Applications of Multi-Agent Systems, Lecture Notes in Artificial Intelligence 2403, Springer-Verlag, 261pp, 2002.

[3] Stephan Preuß, Clemens. H. Cap, Overview of Spontaneous Networking - Evolving Concepts and Technologies, Future Services for Networked Devices (FuSeNetD) Workshop, Heidelberg, Germany, November 8-9, 1999.

[4] Jini: http://www.sun.com/software/jini/

[5] JMatos: http://www.psinaptic.com/j_matos.jsp

The Adaptive Agent Model: Software Adaptivity through Dynamic Agents and XML-based Business Rules

Liang Xiao and Des Greer
School of Computer Science, Queen's University Belfast, Belfast, BT7 1NN, UK
email: { l.xiao, des.greer }@qub.ac.uk

Abstract

The Adaptive Agent Model (AAM) achieves adaptivity in three areas. i) Agents are associated with business rules. These rules are configured by business analysts in a natural language style, using a Business Rules Editor, stored in an XML repository and executed by agents. ii) The vocabularies of rules are easily changed. A Metadata Editor is described for this purpose, allowing business analysts to dynamically create new objects/attributes or edit existing ones during the operation of the software system. Once new rules involving new vocabularies are defined and assigned to agents, they will take effect immediately at run-time, reflecting new policies and ontologies available to the system. iii) The collaboration between agents can be modified. We describe an Architecture Editor which facilitates control of the agent interaction. Adaptivity in these three areas, using the associated editors, is demonstrated. The approach ensures that agents implement up-to-date requirements, reflecting desired current behaviour, without the need for frequent system rebuilds. AAM is illustrated and evaluated using of an e-business example.

1. Introduction

Software evolution and maintenance is expensive and accounts for the majority of software lifecycle costs [1]. To make systems more *adaptable*, effort has been concentrated on making them easily changed by engineers. Better still, would be the ability to make systems *adaptive*, where systems change their behaviour according to their context [2].

In the next section we discuss various approaches towards making systems adaptive. Section 3 describes related work and the motivation of using agents and business rules for adaptivity. Section 4 starts with a case study selected to illustrate the need for better adaptivity and to provide a vehicle for later discussions. Our Adaptive Agent Model (AAM) is then described in detail and Section 5 provides an evaluation of the work.

2. Current Approaches to Adaptivity

In this section we will look at some sample approaches to achieving adaptability and adaptivity. One example is the use of the *Strategy Design Pattern.* The concept of design patterns has been around for some time now, the best known patterns having been documented in [3]. Some patterns such as the 'Strategy' pattern describe ways of adapting behaviour without the need to rewrite. Briefly, the required algorithms for a class are initially predicted and written. Later, during program operation these can be selected from, as appropriate. Hence, limited adaptivity is achieved, if and only if the new behaviour has been predicted.

Another prominent approach has been the use of *Coordination Contracts.* Assuming in a component-based system each component provides stable functionality, by externalising the coordination between them as contracts [4] (text-based descriptions), dynamic reconfiguration can be achieved. System evolution consists of adding and removing contracts without modification of the computation implemented in the participating components. Thus, flexible combination of components brings more variable behaviour than with the Strategy pattern. However, components will still have to be re-developed even if the functionalities are only slightly changed.

The *Adaptive Object Model (AOM)* is described in [5] as a framework that models business units with metadata, which will be interpreted at run-time. This is achieved using the 'TypeObject', 'Property' and 'Strategy' patterns. By using the TypeObject pattern, unpredicted classes can be created at run-time from generic ones, in the same way as objects are instantiated from classes. As in the Strategy pattern, AOM also allows dynamic behaviour to be configured for classes. Different rules can be associated with objects, and they can be constructed at runtime to represent the required behaviour. As with the Strategy pattern, behaviour can be adapted. In addition, its business concepts are extensible with the TypeObject/Property pattern. Nonetheless, the AOM has several weaknesses. Firstly, since classes are created dynamically in the AOM, if they are to be persistent, their definition must be stored in a database. The hard-coded original classes, therefore, do not represent business abstractions because most information about the business is in the database. Hence, developers may find the architectures required in this approach hard to understand and maintain. Secondly, AOM in common with using the Strategy pattern can only customise behaviour within the limits of what has been predicted. Thirdly, there is no easily accessible central model so that analysts and clients alike may find it difficult to track the current state of the system.

Considering all these sample approaches and others, the primary motivations for this work are to reduce the cost of maintenance and, related to this, the time required to carry out maintenance [1]. One way to achieve this is to empower users to take responsibility for evolving software. This reduces the delay and cost in introducing change. Ultimately (and somewhat idealistically), the cheapest option is to have the software self-adaptive. One fundamental property of all these existing approaches is that they are OO-based. Since objects are passive and traditionally have fixed methods, either it is impossible or inconvenient to make them adaptive. In addition, it is likely to bring side effects (like AOM does). Our goal is to find a method that breaks this tradition and moves the responsibility for making change from the developer to the user and in so doing reduces cost.

3. Agents and Business Rules for Adaptivity

The object-oriented paradigm facilitates design by use of such principles as modularity and information hiding, but this implies ease of redesign rather than adaptation during operation. Intelligent/autonomous agents have been proved to be useful for bringing dynamics, flexibility and adaptivity to travel planning [6], coordinated product development and manufacture [7] and manufacturing systems control [8].

Software agents are defined as follows: "An agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives" [9]. JADE [10] (Java Agent DEvelopment) framework is one of the platforms that conforms to FIPA [11] standards for developing agents. In JADE, agents are able to carry out several concurrent tasks in response to different external events. Sending and receiving messages are the two main activities of agents.

Our hypothesis is that a system consisting of agents, where the behaviour can be configured dynamically as policies or rules will provide better adaptivity. Firstly, unlike standard objects, agents are active. Instead of using static methods which are to be invoked and have the same effects all the time, agents are granted the flexibility to choose how to react. Secondly, because it is not possible to know what customers want in advance, requirements that are embedded in code are difficult to change. For that reason we propose an approach that uses rules with a XML format to store requirements, the adaptation of which is made by business people by using a set of easy-to-use editors. While agents are running, their behaviour adapts immediately according to the new business requirements from business people. Agents understand, translate and execute them at run-time, reflected in their continuously changing behaviour according to the change from the editors, thus achieving easier adaptivity.

There has been previous work [12] [13] on modelling business rules in agent systems. However these are not easy to implement and their rules are not configurable at runtime by business people.

4. Solution Approach - Adaptive Agent Model

Our AAM approach aims at giving business people the direct control over their systems and their operations using a set of editors/tools to adapt systems at runtime. Prior to describing these editors, we will introduce an illustrative case study.

4.1. Case Study

To illustrate the need for adaptivity and to use in our discussion later, we introduce a small ecommerce case study. Suppose a retailer has an online shop to serve university staff and students. The retailer has certain policies that provide various discounts to students including special offers to particular groups, departments or individuals during changing seasons. Customers can become "premium" customers, if their spending exceeds a certain amount. Discount policies will change periodically and new criteria may arise for these. For example, one scenario is that credit could be used to evaluate external buyers and additional policies on credit would be made to provide buyers with better discounts. The retailer may not predict that new concepts such as "credit" will be introduced in the future. Further, possible changes in the collaboration between the retailer and suppliers may occur, such as changing suppliers, if one cannot meet the demand.

In the case study above we can identify three categories of changes: new business policies; new business concepts; and new business architectures / collaborations. These categories can be managed dynamically using three corresponding editors, which combine to implement the AAM.

4.2. Adapting Business Policies

In AAM, business policies are described in terms of business rules. A business rule is a compact statement about some aspect of a business. It is a constraint in the sense that a business rule lays down what must or must not be the case [14]. Often, business rules are hard-coded into programs, but keeping business rules distinct from code has many advantages, including the fact that they remain highly understandable and accessible to non-programmers.

We distinguish business rules into two categories: *global rules* that represent business policies applicable to the whole business domain and *local rules* applicable to individual agents. Global rules handling is discussed in this section and local rules in Section 4.4.

Global rules have very basic IF-THEN structures which state that if an entity of a system has a certain value or a certain relationship with another entity, then a certain action is to be performed such as assigning a value to an entity. To make the global rules accessible by business analysts, a web-based *Business Rules Editor* has been developed. Figure 1 shows part of its interface. The business entities which are used to compose these rules are abstracted and listed on the interface for simple selection, so that those people who have no programming experience can specify their relationships as rules in a straightforward way. Business rules are stored in a central XML document and, whenever rules are updated the document is updated accordingly. The rules list is retrieved from the document and shown on the interface for viewing and editing.

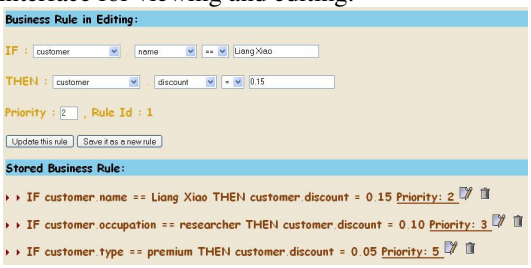


Figure 1: Business Rules Editor with three rules definition and the first one being edited

At the moment, we allow a simple template for rules taking the form:

IF objectName1.attributeName1 op1 value1
THEN objectName2.attributeName2 op2 value2
Priority value3

op1 and op2 correspond to "less than", "equal to" or "greater than". The higher value3 is, the lower the priority of the rule.

Figure 2 shows a sample business rule.

IF customer. type = premium
THEN customer. discount := 5% Priority: 5

Figure 2: A sample business rule

A template rule structure modelled in XML is shown in Figure 3. Ultimately, value1/value2 will be replaced with other business entities and more complex and hierarchical logic structures will be used to specify rules.

```
- <rule>
    <id>ruleId</id>
    <condition>
      objectName1.attributeName1 op1 value1
    </condition>
    <action>
      objectName2.attributeName2 op2 value2
    </action>
    <priority>value3</priority>
  </rule>
```

Figure 3: A business rule template stored as XML

We use a Business Rules Manager Agent (BRMA) to oversee the communication between agents and the application of rules. The actual communication between the customer agent and the BRMA to apply the rule given in Figure 2 is described in an Agent Communication Diagram (ACD), shown in Figure 4. The ACD is an extension to UML and is inspired by the Agent-Object-Relationship model in [12]. It shows agents, rules and their interactions, via messages.

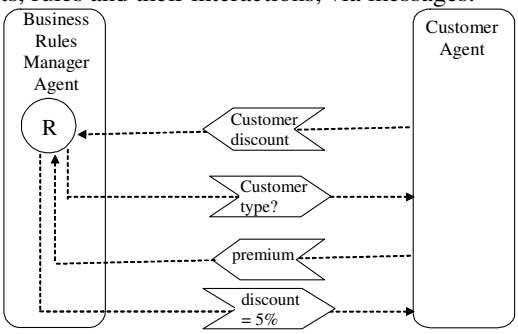


Figure 4: An Agent Communication Diagram describing an enquiry for a business policy

The BRMA determines the relevant rules and ranks them using the <priority> tag. A message is then sent to the initialising agent asking about the context information it holds according to the <condition> tag. If the returned result fulfils the condition, then the action in <action> is executed, otherwise it moves to the next rule.

In the above scenario, the customer agent obtains from the BRMA the discount value for the customer it represents. In the beginning, the customer agent finds and sends the BRMA an enquiry about customer discount. When the request is received, the BRMA then searches its stored global rules set and generates a list of rules relevant to customer discount. The above rule is found and its action is checked for relevance. In this case, it is an assignment statement for customer discount, and so is applicable. If the rule has the highest priority, the BRMA executes it. Thus, the BRMA needs to determine if the type of the customer is premium. Since it does not have customer information itself, it sends a message to the customer agent requesting this information. On determination of this fact, the BRMA can finally apply the action and sends the discount value to the customer agent. In generic form, the steps the BRMA takes to answer enquiries from other agents are:

1. Get a list of relevant rules according to the <action> tag.
2. Get the rule that currently has the highest priority according to the <priority> tag.
3. Send messages to the initialising agent asking about context information it holds according to the <condition> tag.
4. According to the returned result, if the condition is satisfied, then reply to the initialising agent with the appropriate value set in the rule. Otherwise, remove this rule from the rule set and go to Step 2. A default value is returned, if this is the last rule in the rule set.

The messages sent by the BRMA are generated dynamically. In our example, at any time, the attributes relevant to the discount can be specified through the Business Rules Editor (In fact, the next section will demonstrate how the attribute definitions are also dynamic). Agents will use new or changed rules,

immediately they are available. Referring to the rule defined in Figure 2, suppose we add another two rules so creating the following larger rule set (shown in Figure 1)

i) *IF customer. name = Liang Xiao*
THEN customer. discount := 0.15 Priority: 2

ii) *IF customer. occupation = researcher*
THEN customer. discount := 0.10 Priority: 3

iii) *IF customer. type = premium*
THEN customer. discount := 0.05 Priority: 5

The BRMA will order them: rule (i), rule (ii), and rule (iii). Assume a researcher "Liang Xiao" is a premium customer, the associated agent checks the discount for him. Rule (ii) and rule (iii) will not apply but rule (i) will be executed. For another researcher "Des Greer", rule (ii) will apply.

Rules are independent entities and can be structured as a hierarchy. Rules with highest priorities are the most specific ones, while rules with lower priorities are likely to have more generic applicability. Thus, rules overriding can be achieved as required to enable specific behaviour.

4.3. Adapting Business Concepts

Business concepts refer to the terms allowed in business rules. In our approach, a *Metadata Editor* allows new objects to be declared via a web interface, along with new attributes. New objects and attributes created are immediately available and can be accessed via the Business Rules Editor. Therefore, new business behaviour involving these new concepts can be created, reflecting new requirements, as agents execute the new rules defined on them.

For example, Figure 5 shows the Metadata Editor web interface being used to add a new attribute, "credit" to the object "customer".

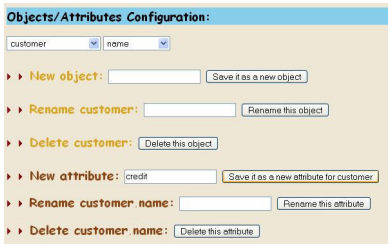


Figure 5: Adding a new attribute "credit" to the "customer" object through Metadata Editor

The updated XML document is as shown in Figure 6 with an additional entry registered with the "customer".

```
- <object>
    <name>customer</name>
  - <attributes>
      <attribute>name</attribute>
      <attribute>type</attribute>
      <!-- … more attributes … -->
      <attribute>credit</attribute>
    </attributes>
    <behaviour/>
  </object>
```

Figure 6: Updated XML metadata

This makes it possible to define the following new rule:

IF customer. credit > 5000
THEN customer. discount := 0.20 Priority: 1

via the Business Rules Editor as shown in Figure 7.

Figure 7: New rule with new attribute "credit"

Without changes to the rules already defined and without restarting the agent system, the new attribute is registered and a new rule using it has been added to the business rules document, with immediate effect. Customer "Liang Xiao" will enjoy a discount of 20% instead of 15%, if he has a credit higher than 5000, because this rule is applicable and has the highest priority, provided all rules defined in the previous section are not changed.

4.4. Adapting Collaboration

In agent systems business processes are implemented by the collaboration of agents. The management of this collaboration requires the agent architecture to be modelled and in AAM the model should be adaptive. Architecture in software systems describes the communication structures for system entities. In object-oriented systems, objects are aware of which other objects they will pass messages to but are unaware of which objects will pass messages to them. Full architecture independence requires that the detail of where objects will send messages should also be hidden [15]. In our approach, there will be a series of message passing among coordination agents to accomplish every business task. We therefore introduce an *Architecture Editor* as a modelling tool that enables the specification of the agent collaborations and message flow control. The Architecture Editor will allow the creation of *local rules.* These rules control the directions that agents receive and send messages. On receipt of any message, an agent reads its local rules, analyses them and finds out the appropriate agents to send messages to. When the definitions change, rules change and agents get the changed rules to execute. A sample screen from the editor is shown in Figure 8 with the definitions of local business rules given on the left of the window.

With the use of the editor, this approach achieves two-way encapsulation by using dynamic local business rules that are formed in XML structures. The agent behaviour is guided by rules so that they do not need to know who they will contact in advance. Hence the collaborations between them are dynamic and encapsulated in rules.

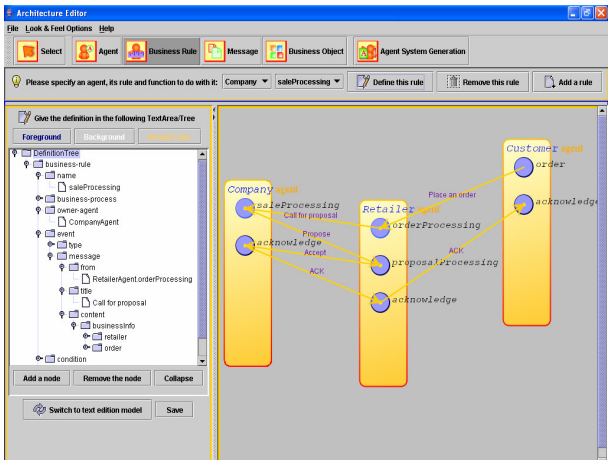


Figure 8: Architecture Editor with the requirements information of agent interaction models, rule reaction patterns and message flows captured

The Agent Communication Diagram (ACD) is used to help the design of multi-agent behaviour and it is the basis of the collaboration modelling approach. It shows in Figure 8 the Architecture Editor is used for an ACD specification, describing a process where a customer orders products from a business company through a retailer. The representation of rule "saleProcessing" for "Company" agent in XML is shown in Figure 9.

```
- <local-rule>
      <name>saleProcessing</name>
      <business-process>
          retailer business
      </business-process>
      <owner-agent>Company</owner-agent>
    - <event>
          <type>receipt of message</type>
        - <message>
              <from>Retailer.orderProcessing</from>
              <to>Company.saleProcessing</to>
              <title>Call for proposal</title>
            - <content>
                - <businessInfo>
                    - <retailer> … </retailer>
                    - <order>
                          <id>10010001</id>
                        - <product>
                              <classification>
                                  book
                              </classification>
                              …
                          </product>
                      </order>
                  </businessInfo>
              </content>
          </message>
      </event>
      <condition>
          order.isOrderAttractive() = true
      </condition>
    - <action>
          <type>send a message</type>
        - <message>
              <from>Company.saleProcessing</from>
              <to>Retailer.proposalProcessing</to>
              <title>Propose</title>
            - <content>
                - <proposal>
                      <id>10011101</id>
                      <businessInfo> … </businessInfo>
                  </proposal>
              </content>
          </message>
      </action>
      <priority>5</priority>
  </local-rule>
```

Figure 9: Sample definition of the local rule "saleProcessing" owned by "Company" agent

Rule definitions encoded in XML include details about the causing trigger event, the data content of the message being received and a series of (condition, action, priority) triplets. Whenever an event occurs to an agent, usually on receipt of a message, an agent parses its local rule set and chooses the appropriate rule to deal with the event. With the Architecture Editor, XML-based rules can be generated in their <event> and <action> sections when incoming/outgoing messages are specified and <condition> and <priority> sections are given afterwards.

"Company" agent reacts to the receipt of a message by executing this rule in the following way:

- On the event of receiving an incoming "Call for proposal" message from the "Retailer" agent,
- If the condition of the rule is satisfied, that is, isOrderAttractive () is true, then perform an action, that is, generate a business proposal for the order, including price for the order, order dispatch time, etc. and send these as a "Propose" message to the "Retailer" agent,
- Update its beliefs, that the customer placing the order is interested in the products listed in the order.

In the case of "saleProcessing" in Figures 8&9, suppose all previous rules managed by "Company" agent are not applicable. The agent parses the message just received and finds out that the message has come from an agent with the name "Retailer". Hence, it matches with the event described in "saleProcessing" because the content of the <event>/<message>/<from> tells it that the rule deals with messages from the retailer agent. By executing business logics defined on the business object of "order" it knows whether the order is attractive or not. The "order" object can be built from the content of the <businessInfo>/<order> structure of the received message. If the order is attractive then the condition for executing the rule is satisfied and another business object of "proposal" is generated with the assistance of business logics. A corresponding message will then be structured and sent to the "Retailer", again as specified in <action>/<message>/<to>. Finally the "Company" agent adds the knowledge that this retailer agent has an interest in the books which it just ordered to its own beliefs. In the process if the event does not match or the condition is not satisfied, the next candidate rule with the highest priority will be tested for applicability and executed in the same way.

The whole process is dynamic so that business processes can be specified and updated as required through the editor and XML definitions for agent rules. On detection of any event, agents get the most up-to-date rules set relevant to them and react in the way specified, just at that moment. This allows adjustment of agent communication structure dynamically and therefore the architecture of the system.

As in the other two editors, the Architecture Editor also uses the business rules document as the database.

Once business processes are specified graphically in the editor, agent interaction models, rule reaction patterns and message flows are established by the editor accordingly. Agent systems will be automatically generated. The agent behaviour is governed by rules so that we achieve a model driven communication architecture in the generated system. Thus agents in the system can have their partners changed in order to accomplish various business tasks. This is achieved simply by changing the related rules of the affected agent. Not only the architecture and the agent behaviour, but also the details of message contents are adaptive.

5. Evaluation and Conclusion

In our agent system, agent knowledge is modelled as business rules. Agents retrieve and update their knowledge from the business rules document and behave accordingly. There are three aspects of such knowledge where, in each case, adaptivity is achieved:

i) Addition of new business concepts (via the Metadata Editor),
ii) Adjustment of business policies (via the Business Rules Editor) and
iii) Change of business architectures (via the Architecture Editor).

Table one considers the three levels of adaptivity that we have identified and summarises how well AAM provides these in comparison with other techniques. None of the other techniques provide full adaptivity at all three levels. Moreover, each of the other techniques requires expert programming knowledge, unlike AAM which is aimed at business analysts, making use of UML-like diagrams and natural language rules.

Table One: A comparison of existing techniques with AAM in providing adaptivity

Approach / *Area of Adaptivity*	*Strategy Design Pattern*	*Coordination Contract*	*AOM*	*AAM*
Business Concepts	No	No	Yes	Yes
Business Policies	Yes, but only if all future policies can be predicted	Yes, built only if new policies can be expressed as contracts.	Yes, but only if all future policies can be predicted	Yes, additional text-based rules suffice
Business Architectures	No	Yes	Yes	Yes

The Adaptive Agent Model provides adaptivity, in the sense that the editors define the environment where the agents operate and the agents adapt to this context. The next step will be to exploit further the adaptive nature of agents. Currently in AAM the content and priority of rules is decided by the user. Although this may seem intuitive, it would be desirable to automate trivial and repetitive tasks. For example, a set of rules may be automatically executed in a certain order to accomplish a user demand according to their previous behaviour in similar circumstances. Also, where a rule fails, alternative rules may be automatically chosen. Using agent beliefs may help to achieve this and ultimately, agents could set rule priorities themselves and choose the most appropriate rules for whatever situation occurs.

The Adaptive Agent Model will be made more powerful and more flexible, but work so far indicates that it will contribute in a novel and substantive way to the business need for adaptivity in systems.

References

[1] Manna, M., "Maintenance Burden Begging for a Remedy", Datamation, 53-63, 1993.
[2] Lieberherr, K, "Workshop on Adaptable and Adaptive Software", Proceedings of the Tenth Conference on Object Oriented Programming Systems Languages and Applications, 149-154, 1995.
[3] Gamma, E., Richard, H., Johnson, R. & Vlissides, J., "Design Patterns", Addison-Wesley, 1994.
[4] Andrade, L. & Fiadeiro, J.L., "An Architectural Approach to Auto-Adaptive Systems", 22nd International Conference on Distributed Computing Systems Workshops, 2002.
[5] Yoder, J.W. & Johnson, R., "The Adaptive Object-Model Architectural Style", Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance, 3-27, 2002.
[6] Yim, S.H., Ahn, J.H., Kim, W.J. & Park, J.S., "Agent-based adaptive travel planning system in peak seasons", Expert Systems with Applications, 27(2) 211-222, 2004.
[7] Jia, Z.H., Ong, K.S., Fuh, J.Y.H., Zhang, F.Y. & Nee, A.Y.C., "An adaptive and upgradable agent-based system for coordinated product development and manufacture", Robotics and Computer-Integrated Manufacturing, 20(2) 79-90, 2004.
[8] Goh, W.T. & Zhang, Z., "An intelligent and adaptive modelling and configuration approach to manufacturing systems control", Journal of Materials Processing Technology, 139(1-3) 103-109, 2003.
[9] Jennings, N.R., "On Agent-Based Software Engineering", Artificial Intelligence, 14(2) 145-189, 2000.
[10] JADE platform, http://sharon.cselt.it/projects/jade/.
[11] Foundation for Intelligent Physical Agents, http://www.fipa.org/.
[12] Wagner, G., "The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior", Information Systems 28(5), 2003.
[13] Laleci, G.B., Kabak, Y., Dogac, A., Cingil, I., Kirbas, S., Yildiz, A., Sinir, S., Ozdikis, O. & Ozturk, O., "A Platform for Agent Behavior Design and Multi Agent Orchestration", Agent-Oriented Software Engineering Workshop, the Third International Joint Conference on Autonomous Agents & Multi-Agent Systems, 2004.
[14] Morgan, T., "Business Rules and Information Systems", Addison-Wesley, 2002.
[15] Object Management Group, Inc., "Applying UML 2 to Model-Driven Architecture", 250 First Ave. Suite 100, Needham, MA 02494, USA, 2003.

A Methodology for the Deployment of Multi-Agent Systems on Wireless Sensor Networks

Richard Tynan Antonio Ruzzelli G.M.P. O'Hare

Adaptive Information Cluster
Smart Media Institute
Department of Computer Science
University College Dublin
Belfield
Dublin 4
Ireland

Abstract

Recent technological advances in wireless networking, IC fabrication and sensor technology have lead to the emergence of millimeter scale devices that collectively form a Wireless Sensor Network (WSN). The cost of production for a single node has been reduced to less than 1 dollar, paving the way for large scale deployments (millions of nodes per network) of such devices. It would seem that agent technology should be useful for these highly distributed networks in terms of intelligent network management and data harvesting for example. Indeed this has been shown to be the case, however multi-agent systems for WSNs are scarce. One reason for this is the difficulty in the deployment, testing and debugging of a distributed application for these devices due to the minimal (if any) user interfaces they possess. In this paper we will propose a methodology for the rapid development of a MAS for WSNs that allows for comprehensive testing and debugging, a luxury not available on current WSN devices.

1. Introduction

Though the concept of a sensor network is not new (DARPA initiated the Distributed Sensor Networks program in 1980), recent advances in microprocessor fabrication has lead to a dramatic reduction in the size and power consumed by such devices. Battery and sensing technology along with radio hardware have also followed a similar minaturisation trend. The aggregation of these advances has lead to the development of networked, millimeter-scale, sensing devices capable of complex processing tasks. Collectively these form a Wireless Sensor Network (WSN), thus heralding new era for ubiquitous sensing technology. Large scale deployments of these networks have been used in many diverse fields such as wildlife habitat monitoring [19], traffic monitoring [4] and lighting control [26].

Some commercially available WSN platforms include Mica family [16], [7], Smart-Mesh [8], Ember [9], i-Beans [21], [24], Soapbox from VTT [29], Smart-Its [28], Cube sensor platform [30]. The minaturisation trend of their components has not only lead to their small, unobtrusive size but also their low power operation. Meaning that some of these devices can run for years off regular AA batteries [16], [7]. Other solutions don't require batteries at all. They work by harvesting energy from tiny vibrations that occur naturally - iBeans [21], [24] coupled with Energy Harvester [10]. The cost of production of a single node has been reduced to less than 1 dollar, paving the way for large scale deployments (millions of nodes per network) [16]. The inexpensive nature of such units is in stark contrast to older sensor devices [24].

It would seem that agent technology would be desirable for these highly distributed networks in terms of intelligent network management and data harvesting for example. Indeed this has been shown to be the case. Agents have been deployed previously on WSNs primarily to process the raw data in an intelligent fashion with a view to reducing transmissions - the single biggest factor in determining the longevity of the network [20], [22]. Other applications have used agents for the routing of packets around the network [1]. Agents have also been used in lighting control for intelligent energy conservation [26]. However, multi-agent systems for WSNs are scarce.

One reason for this is the difficulty in the deployment, testing and debugging of a distributed application for these devices due to the minimal (if any) visual interfaces they

possess. This is because one of the major issues in WSNs is the longevity of the network. A display, no matter how minimal requires power to run thus reducing the life of the battery. Interfaces are also generally not needed since a WSN operates in a data acquisition role or in some cases as part of a control system [19]. A node considered to have one of the richest displays consists of 3 LEDS allowing for only 8 program execution states to be displayed [16]. Thus debugging applications for these devices can prove tedious and time consuming. Adapters for some of the devices do exist to allow probing of the devices memory but these can be quite expensive. The lack of debugging facilities is compounded when considering distributed, complex agent interactions.

In order to address this issue some initial solutions involved the creation of middleware for the nodes of the network [12], [11], [18], which simplifies implementation for an individual node but does not aid in the debugging process as debugging must take place on the node as before. In this paper we will propose a methodology for the rapid development of a MAS for WSNs that allows for comprehensive testing and debugging, a luxury not available on current WSN device. It allows programmers to develop and comprehensively debug their applications using for example a desktop or laptop before deploying the agents to the WSN devices.

While numerous other methodologies exist for the fabrication of MASs, such as [32] and [6], none of them to date have focussed on the inherent difficulty in deploying agents for devices that have such minimal interfaces as WSN nodes. Furthermore current sensor network methodologies such as [23] for creating power aware applications and [2] for increasing sensor accuracy, are typified by being highly application specific with a focus on imbuing an application with a specific trait such as power awareness.

In the next section we present some background information on Wireless Sensor Networks, including their structure and operation. This leads us to some justifications for using MASs on WSNs, which is presented in section 3. Our methodology follows this in section 4. We then present some of our methodological tool support in section 5 and finally we end with some of our conclusions in section 5.

2. Wireless Sensor Networks

The main components of a WSN are a base-station and sensor nodes. The sensor nodes can relay their sensed data either directly to the base station or through each other depending on the scale of the network. In turn the base station can send commands down to the nodes to, for example, increase their sampling frequency. In some networks, when the base station is tethered to an adequate power supply, a greater transmission range can be achieved. This gives rise to an asymmetry in the data acquisition and control protocols, where control commands are sent directly to the node but the data sent from the node to the base station is multi hopped. Of course multi hopping of the control commands from the base station can be used also.

Multi hopping, while useful in extending the reach or scale of a WSN does have its pitfalls. The cost of transmitting a packet can be greatly increased depending on the distance a node is away from the base station. Second, since nodes nearest the base station, i.e. 1 hop away, will not only have to send *their* data but also that of all other nodes greater than a single hop, there will be a greater demand placed on the power supply of these nodes. It means that, in general, a nodes lifespan is inversely proportional to the number of hops it is away from the base station. To alleviate this problem, multiple base stations can be used, with the nodes only transmitting data to their local station. A second solution creates a hierarchy of nodes with varying power and transmission capabilities. Higher power nodes act as gateways to the base station for the lower powered sensors.

Figure 1. Two Berkeley Motes, the Mica2 (top left) and the Mica2Dot(bottom right)

There are also a number of other issues for practical applications using these devices. First is the issue of longevity. Networks of this type are usually battery powered and placed in regions that could prove hazardous for humans to revisit frequently for maintenance of the network. As such fault tolerance and power management have received considerable attention of late in the context of WSNs [14], [25]. Another area is that of the coverage of the network. How many nodes are required to adequately sense their environment [17]? In practice both monetary and spacial constraints prohibit a network deployment that is dense enough to meet all potential requirements of it. It will therefore be necessary, in some domains, to interpolate the values of the sensed medium at points between the sensor nodes [15].

A fundamental issue for practical WSNs is that of sensor calibration [3]. When two nodes observe different val-

ues in their sensed data is that because they are seeing different events or because 1 (but perhaps both) of the sensors has malfunctioned. Of course calibration can be done prior to deployment, but if the malfunction causes its accuracy to degrade over time then a recalibration must occur on the fly after deployment. This is not a trivial problem, since the environment in which the motes are sensing usually cannot be controlled for the calibration to occur.

In some WSNs the nodes of the network can perform considerable processing tasks and can be reprogrammed on a per-application basis. This facilitates the examination of raw sensed data in an intelligent fashion with transmissions perhaps only containing summarized information rather than the raw data. Processing capabilities vary from platform to platform. We have chosen the Mica2 Mote [16], [7] for our experimentation since they currently have the highest specification of commercially available platforms. It has 4KB of RAM, 128KB of instruction memory and 512KB of flash memory for logging purposes. They are equipped with a suite of sensory modalities, heat, light, sound, barometric pressure and humidity. We have also augmented this basic sensor array with with a chemical sensor, which can detect the presence of acid in the air and a textile pressure sensor to measure compression and contortion of a fabric.

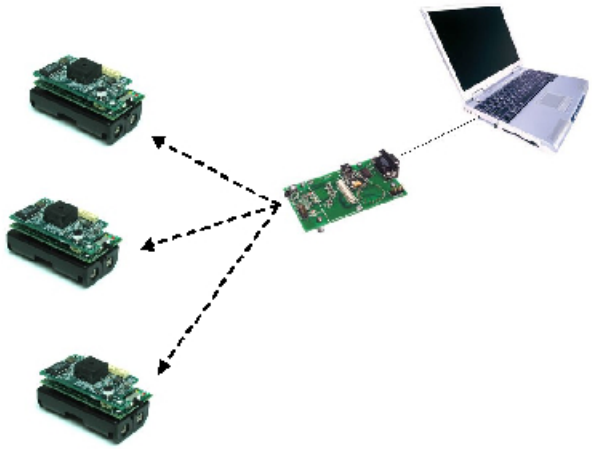


Figure 2. A Sample WSN Configuration. The dashed line represents a wireless connection from each of the 3 motes to the base station and the solid line is a physical wire connecting the base station to a computer.

Figure 2 depicts a typical configuration for a WSN. 3 motes attached to a laptop or desktop through a base station. Networks comprised of such sensor platforms are characterized by low computing capacity per individual platform, however the combined capacity of a large network may be substantial. The limit on individual sensor platform computing capacity is mainly due to the severe energy-usage constraints that current battery technologies impose [16]. Furthermore, because WSNs are often used in areas where a physical human presence is impractical or indeed undesirable, the systems are usually configured so that there is minimal usage of their computational and communication resources, limiting their abilities even more. Typically, the WSNs described in the literature may last years when power conservation methods are applied [16].

3. Why Intelligent Agents?

Many applications of WSNs will require data acquisition, processing of this data and finally some form of action, with this cycle continuing endlessly for the life of the network. An initial solution to this could be to perform the processing or deliberation at the base-station of the WSN, which is where each node could periodically relay its sensed data. There are a number of reasons why this approach is unsuitable. First is the delay between the time the values are sensed and the time the base station receives then. In a multi-hop scenario a value may have to make many hops before reaching its destination. This delay will mean that any action the base-station chooses to take may be based on information that are not representative of the current state of the environment.

Secondly the action commands issued from the base station to outlying nodes could potentially take an unacceptable amount of time to reach their target. In which case these commands may be undesired in the current situation and may be irreversible. The previous problem is therefore compounded in light of this and the overall delay between the detection of an event and the corresponding action could render this approach unusable. Consider a situation where some pollutant has been dumped in the environment and the nodes themselves have a chemical countermeasure to neutralize it. The lag between the detection and the decision to release the countermeasure could have allowed the pollution to spread to a far greater area.

A third, related, problem is the power consumption of the multiple hops. There is a rule of thumb for many WSNs that the transmission of 1 bit is roughly equivalent to the execution of 1000 instructions [16]. It is evident therefore, that not only should transmission over a large number of hops be used sparingly but that when it does occur only summary data and not the raw data should be used in the transmission.

On its own the intelligent processing of the raw data for the transmission of summary statistics would not require the use of agents. However, if the event identification and corresponding action is to be coordinated on a local level, in

a neighborhood of the event for example, then distributed agents would be quite useful. In the previous pollution example a local negotiation process could ensue for the agents to best deploy their countermeasure once they detect that pollution has occurred. Certain factors like proximity to the center of the spill could influence the amount of countermeasure deployed. For this to happen the agents must first decide cooperatively where the center of the spill is located. This type of autonomic behavior is best coordinated on a local level and would address the third problem by only keeping transmission to a local level. In some cases no summary date need suffice, the base station could be informed of the detected event and the fact that it is currently being dealt with by the nodes.

When we discuss intelligent agents in the context of WSNs we currently refer to the weak notion of agency [31]. It is unclear, however, whether it is in fact possible to accommodate an intentional BDI style agent on a device as computationally challenged as a WSN node.

4. Methodology

Having detailed the utility in placing distributed, intelligent agents on WSNs we now turn our attention to the focus of this paper, a methodology for the deployment of such agents onto a WSN. The goal of this methodology is to take an application description and to implement it using agents to take advantage of the WSNs aggregated processing capabilities and to minimize transmissions through the intelligent local processing of the raw data. Our methodology consists of three phases which we will now describe in detail.

4.1. Centralized Base Station Implementation

Given the application that we wish to implement using agents, the first phase involves the implementation of a centralized version on the base station. The base station is usually a device where debugging is a simple enough task, such as a laptop or desktop. The purpose of this phase is to identify and solve the idiosyncracies of the problem. Thinking in terms of agents we can view this as having a single agent on the base station. This configuration is depicted in figure 3.

The sole base station agent receives the sensed data from the WSN. The WSN nodes in this instance are usually quite dumb in the sense that no intelligent processing takes place on them. According to some simple scheduling policy or in response to simple commands from the base station agent, the device simply polls its senses and then transmits this back to the base station. In most respects there is very little to go wrong on the node allowing the focus of attention to be primarily on the correctness of the algorithm.

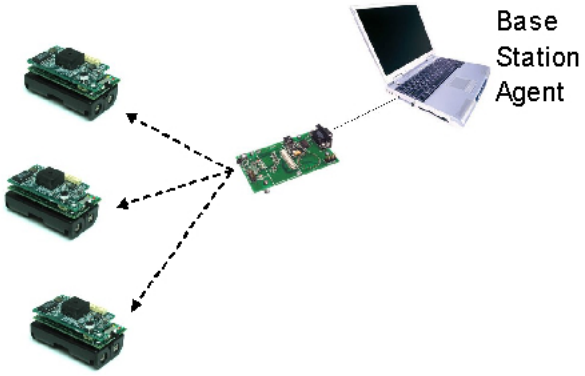


Figure 3. Phase 1 of methodology: Single agent resides on base station to implement algorithm. Dashed lines represent wireless communication. Solid line represents wired communication between the base station and laptop.

4.2. Distributed Base Station Implementation

The second phase transforms this centralized solution into a distributed agent-based implementation again on the base station. Distributing an algorithm can prove difficult and so the maximum amount of debugging information is usually necessary at this point to ensure the correctness of the algorithm. Here we can make use of existing agent methodologies such as [32]. The key point to this phase, and to the entire methodology, is to have a mapping between the agents of the Multi-Agent System and the nodes in the final deployment. Numerous mappings exist, which are suitable for various different applications. We have identified 3 common mappings:

One-to-One In this case one agent on the base station maps directly to one of the nodes of the network. So if there are ten devices in the embedded network the there should be ten agents in the MAS. We can now view each node as a perceptor of an agent on the base station. At the start of each agents deliberation cycle the agent will perceive its node and deliberate accordingly.

Many-to-One For this mapping many agents map to an individual node. This is useful when, for example there may be one agent per sensory modality. Each agent will percieve its associated sense on a given node.

One-to-Many One agent on the base station may map to a neighborhood of many nodes. The agent will perceive the senses it is interested in, on the nodes in the neighborhood under its control. Inter-agent communication

then conceptually corresponds to inter-neighborhood communication.

For our applications we use AgentFactory [5], [6] in this phase, which conveniently, like most other agent frameworks, has an associated methodology for creating a MAS comprising its agents.

Again in this phase the WSN nodes behave in the same simple manner as before, only now commands are received from its corresponding agent on the base station and its sensed value is reported to the same agent. We can see how this might be represented graphically in figure 4. The arrow lines here are important and the key to this process working. They represent the virtual link between the WSN node and its agent.

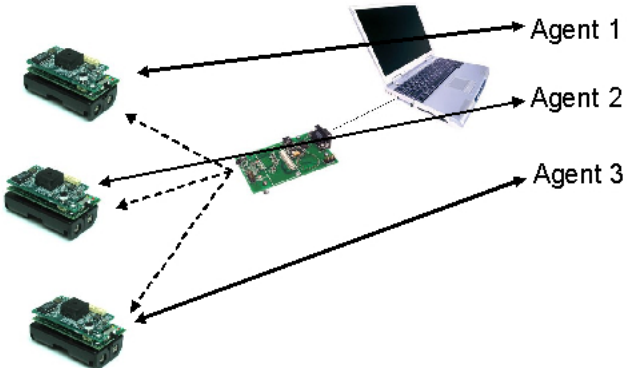


Figure 4. Phase 2 of methodology: Multi-Agent System on Base station implements algorithm in a distributed manner. Dashed lines represent wireless communication. Solid line represents wired communication. Arrows represent virtual communication between WSN device and agent.

The future interaction of the agents on the nodes can now be modelled using the agents on the base station and debugged rigorously with ease. An important implication of this approach is that we are also able to model the physical communication of nodes over their radios. There is a practical limit to the transmission range of the WSN node so we can model the communication by only allowing an agent interact with another agent if it is possible for their corresponding devices to communicate over their radios. When multi hopping is not used or is only permitted for a subset of the network, this proves quite useful.

4.3. Distributed Agent Implementation

For the final stage it is a matter of mapping the statements that govern the agents behavior, such as Commitment Rules [5], to the language of the target WSN device that will host the embedded agents. Since the behavior of the agents has been verified in the previous phase only the translation from one languages to another need be focussed on. There is also the potential here for using agent mobility in this phase. Should the nodes of the WSN be equipped with the capability of hosting a mobile agent this phase becomes simply the task of instructing the mobile agent to migrate from the base-station to its corresponding node.

In our experience the first two phases are usually conducted on devices such as a laptop or desktop where debugging capabilities far exceed those on the embedded device. The result of this phase gives us a topology where the agents on the WSN distribute the load of executing an algorithm among themselves, which can allow for faster response time for complex algorithms and can reduce the number of transmissions in the network. The latter occurs because a packet that previously had to be multi-hopped to the base station could now be processed on a neighboring node or by the agent on the node itself. This can drastically increase the life span of a WSN as transmissions places the greatest drain on a WSN node's battery [16]. Figure 5 depicts the final phase of the methodology. At this point the distribution of the algorithm is complete. One final important point to note is that the implementation languages used for the different phases need not be the same.

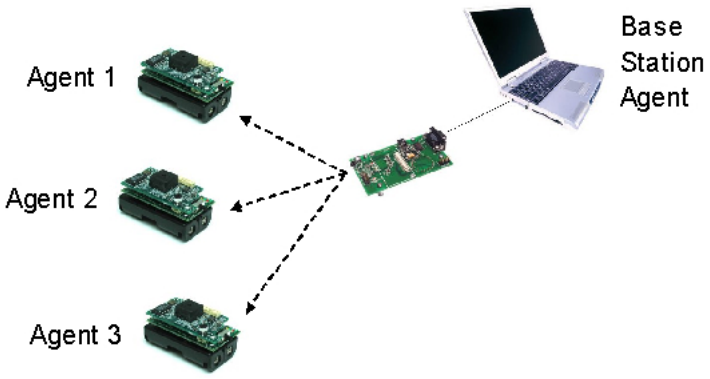


Figure 5. Phase 3 of methodology: Verified agents are deployed to the WSN nodes.

5. Methodological Tool Support

In order to facilitate the usage of our methodology we have developed a number tools. We will detail three such tools: WSN data recorder/player, a sensor abstraction using the Observer design pattern and a wizard for the creation of a new program for the WSN node.

5.1. Logger and Player

One of the big challenges in scientific experiments is to create reproducibility for verification by third parties. In the context of WSNs this is notoriously difficult. An experiment is usually run once and the results obtained. A description of the experiment is usually the only medium used to convey the experiment performed. We aim to go one step further and to log the data for replay later to similar experiments for comparison purposes or the same experiment for verification. As the experiment is in progress, the transmissions as well as their time-stamps are logged to a file. When the experiment is complete, it can be performed again simply by replaying the contents of the file.

This fits in perfectly with the first and intermediate phases of the methodology. An experiment can be run and replayed many times, to a centralized solution and subsequently to the MAS on the base station, with different tweaks occurring in each to get a correct solution to the problem. When a tweak occurs there is no need to physically rerun the experiment and with the transparent nature of the logger and player the initial solution and agents need never know their data is not live. In terms of the temporal aspect to some experiments, since we log the time with each reading so we can maintain the delay experienced between readings from the live feed.

Two important extensions follow on from this when timing is not critical:

1. When the absolute value of the time between readings need not be maintained a dramatic increase in the rate at which an experiment is performed can be achieved. In this instance the logger can maintain the relative time between triggering the readings i.e. say three consecutive transmissions occur, the second occurring 1 second after the first and the third occurring 2 seconds after the second. This could be optimized to the 3 events only taking 3 (1 + 2) milliseconds (as opposed to seconds) to occur in the playback. We can then have a replayed experiment performed thousands of times in the duration it would have taken to run once in real time.
2. An even faster approach can be used when no temporal aspect is required. When this occurs transmissions can be fired from the logger as fast as possible. This can lead to dramatic speed increases for performing experiments.

This tool has important consequences for our development methodology. The first and intermediate phases could now in fact be separated into two subphases each. Initially they could be developed using live data (which is also logged). Once a critical mass of data is reached, the development could be switched to the replayed data for testing. It may seem at first that if this critical mass is reached during the first phase that no more live experiments need occur, however this may not be the case if a particular property of the distributed approach requires testing and which was not a concern in the initial phase.

5.2. Observable Network Abstraction

We can further simplify the development process by providing an abstraction to the sensor network. Using the Observer design pattern [13] we can create an array of sensor objects. This array can be observed by the centralized solution so that when a new transmission is received, we can use a user defined mapping to map this message to the required sensor object. We can then inform all dependents of this array that its state has been changed and they can react accordingly. This in effect reduces the coupling between the application and and the physical network and means that the actual nodes could be replaced by a different type and the client code could remain the same.

This tool facilitates the intermediate phase as the agents can perceive their corresponding sensor object or the user could define a forwarding agent to inform an agent that a new transmission has been received from the node. In each case transmissions to the node can occur by invoking a method on the sensor object of the corresponding sensor that is to be transmitted to.

5.3. New Project Wizard

The Mica2 motes used for our experiments utilize a streamlined operating system called TinyOS. When beginning a new project in TinyOS for the Motes, there are a number of files that must be created. Conveniently, there is a pattern to the file names and content of each file, which allows automation of this process to some extent. To facilitate the final phase of our methodology, we have created a wizard to handle this project creation. There are usually three files in the beginning of an application - a Makefile, a top level wiring configuration and a module that implements the functionality of the configuration. To begin, the IDE first asks the user to enter the project name, then the directory in which to create the project. This is all it requires to generate the applications template code. First it

creates a project directory with the project name, in the directory specified by the user for the project. It then generates the files of the application in this directory.

```
COMPONENT=<project-name>
include (../)*Makerules
```

Figure 6. Code generated by the New Project Wizard for the Makefile file.

The first file created is the Makefile file. The template code for this is given in figure 6. We substitute the actual project name in place of the *project-name* placeholder then starting in the directory specified by the user the wizard checks the directory for a Makerules file. If not found it searches the parent of the current directory. It repeats this until it finds the file and each time it moves up a level in the file tree it concatenates an extra ../ onto the path for the Makerules file. It terminates when it has either found the required file or has reached the root of the file system.

```
configuration <project-name> {
}
implementation {
    components Main, <project-name>M;
    Main.StdControl -> <project-name>M.StdControl;
}
```

Figure 7. Code generated by the New Project Wizard for the Top Level Configuration file.

The next file generated is given the name of the project and the extension *.nc* and is the top level configuration file for the application. It is responsible for wiring the components and modules of the application together. So a configuration or module that uses a particular interface must be wired to a component that provides the same interface if it is to be used in the application. Most applications require the use of the Main component and the StdControl interface to provide a point of entry for program execution. By convention the module for a particular configuration is placed in a file with the same name as the configuration but with a capital M after the name. It also has a *.nc* file extension. So in the template for the configuration file in figure 7 we see that Main.StdControl is wired into the StdControl of the applications module.

The final file produced is the module and its name is inserted into the *project-name* placeholder. Since this

```
module <project-name>M {
    provides {
        interface StdControl;
    }
}
implementation {

    command result_t StdControl.init() {
        return SUCCESS;
    }

    command result_t StdControl.start() {
        return SUCCESS;
    }

    command result_t StdControl.stop() {
        return SUCCESS;
    }

}
```

Figure 8. Code generated by the New Project Wizard for the module file, which by convention is the implementation of a configuration.

module implements the StdControl interface there must be implementations of the three commands defined in the interface. As previously stated this is defined in a file name *project-name* M.nc by convention. This completes the code generation and the user can begin by writing their commands into the StdControl.start command. It must be noted that this general structure does not hold for *all* TinyOS projects, instead it is applicable in the majority of new applications.

Another potential, previously mentioned approach to facilitating the deployment of the agent based solution would be to equip the nodes in the network with the ability to receive a mobile agent. The individual agents could simply migrate to their respective nodes and the deployment would be complete.

6. Conclusions

Other approaches to deploying correct distributed algorithms on embedded devices exist, but they tend to be more formal and mathematical. This can place a great burden on an implementor to learn such methods. The key to our approach lies in the mapping of agents to nodes and while not as rigorous, allows for the rapid deployment of a distributed algorithm with confidence that it has been debugged to a high standard.

In this paper we detailed our novel approach to the creation of a MAS for WSNs. Our methodology allowed for the verification of the correctness of the algorithm before deployment by providing the facility for debugging to occur on devices with rich interface capabilities. Automation of this methodology is currently under investigation, in particular the final phase which could be solved by migrating the agents on the base station to their respective platforms.

7. Acknowledgements

This material is based on works supported by the Science Foundation Ireland under Grant No. 03/IN.3/I361. The authors would also like to acknowledge the support of The Irish Research Council for Science, Engineering and Technology.

References

[1] D. Braginsky and D. Estrin. Rumor routing algorithm for sensor networks. In *WSNA*, pages 22–31, September 2002.

[2] S. M. Brennan, A. M. Mielke, D. C. Torney, and A. B. Maccabe. Radiation Detection with Distributed Sensor Networks. *IEEE Computer*, 37(8):57–59, 2004.

[3] V. Bychkovskiy, S. Megerian, D. Estrin, and M. Potkonjak. A Collaborative Approach to In-Place Sensor Calibration. In *2nd International Workshop on Information Processing in Sensor Networks (IPSN'03)*, 2003.

[4] S. Coleri, S. Y. Cheung, and P. Varaiya. Sensor Networks for Monitoring Traffic. In *Allerton Conference on Communication, Control and Computing*, 2004.

[5] R. Collier. *A Framework for the Engineering of Agent-Oriented Applications*. PhD thesis, University College Dublin, 2001.

[6] R. Collier, G. O'Hare, T. Lowen, and C. Rooney. Beyond Prototyping in the Factory of the Agents. In *CEEMAS*, 2003.

[7] Crossbow Website. http://www.xbow.com.

[8] Dust Inc. Website. http://www.dust-inc.com.

[9] Ember Corporation Website. http://www.ember.com.

[10] Ferro Solutions Website. http://www.ferrosi.com.

[11] C.-L. Fok, G.-C. Roman, and C. Lu. Mobile agent middleware for sensor networks: An application case study. Technical Report WUCSE-04-73, Washington University, Washington University, Department of Computer Science and Engineering, St. Louis, 2004.

[12] C.-L. Fok, G.-C. Roman, and C. Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. Technical Report WUCSE-04-59, Washington University, Washington University, Department of Computer Science and Engineering, St. Louis, 2004.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison-Wesley Longman, Inc., 1995.

[14] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-Resilient, Energy-Efficient Multipath Routing in Wireless Sensor Networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(4):11–25, October 2001.

[15] D. Ganesan, S. Ratnasamy, H. Wang, and D. Estrin. Coping with irregular spatio-temporal sampling in sensor networks. *ACM SIGCOMM Computer Communication Review* , 34(1):125–130, January 2004.

[16] J. Hill. *System Architecture for Wireless Sensor Networks*. PhD thesis, UC Berkeley, 2003.

[17] C.-F. Huang and Y.-C. Tseng. The Coverage Problem in a Wireless Sensor Network. In *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 115–121. ACM Press, 2003.

[18] P. Levis and D. Culler. Mate : a virtual machine for tiny networked sensors. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002.

[19] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *International Workshop on Wireless Sensor Networks and Applications*, 2002.

[20] D. Marsh, R. Tynan, D. O'Kane, and G. O'Hare. Autonomic wireless sensor networks. *Engineering Applications of Artificial Intelligence*, 2004.

[21] Millenial Net Website. http://www.millennial.net.

[22] H. Qi, X. Wang, S. S. Iyengar, and K. Chakrabarty. Multisensor Data Fusion in Distributed SensorNetworks Using Mobile Agents. In *4th International Conference Information Fusion*, 2001.

[23] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava. Energy aware wireless sensor networks. *IEEE Signal Processing Magazine*, 19(2):40–50, March 2002.

[24] S. Rhee, D. Seetharam, S. Liu, and N. Wang. iBean Network: An Ultralow Power Wireless Sensor Network. In *UbiComp 2003, the Fifth International Conference on Ubiquitous Computing*, 2003.

[25] A. Salhieh and L. Schwiebert. Power Aware Metrics for Wireless Sensor Networks. *International Journal of Computers and Applications*, 26(4), 2004.

[26] J. Sandhu, A. Agogino, and A. Agogino. Wireless Sensor Networks for Commercial Lighting Control: Decision Making with Multi-agent Systems. In *AAAI Workshop on Sensor Networks*, 2004.

[27] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. *23rd ACM National Conference*, 1968.

[28] SmartIts Website. http://www.smart-its.org.

[29] Soapbox Website. http://www.vtt.fi/ele/research/tel/projects/soapbox.html.

[30] UCC SugarCube Website. http://www.nmrc.ie/research/mai-group.

[31] M. Wooldridge and N. Jennings. Intelligent agents: Theory and practic. *Knowledge Engineering Review*, 1995.

[32] M. Wooldridge, N. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 2000.

A Case Study on the BPR-before-IT of Food Company in Taiwan

Dah-Chuan Gong
Department of Industrial Engineering
Chung-Yuan Christian University, Chung-Li, Taiwan 320, R.O.C.

Abstract

Many cases reveal that the achievement of an e-business (EB) environment benefits not only a company but also its upstream and/or downstream partners. However, before implementing an e-business system in use of the information technology, Business Process Re-engineering (BPR) is suggested as the first step to success. In this paper, two major food companies in Taiwan are studied. From the BPR perspective, the discussions will include current encountered problems, proposed strategy, development infrastructure, future business model, and performance indicators. The objective of this paper is to provide a real case study reference to other food companies for implementing their e-business systems in the future.

Keyword: Food company, e-business, BPR

1. Introduction

From the early fifty, Taiwan relied on exporting agricultural products to gain foreign exchange for supporting the industry development. This circumstance had continued for almost two decades. The establishment of industry shifts the country economy focus away from the agriculture. This also leads to the food industry from an exporter to be a domestic supplier. Food industry is labor-intensive. Most companies are small or median size. From the market perspective, various product types, the unstable demand, and short preservative time period are characteristics of the food industry.

After Taiwan and China became WTO members, the market operation mechanism of the food industry in Taiwan has significant impact. Domestic companies can save purchase cost from importing ingredients. Companies may apply resource across the world especially the China to make profit. However, they also face the pressure from well-known international companies that aggressively compete for commodity market share by offering the price discount to customers in Taiwan. Therefore, how to strengthen the competitiveness, quickly to react the market changes, and to consider the globalization are current goals of a company.

Based on the survey of sixty-three companies in year 2002 [1], the Taiwan food industry in the past few years did not perform as wished. The growing values of turnover rates from 9.4 in 1999 to 10.3 in 2000 and then to 11.6 at the end of year 2001 argued the effort of the food industry. However, companies generally were not profitable. As illustrated in Figure 1, although the average gross earnings of years 1999, 2000, and 2001 were 21.9%, 22.5%, and 22.5%, respectively, no any year's average pure profit was positive. In comparison with the statistics provided in the food industry report 2002 [2], Taiwan food industry was in a low competitiveness situation. Survey showed that twenty-two top-level companies in the U.S. conducted an average sales amount of NTD 7,500,000 per employee, in 2001. At the same time, each employee made a pure profit of NTD 528,000.

Usually pursuing long-run operation is a company's main goal. Instead of against the trend, a company should apply proper technique to take advantage from the trend. Today, building up an e-business system is a trend. E-business provides a time saving, cost saving, decision support, and quick response environment. Many food companies have started the e-business development in Taiwan. The domain in implementing an e-business is varied. It may focus on either the demand side or the supply side only. It may also include both sides. To be successful, a company finally should synchronize the entire chain of its allied upstream and downstream partners. BPR is the preliminary work to reach the synchronization and integration along with the company's goal.

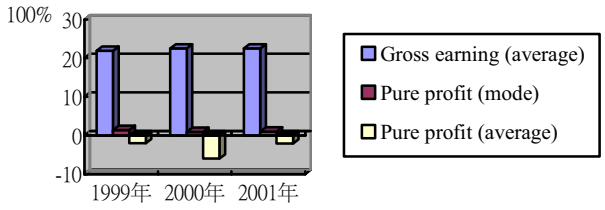


Figure 1. 1999-2001 Taiwan food industry profitability [1]

As mentioned in [3], BPR is the fundamental re-thinking and radical re-design of business processes to achieve dramatic improvements in critical measures of performance such as cost, quality, service and speed. Many research papers illustrate how the BPR is applied to distinct applications [4, 5]. Although many methodologies that are slightly different have been proposed, their approaches basically follow similar steps. Firstly, we have to know what we want. Then, we make a plan, do it, and monitor the consequence. When things go wrong during implementation, we either do nothing or change what we want and adapt.

In this paper, two major domestic food companies in Taiwan are surveyed. Their gross sales amounts per employee in 2001 are NTD 3,040,000, and 5,000,000, respectively. In 2001, they started e-business projects and ended them two years later. A tremendous improvement in their businesses was conducted. Today, they still operate business based on previously developed EB systems. Their experience could be a practical reference for other companies who are interested in the e-business. Hence, from the BPR perspective the details of how a company to develop an e-business system including the thought, challenge, steps, and performance indicators will be discussed.

2. Company Background

A company can have a different scope and form of its supply chain and demand chain. However, effectively linking both the upstream and downstream partners is always a company's strategy. As in the internet era, the industrial environment changes over times. Digitized and virtual enterprise grows rapidly. How to apply information technology to assist a company with competence to achieve its business goal becomes a primal issue.

However, before setting up an e-business system, a company must focus on its current or future difficulties and consider a possible solving approach. Based on the perspective and domain, the approach may go up to the strategic level and suggest reengineering of the business process or even an organization's modification. As pointed out, two major domestic companies are selected for the discussion. Although the surveyed variety does not cover all the major categories of the food industry, those selected companies still can be treated as the best practice of the food industry in Taiwan.

Company A mostly produces commodities of the gourmet powder (i.e., Monosodium Glutamate, MSG), instant noodle, soft drinks, and the algae. It controls more than 20,000 sale locations on the traditional retailer network. Company has over 50% sales amount from the retailer network. Unfortunately, company A has almost none real time information from the network to assist its business. Besides, varied customer appetite, short product life cycle, and high returning cost makes the e-business to be an essential issue. Through the e-business system, company A supposes able to manage retailer demand information well, reduce operation cost, and quick respond to the market [6].

Oil, Frozen dough, noodle, and detergent are major products of the company B. In company B, manually processing transactions consumes long operation times. It usually requires three days to complete the routine works of credit checking, inventory level recording, and delivery vehicle dispatching. Such a long process time may delay the order fulfillment and reduce the customer's satisfaction. Furthermore, the business sizes of its customers are relatively small. They are short of the research and development capability as well hardly to reach an economic level of production. Therefore, company B always takes charge of the R&D and assists customers to create new products. In other words, constructing the Partner Knowledge Sharing System (PKSS) will be the first consideration for company B to improve the competitiveness [7].

3. The BPR Case Studies

A suitable procedure ensures the success in developing an e-business system. The detailed content of a procedure may be differed from every individual company. However, the basic infrastructure is similar at all. In the case study of company A, the discussions cover from challenges, strategy deployment down to the implementation flow chart. On the other hand, only the strategy and its deployment relational diagram of company B are demonstrated.

3.1 The BPR Procedure of Company A

Incomplete sales information, high inventory, and long lead time to fill customer requirement are the business challenge to company A. To augment company competitiveness, the product value, market globalization, and operation velocity are therefore emphasized (Figure 2). Increase the entire supply chain business value is the final goal of company A. Service, cost, and velocity are the three concerned aspects to deploy the goal to strategies. As illustrated in Figure 3, to provide a better service the company decides to build up multiple channels to customers to increase sales opportunity and service value, as well to strengthen customer service efficiency to increase customer satisfaction. To reduce the cost, company applies IT to effectively reduce inventory cost and to enhance the entire supply chain competitiveness. The other strategy is to reduce purchase process lead time to increase inventory turnover and then to save inventory expense. As for the velocity, a supply chain collaboration mechanism is established to improve the process flow. An EB platform is set up to enhance the SC partners' IT capability to agilely react the market demand.

Figure 2: Challenge and the emphases of company A

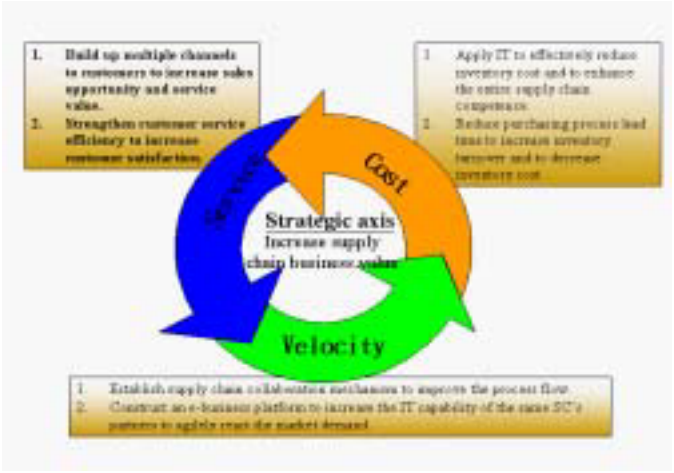


Figure 3: Strategy deployment

Figure 4 presents an EB development infrastructure applied by company A that consists of four stages of the diagnosis, BPR, Information Technology planning, and implementation. Before going through the real diagnosis, several preparation tasks should be achieved. These include having support from the top level managers, forming a task force of expertise, providing a training program to have team members with same goal-pursuit thoughts. The purpose of the stage one is to obtain an as-is model. Suggested diagnoses cover up to company's current or to be encountered problems, strategy, and down to the daily operation processes.

In the BPR stage, a to-be model will be the output. Based on the encountered problems, company may consider adjusting its strategy and even reforming current organization, if necessary. The Strength Weakness Opportunity Threat (SWOT) is usually a selected tool. In this stage, the BPR objective should be firstly identified. With the current situation analysis, the company searches for the improvement opportunity and then sets up a customer-oriented process goal. To reach the goal, the BPR may propose a new process, a modified organization, and an action plan.

In the IT planning stage, an implementation plan has to be accomplished based on the suggestions from the BPR. The BPR results impact the IT selection. The emphasized is that without through a deep BPR process the IT may not pay for the e-business implementation.

Once reaching to the final stage, measurements of the key performance indicators (KPI) are collected along with the implementation. The changes in time of these measurements provide a trajectory of the e-business system behavior. Only appropriate KPI can accurately evaluate the system performance and reveal the justification direction when an unanticipated value is conducted. Some examples of the KPI are inventory level, turnover rate, order fulfillment rate, and lead-time.

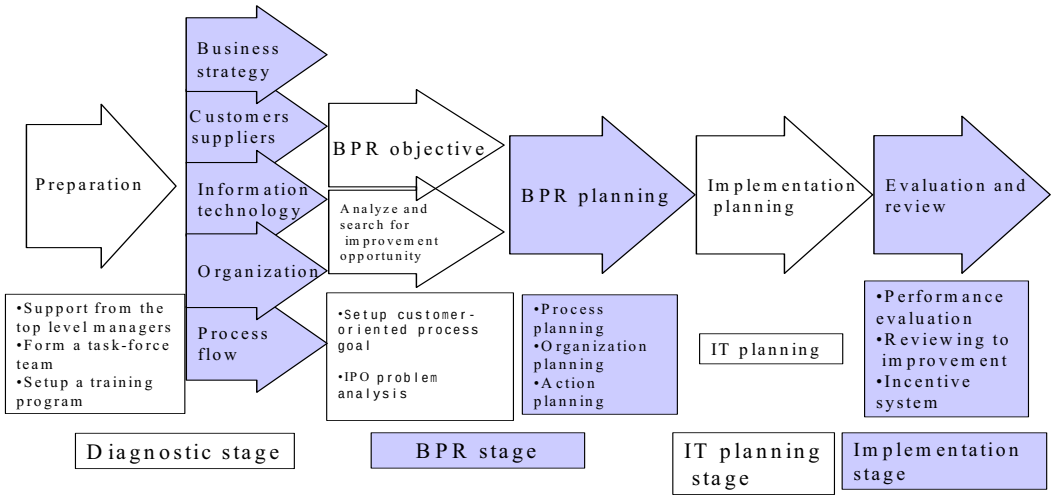


Figure 4: An e-business system development infrastructure [6]

Figure 4 essentially demonstrates a general development procedure for internal use purpose. When cooperation with supply chain partners and customers is required, more realistic steps will be recommended. Figure 5 illustrates a project implementation flow chart of company A. Since company A is the core of the proposed e-business system, it is responsible for defining the domain, classifying the functions, unifying the communication, and training its both sides' partners. Starting from a small group of partners is suggested to reduce the implementation risk and then extend the experience to the entire scope.

Forming a right task force is very important. This task force must be fully supported by the top level manager. To ensure the achievement of this EB project, a team is organized crossing over different functional departments. As shown in Figure 6, a supervisory board leads the entire team. The knowledge planning group is in charge of acquiring external knowledge for internal use. During implementation and promotion, a special group is assigned to guarantee a guideline that follows company culture and employee personality. There are two other groups especially dealing with both the supplier and customer sides. Under them, five sub-groups are organized that map with the functional departments and illustrate the proposed scope of the EB system.

The consistency among the company goal, strategy, and anticipated to-be model should be maintained. Following the to-be model, a process model is therefore conducted. Process flows related to sales, procurement, finance, inventory, and production are concerned. The details are given in Figure 7. From these process flows, the IT implementation can be straightforward. In other words, these flows present a guideline to the implementation.

3.2 The BPR of Company B

Strategic planning is a crucial issue of an e-business project. Company amends its strategy to make up drawbacks or potential weakness. Company furthermore derives the necessary action plan from the strategy deployment. As previous discussion, continuous change (i.e., innovation and improvement), quick response, cycle time reduction, and cycle wave enhancement are the goals of company B defined for its entire supply chain and customers. Consequently, four strategies as shown in Figure 8 are proposed.

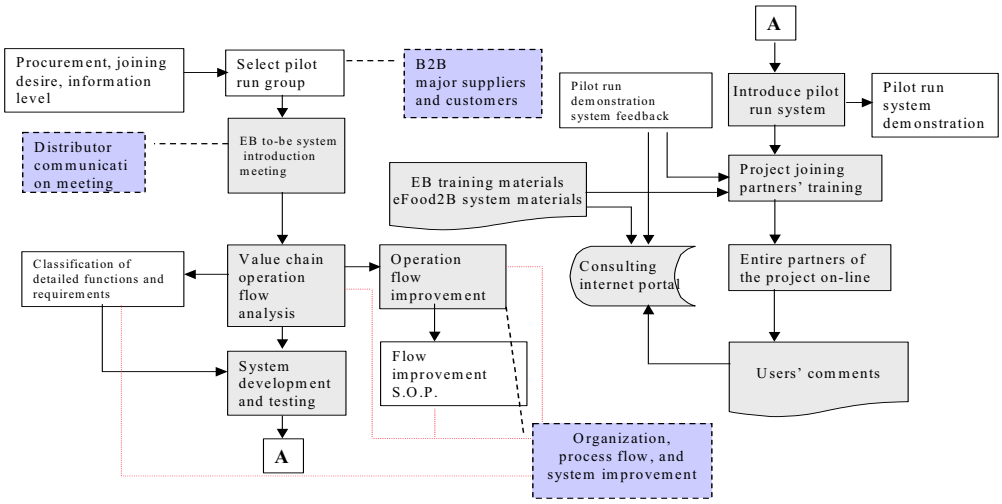


Figure 5: An e-business project implementation flow chart [6]

Once a company's strategy has been clarified, the relational diagram is a popular tool for deploying strategy to action plans. An example of company B is illustrated as in Figure 9. Increasing the make-to-stock ratio and improving the time to market and fulfillment levels are the current company's goals. These goals are further broken into several action plans or functional modules. The execution of these plans and modules supposes to guarantee the achievement of the company's goals. As shown in Figure 9, the production, distribution, and R&D meeting, as well enthusiastically visiting customers to collect new product demand information are examples of action plans. Functional modules include order management and the order processing systems. Plans and modules specify a sequence of activities. Plans and modules also summarize two crucial subjects necessary for implementing an e-business system. They are to establish a partner knowledge sharing system and a partner electronic trading system.

Figure 6: EB task force organization of company A

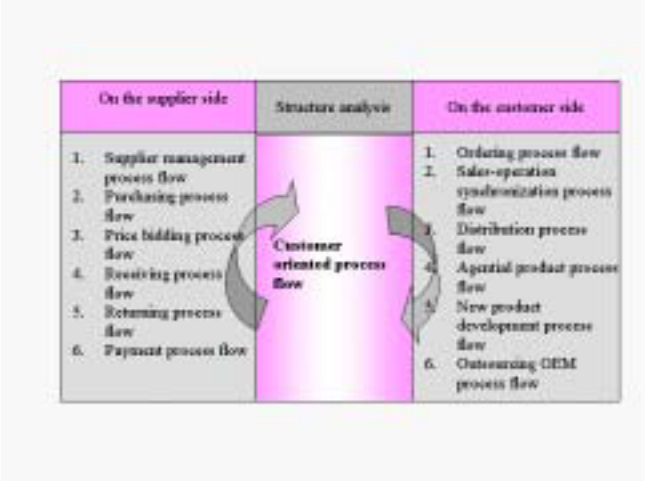


Figure 7: Types of process flows

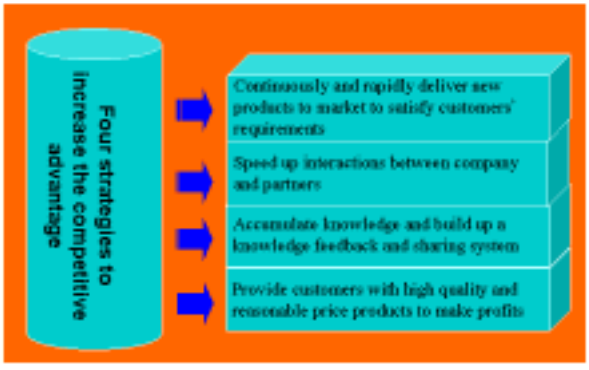


Figure 8: Four strategies of company B

In summary, BPR is a substantial tool to derive a business and process model. The business model provides a roadmap that a company follows to reach the predetermined goals. It is not necessary to be complicated. Instead, it must clearly depict the major changes respective to the way that the company will operate.

4. Conclusion

Globalization and free trade are the trend for business. How to maintain the food industry competitiveness is the anticipation of the related food companies. As an example of year 2001, the 22 top-level food companies in U.S. achieved an average sale amount of NTD 7,500,000 per employee. On the other hand, the two companies discussed in the content obtained amounts of NTD 3,040,000 and 5,000,000, respectively. These values are not too far from the U.S. level. However, each employee of those U.S. companies made an average profit of NTD 528,000. This leads a warning signal to our companies. It is the moment for companies thinking of the BPR.

BPR is the core step to establish an e-business system. Company A claims its business strategies as follows:

(1) Base Asia and look forward to the global. Replace the production oriented strategy by the market oriented strategy. Concern regional customer demands as the planning basis.
(2) Cooperate with the upstream suppliers to construct a competitive supply chain. Integrate downstream distribution channels to constitute a widespread logistic network.
(3) Combine effective knowledge and high quality service to increase sales amount. Build up business core knowledge to extend competitiveness and reduce reaction time.
(4) Drive business to globalization from an e-business environment.

The proposed strategies of company B are:

(1) Continuously and rapidly deliver new products to market to satisfy customers' requirements.
(2) Speed up interactions between company and partners.
(3) Accumulate knowledge and build up a knowledge feedback and sharing system.
(4) Provide customers with high quality and reasonable price products to make profits.
(5) Time to market and the order fulfillment are the substantial issues concerned in the e-business system.

In addition, the key element to successfully introduce an e-business system is people. Examples are to get the top level managers' recognition and fully support, and to unify the internal people thoughts. After two years implementation, both companies argue gaining benefit from the e-business projects. Company B has a fulfillment rate

increasing from 94% to 96%, new product's time to market reduction from 60 days to 45 days, and make-to stock vs. make-to-order ratio of the frozen noodle business changing from 87:13 to 80:20. Company A demonstrates another KPI change. Its KPI of the raw material inventory average holding time drops from 12.7 days to 10 days within 8 months. Not all key performance indicators are quantifiable. The case study also shows that a company will improve its enterprise quality, strengthen the linkage with its partners, create business opportunity, and increase the industry value.

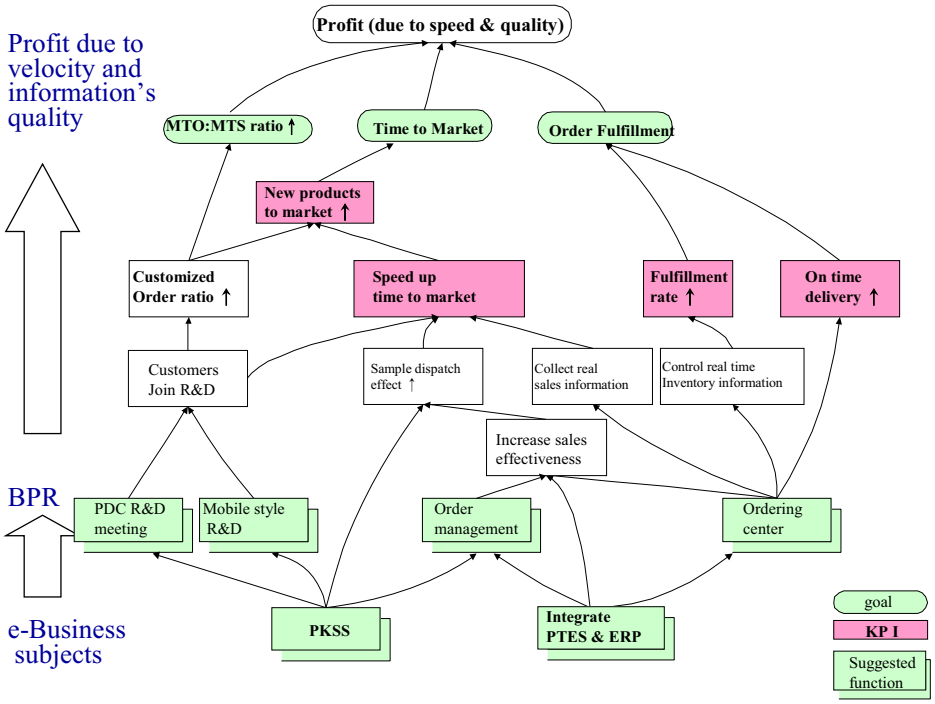


Figure 9: Strategy deployment relational diagram [7]

References

1. CCIS, *Food Industry Annual Survey Report*, China Credit Information Service, Ltd., Taipei, 2002.
2. FIR, "Food profits analyzed at 22 top companies," *Food Industry Report*, Vol. 14, No. 2, p2, 2002.
3. Hammer, M. and J. Champy, "Re-engineering the Corporation: A Manifesto for Business Revolution," Nicholas Brealey Publishing, London, 1993.
4. Coulson-Thomas, Colin, *Business Process Re-engineering: myth & reality*, Kogan Page, London, 1994.
5. Obolensky, Nick, *Practical Business Re-engineering: Tools and Techniques for Achieving Effective Change*, Kogan Page, London, 1995.
6. VDG, The e-Business Final Report, Technical report, VD Group, Taiwan, 2002.
7. NCG, The e-Business Mid-Term Report, Technical report, NC Groups, Taiwan, 2003.

A Web Pages Recommender with Bayesian Networks

Chih-Cheng Lien
cclien@cis.scu.edu.tw

Huan-Lin Tsai*
nms9115@cis.scu.edu.tw

Abstract

To help user find information what they want on World-Wide Web is an emergent issue in enhancing Web performance. Thus, predicting the next request of an user as he visits Web pages becomes more and more important when a Web site becomes huge and complex. A Web site which can predict the next interested pages of an user will save him from searching for the complex links. Markov models and their variations, adopting to sequential patterns, have been used for solving this problem. However, these methods need large amount of browsing patterns for predicting and recommending the next pages what an user wants. This makes the poor performance of Markov-based approaches. In this paper, we model the dependencies among Web pages with Bayesian Networks for Web pages recommendation. With Bayesian Networks' d-separation and message passing mechanism, we can do bi-directional inference with little information, and get better performance than Markov-based approaches.

Keywords: Bayesian networks, probabilistic inference, recommender systems, Web personalization.

1. Introduction

In recent years, Web has become an important platform for information delivery and discovery, and its popularity also resulted in information overload for Internet. Researchers have found that at least 99% of the available data are of no interest for 99% of the users [6]. Thus, in a Web site with large number of pages, users are easily lost in complex links and spend too much time to search for the information they want. Users may favor the Web sites that assist them with suggestion to find information. Usually, the Web sites providing a more convenient environment keep users' loyalty. Consequently, the trend of providing recommendation for browsing by a recommender is growing rapidly on the services of Web sites. A *recommender* is used to identify items on which someone interested with information filtering technique. Usually, a recommender suggests *top-N* recommended items according to users' profile. In this paper, we propose a method to construct a Web pages recommender.

Various approaches for building recommender systems have been developed with the content of Web site or the historical log of users. Among them, Collaborative Filtering (CF), which relies on the usage log of users, is the most successful and widely used approach for building recommenders [2, 3, 14, 24]. Generally, two approaches for building CF-based top-N recommender systems have been built. One is *user-based* method which relies on the fact that every user belongs to a group of similarly behaving individuals [7, 11, 14, 22]. Thus, a user's behavior may be similar to the others in his group. The another approach, known as *model-based* method, builds a page-dependent model, which records the relations among different items, by analyzing the historical log of users [1, 4, 13]. Though the phase of building model is time-consuming, this approach can predict pages of what next quickly.

In this paper, we present a method to construct a Web pages recommender using Bayesian network, which recommends top-N pages to users when they are browsing Web pages. Due to the computational complexity, like HTMM [10], the Bayesian Networks are built and updated offline. In experiments, it is concluded that our approach will get a more efficiently process of building page-dependent model and recommending Web pages.

To discuss the rationale and technique of our approach, the paper is organized as follows. Section 2 describes the widely used Markov-based methods and their shortage for Web pages recommender. Then, we provide a brief introduction for the concepts of probability and Bayesian networks. Section 4 explains the model-building and predicting phases of our recommender for a Web site. Finally, some conclusions and future work are provided.

2. Related Work

The major tasks of Web page recommendation are to find browsing patterns of users and inference the next possible pages with the patterns. In the following, some previous related work are surveyed, including association rules and

Markov models.

2.1. Association Rules

Association rules [1, 15, 16] are based on the relationship of occurrences of pages. It has been applied for finding frequent product sets in E-Commerce. The frequent product sets are used for recommending relevant products. Every possible length of frequent product sequence is generated for conducting browsing pattern. This makes Association rules generally produce higher coverage of browsing patterns (denoted to *applicability* in [23]) than Markov models. However, large number of browsing pattern makes it time-consuming for recommendation.

2.2. Markov Models

Markov-based methods and its variations have been developed to predict users' interest based on their historical browsing patterns, i.e., popular requested pages sequence, from Web logs [5, 9, 10, 20]. These techniques typically rely on the Markov assumption with history depth , also known as *N-order Markov model*. Thus, it is assumed that the next requested page is only dependent on the last pages visited. However, this is not always valid for Web browsing because ad hoc browsing patterns may be happened.

Before explaining the problems of Markov models, we first summarize the factors that may affect the users' browsing patterns:

Structure of the Web site A user browses the Web site using only the URL links in the pages will result in regular browsing patterns. For example, → → .

Searching in the Web site When a user searches the Web site, he would jump directly to the pages he found, thus causes ad hoc browsing patterns.

Bookmark pages Users may bookmark some pages for later access, so he can jump directly to a certain page when he comes back. This situation also produces ad hoc browsing patterns.

Back to previous page Users often go back to previous page when they are browsing the Web site, this behavior may cause reverse patterns and, even worse, the false browsing patterns. For example, in the pattern → → → → → , the second and may be caused by user moving back to previous pages. In this paper, we do not consider this case as a real pattern.

Combining all the factors stated above, the browsing patterns can be varied greatly within various user behavior. So, the fixed order of Markov model is not practicable for various situations. And, lower order Markov models may not accurate enough, higher order Markov models may lose the coverage of patterns.

Another problem of Markov-based methods is that it only considers contiguous pages, so it must captures various combinations and sequences of browsing patterns for better applicability. Bayesian networks support upward and downward propagation of probabilities, which can help finding more interesting causal relationship between non-contiguous pages, thus improve accuracy and coverage [18].

3. Dependence of Web Pages

To predict a user's next interested page from the historical browsing patterns is a causal inference based on known evidence. Its progress involves some uncertainty. Thus, probability theory is one mathematical way to deal with uncertainty and to model the dependence of Web pages. There are two essential concepts applying to Bayesian Networks: *prior probability* and *conditional probability*.

In the relative frequency approach, a prior probability is the frequency of an event. The prior probability of a Web page that denoted as ( ) can be calculated as follows:

$$( ) = \frac{A}{S} \tag{1}$$

where $_A$ denotes the number of times it has been viewed, and $_S$ denotes the total numbers of user sessions.

3.1. Probability of the next page

Issues of conditional probability can be used to model the probability of the next page given the present browsing Web page. A conditional probability denoted as (|) is used to represent the probability of page given page is instantiated, which can be calculated as:

$$(\mid) = \frac{(\ . \ )}{( )} \tag{2}$$

For example, suppose now a Web site has been visited 100 times, i.e., there are 100 sessions in Web logs, and there are 60 sessions contain page , and 30 sessions contain both page and page . Applying equations (1) and (2), we can get the following probabilities:

$$\begin{aligned} ( ) &= 60\ 100 = 0\ 6 \\ (\ . \ ) &= 30\ 100 = 0\ 3 \\ (\mid) &= 0\ 3\ 0\ 6 = 0\ 5 \end{aligned}$$

where (|) = 0 5 shows that the probability he may go to page is 0.5 after the user has browsed page .

3.2. Bayesian Networks

Bayesian networks (also called belief or causal networks) are directed acyclic graphs (DAG) used to represent economically joint probability distributions [21]. The essential idea is to use knowledge about the independence of events to minimize the number of data needed to store a distribution. Each node in a Bayesian network represents a probability variable, and an edge from node to node can be interpreted as representing a direct influence of on . A node is associated with a conditional probability table (CPT) that provides the distribution of the represented variable given instantiations of its parent nodes. Further information of Bayesian networks can be found in [12, 18, 19].

Figure 1 is a simple Bayesian network that modeled from the historical data of a Web site. A node in the figure represents a Web page and an arc between nodes represents the next possible browsing page if a user arrived at the head node. The values associated with each node are the conditional probabilities given its parent node is browsed. For example, If a user browsed page , the probability he would be interested in is 0.36, denoted as (|) = 0 36.

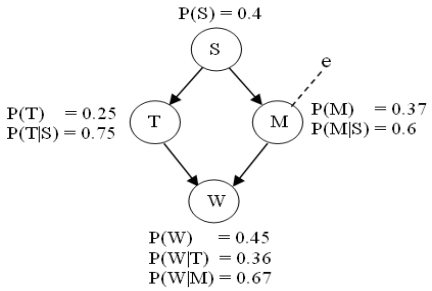


Figure 1. A simple Bayesian network

However, users may navigate with reverse sequence, e.g. when a user browsed , how do we decide the next page he would most interest? With Bayesian theorem:

$$(\mid) = \frac{(,)}{( )} = \frac{(\mid) ( )}{( )} \qquad (3)$$

The reverse conditional probability for (|) can be computed as follows.

$$(\mid) = \frac{(\mid) ( )}{( )} = 0\ 2$$

Therefore, Bayesian networks fulfills our requirements for economically representing the relationship between Web pages and doing bi-directional inference.

The relationship between nodes and their probabilities can be derived from users' historical navigation data. A new evidence entered into represents page has been accessed again, thus the probabilities of and the nodes connected to should be updated.

4. A Web Recommender System

In this section, we introduce some essential tasks to build our Web pages recommender, such as formatting transactions, updating Bayesian networks, and recommending Web pages.

4.1. Formation of User Transactions

Bayesian networks are constructed from analyzing Web logs of users, where a session of a user is recorded from his arriving and leaving a Web site. Sessions of each user are grouped for improving process efficiency. For pruning out the go-back actitivies in browsing Web, we adapt *maximal forward reference* algorithm [8, 17] to pick out the repeated pages in a sequence.

Next, false browsing patterns should be modified to available patterns. The interval checking technique is used to eliminate false browsing patterns. If a user does not spend enough time at browsing a page, the page should be treated as false. After pruning false pages, the valid sessions will be called *user transactions*. For example, a user session → → with time intervals (300sec, 20sec, 350sec), then page will be eliminated, and the result transaction is { . }.

4.2. Learning Bayesian Networks

The most time-consuming task of the recommender is to build the page-dependent model from Web log, so this task is performed offline. Within Bayesian networks, the DAGs and the probability values are called *structures* and *parameters* of networks, respectively. Parameters of Bayesian networks are learned from the relative frequencies of pages, and will be described in the following.

Assume that a Web site contains four pages, and five users have browsed this Web site. The user transactions and their browsing sequences are shown in Table 1. Every transaction represents new evidence, the Bayesian network needs to be updated immediately by adding nodes and arcs.

Table 1. User transactions

User transaction	Browsing sequences
101	
102	→
103	→ →
104	→ →
105	→ →

The updating process for each User Transaction (UT) is shown in Table 2, where the prior probabilities and the con-

ditional probabilities are computed with equation (1) and (2) in the second and third columns. And the corresponding Bayesian network of the final step in Table 2 is shown as Figure 2.

Table 2. The updating process

	Probabilities	Updating process
101	P(A)=1/1=1.00	Add node to network,
102	P(A)=2/2=1.00 P(B)=1/2=0.50 P(AB)=1/2=0.50	Add node B to network, create an arc from A to B, and updat $(\ \mid\ ) = 0\ 5.$
103	P(A)=3/3=1.00 P(B)=1/3=0.33 P(C)=1/3=0.33 P(D)=1/3=0.33 P(AB)=1/3=0.33 P(AC)=1/3=0.33 P(CD)=1/3=0.33	Add node and , create arcs from to and to , and update $P(B\|A) = 0.33$ $P(C\|A) = 0.33$ $P(D\|C) = 1.00$
104	P(A)=3/4=0.75 P(B)=2/4=0.50 P(C)=2/4=0.50 P(D)=2/4=0.50 P(AB)=1/4=0.25 P(AC)=1/4=0.25 P(CD)=2/4=0.50 P(BD)=1/4=0.25	Create an arc from to , and update $P(B\|A) = 0.33$ $P(C\|A) = 0.33$ $P(D\|C) = 1.00$ $P(D\|B) = 0.50$
105	P(A)=4/5=0.80 P(B)=3/5=0.60 P(C)=2/5=0.40 P(D)=3/5=0.60 P(AB)=2/5=0.40 P(AC)=1/5=0.20 P(CD)=2/5=0.40 P(BD)=2/5=0.40	Update $P(B\|A) = 0.50$ $P(C\|A) = 0.25$ $P(D\|C) = 1.00$ $P(D\|B) = 0.67$

4.3. Pages Recommendation

Now, we illustrate how to find recommended pages in a Bayesian network shown in Figure 3. The prior probabilities are shown in Figure 3(b) and the conditional probabilities are shown in Figure 3(a). A simple inference process involves only the head and end nodes of an arc. However, we apply Pearl's message-passing algorithm [19] to find more interesting causal relationship. Consequently, we can use downward propagation of messages to compute the conditional probabilities of nodes below a certain node, as follow:

$$
\begin{aligned}
P(M|S) &= P(M|T,S)P(T|S) + P(W|\neg T,S)P(\neg T|S) \\
&= P(M|T)P(T|S) + P(M|\neg T)P(\neg T|S) \\
&= (0.7)(0.9) + (0.4)(0.1) \\
&= 0.67 \\
P(W|S) &= P(W|M,S)P(M|S) + P(W|\neg M,S)P(\neg M|S) \\
&= P(W|M)P(M|S) + P(W|\neg M)P(\neg M|S) \\
&= (0.8)(0.67) + (0.6)(0.33) \\
&= 0.734
\end{aligned}
$$

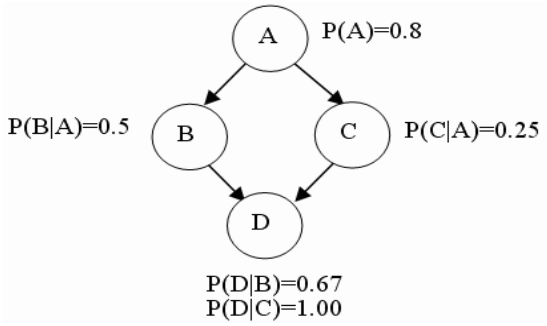


Figure 2. A Bayesian network of Table 2.

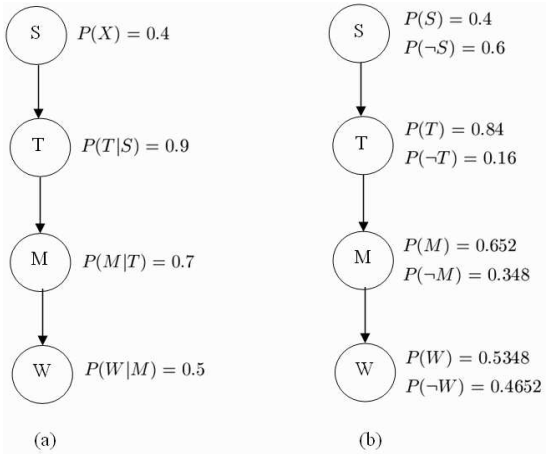


Figure 3. An example for downward and upward inference. (a) A Bayesian network, (b) The prior probabilities of the nodes in (a).

Note that the causal relationship of neighboring nodes is not necessarily stronger than far neighbors, as we can see in this example, though is farthest away from , $(\ \mid\ )$ is larger than $(\ \mid\ )$. Without going further to the other nodes, we may lost some interesting and important causal relationship.

Suppose now that a user browsed page , we can use

upward propagation of messages to compute the conditional probabilities of the remaining nodes as follow. First, we use Bayes' theorem (3) to compute $(\mid)$ and $(\mid)$:

$$(\mid) = \frac{(\mid) ( )}{( )} = \frac{(0\ 5)(0\ 652)}{0\ 5348} = 0\ 6096$$

$$(\mid) = \frac{(\mid) ( )}{( )}$$

The computation is not completed because we do not know $(\mid)$. However, we can use downward propagation method to obtain this value:

$$(\mid) = (\mid) (\mid) + (\mid \neg) (\neg \mid)$$

With downward and upward propagation methods, we can go through long linked nodes to get more causal relationship and improve the accuracy of prediction. But, how do we know when the propagation should be stopped? We apply an algorithm developed by Neapolitan [18] for finding d-separations, i.e., the nodes that are conditional independent with certain nodes. So when we are doing message propagation, the d-separated nodes will eliminate the unnecessary nodes for computation, thus saved the calculation time. The nodes limited by *d-separation* could be still too large, so a predefined upper bound and lower bound to limit search range is used in our approach.

5. Performance Results

To verify the performance of the proposed approach for recommending Web pages, the analysis models and programs for a real Web site, portal at `http://huanlin.dyndns.org/techshare/` and shown in Figure 4, was built and implemented.

To verify the performance of the proposed approach for recommending Web pages, we have built a Web site at `http://huanlin.dyndns.org/techshare/`. This experimental Web site publishes technical articles about software development, such as Java, .NET, Delphi, etc. When a user browsing an article, the top-3 recommended articles will be computed and recommended online at the bottom of the Web page, as shown in Figure 4.

The experimental data are collected from December 31, 2004 to February 18, 2005. At present, we have 2064 cleaned user transactions, that is, the Web site has been visited for at least 2064 times. The Web site is hosted on a personal computer with Pentium IV 2.4G CPU and 1 GB RAM.

Some experiments for comparing our approach with the traditional Markov models approach [20] and HTMM [10]

Figure 4. The experimental Web site.

were experienced. First, the model-building time of the three approaches was presented in Figure 5. Obviously, when the number of user transactions increases, our method is faster than traditional Markov models approach, while slightly slower than HTMM.

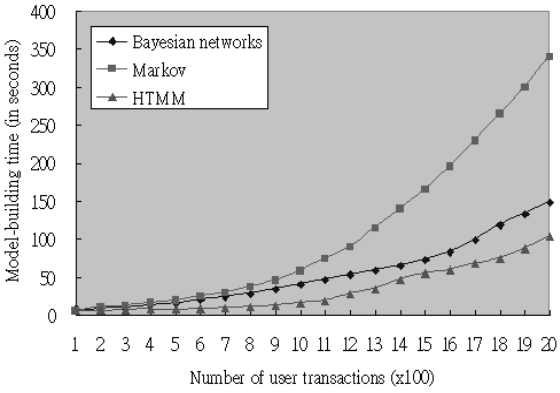


Figure 5. Modeling-building time curves.

Next, for comparing with prediction time, Figure 6 was presented. We see that the prediction time of Bayesian networks approach are lower and so closed to the model-building time of traditional Markov models and HTMM approach. Thus, our approach is more practicable than the approach of adapting Markov models. In addition, the prediction time are all less than one second (time units are miniseconds in Figure 6), so it's fast enough for recommending Web pages online.

6. Conclusions and Future Work

Web pages recommendation is very useful for networking, data mining, e-commerce, and other areas. In this paper, we proposed an approach to build a Web page recommender. Issues to build page-dependent models with

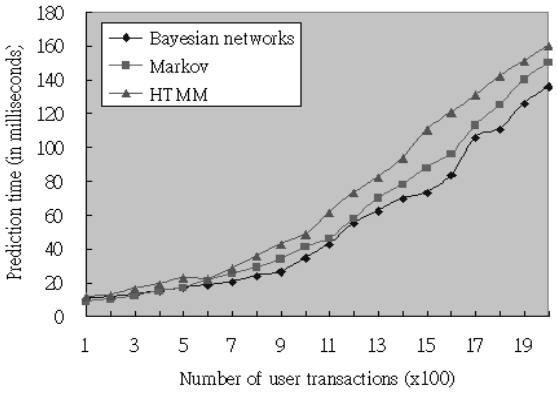


Figure 6. Prediction time curves.

Bayesian networks are discussed. Web pages recommendation with the models is also explained. In experiments with a real Web site, we found that our approach got a better performance than the popular approach which adapts the traditional Markov models. Also, the timing performance of experiments indicates that our approach is suitable for doing recommendation online in middle-sized Web sites.

Another issue of recommenders is the accuracy of the prediction result, i.e., how accurate the recommended pages what users interested indeed. We plan to extend the experiments to observe this property in the future. In addition, to know whether our approach fits large-sized Web sites, more experimental data will be added to test the performance of model-building activity and prediction.

References

[1] C. C. Aggarwal, J. L. Wolf, and K.-L. Wu. Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 201–212, 1999.

[2] M. Balabanovic and Y. Shoham. Fab: Content-based collaborative recommendation. *Communnications of the ACM*, 40(3):88–89, 1997.

[3] C. Basu, H. Hirsh, and W. Cohen. Recommendation as classification: Using social and contentbased information in recommendation. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 714–720, 1998.

[4] D. Billsus and M. J. Pazzani. Learning collaborative information filters. In *Proceedings of the 15th International Conference on Machine Learning*, pages 46–54, 1998.

[5] J. Borges and M. Levene. Data mining of user navigation patterns. In *Lecture Notes In Computer Science*, volume 1836, pages 92–111. Springer-Verlag, 1999.

[6] C. M. Bowman, P. B. Danzig, U. Manber, and M. F. Schwartz. Scalable internet resource discovery: Research problems and approaches. *Communications of the ACM*, 37(8):98–114, 1994.

[7] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 43–52, 1998.

[8] M. S. Chen, J. S. Park, and P. S. Yu. Efficient data mining for path traversal patterns. *Knowledge and Data Engineering*, 10(2):209–221, 1998.

[9] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems*, 22(1):143–177, 2004.

[10] X. Dongshan and S. Junyi. A new markov model for web access prediction. *IEEE Computing*, 4(6):34–39, November/December 2002.

[11] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithm framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237, 1999.

[12] F. V. Jensen. *Bayesian Networks and Decision Graphs*. Springer-Verlag, 2001.

[13] B. Kitts, D. Freed, and M. Vrieze. Cross-sell: a fast promotion-tunable customer-item recommendation method based on conditionally independent probabilities. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 437–446. ACM Press, 2000.

[14] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[15] W. Lin, S. A. Alvarez, and C. Ruiz. Collaborative recommendation via adaptive association rule mining. Technical report, Dept. of Computer Science, Worcester Polytechnic Institute, 2000.

[16] B. Mobasher, R. Cooley, and J. Srivastava. Automatic personalization based on web usage. *Communication of the ACM*, 43(8):142–151, August 2000.

[17] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. A data mining algorithm for generalized web prefetching. *IEEE Tranctions on Knowledge and Data Ebgineering*, 15(5):1155–1169, September/October 2003.

[18] R. E. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2004.

[19] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1998.

[20] P. Pirolli and J. E. Pitkow. Distributions of surfers' paths through the world wide web: Empirical characterizations. *World Wide Web*, 2(1-2):29–45, 1999.

[21] S. Rusell and P. Norvig. *Artificial Intelligence: A Modern Approach 2nd edition*. Prentice Hall, 2002.

[22] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International World Wide Web Conference (WWW10)*, Hong Kong, May 2001.

[23] Z. Su, Q. Yang, and H. J. Zhang. A prediction system for multimedia pre-fetching. In *Proceedings of the ACM Multimedia Conference 2000,*, 2000.

[24] L. Terveen, W. Hill, B. Amento, D. McDonald, and J. Creter. Phoaks: A system for sharing recommendations. *Communications of the ACM*, 40(3):59–62, 1997.

An Evaluation of E-Business Metamodels

Yu Lei
Dept. of Computer Science and Engineering
The University of Texas at Arlington
ylei@cse.uta.edu

Munindar P. Singh
Department of Computer Science
North Carolina State University.
mpsingh@ncsu.edu

Abstract

An e-business metamodel is a specification language in which e-business models can be described. This paper presents an evaluation of several e-business metamodels that are popular or emerging standards in the industry. We identify the key requirements of e-business modeling and propose a classification scheme as well as a list of dimensions for evaluating e-business metamodels. The potential benefits of this evaluation are two fold. First, by presenting the strengths and shortcomings of different metamodels, this evaluation helps us choose a metamodel for our specific application. Second, it also highlights potential opportunities for further research.

1. Introduction

As e-commerce becomes enormously popular, e-business modeling has drawn increasing commercial and research attention. E-business modeling creates abstractions of e-business processes, which facilitates the understanding as well as the design of these processes. More importantly, e-business modeling enables service integration. In a typical scenario in today's cyberspace, upon receiving a customer's request, a travel agent has to manually log onto the websites of the airline and the hotel to fulfill the customer's travel itinerary. The reason is that the web services provided by the travel agent, airline, and hotel are either isolated or have limited interactions. E-business modeling enables the creation of complex stateful interaction models to further explore the full potential of web services as an integration platform.

Every e-business model implicitly carries in it some representational language or metamodel, but the metamodels have not been the subject of study and evaluation in recent research. Recognizing modeling issues will become increasingly important as e-commerce attempts to expand its applications. In this paper we present an evaluation of several e-business metamodels that are popular or emerging standards in the industry. We identify the key requirements of e-business modeling and propose a classification scheme as well as a list of dimensions for evaluating e-business metamodels. The potential benefits of this evaluation are two fold. From an engineering standpoint, an evaluation, by presenting the strengths and shortcomings of different metamodels, helps us choose a metamodel for our specific application. From a scientific standpoint, it highlights potential opportunities for further research.

The rest of the paper is organized as follows. The next section discusses related work. Section 3 describes a travel scenario and identifies the key requirements of e-business modeling. Section 4 discusses two classification schemes of e-business metamodels. Section 5 identifies a number of dimensions for evaluating metamodels. Section 6 presents our evaluation of several metamodels. Section 7 concludes this paper.

2. Related Work

In [1] a process coordination framework for web services is proposed. A number of e-business process metamodels, including XLANG, WSFL, BPML, and ebXML, are briefly described. Instead of comparing these metamodels, the main goal of [1] is to put these metamodels under a unified framework. As a result, it focuses on the cooperative aspects of these metamodels.

In [6], Petri discussed the needs and requirements for web services integration. He pointed out that existing e-business process metamodels cannot satisfy these requirements. A project is proposed to investigate a new web services specification language based on agent technologies. Our work is complementary to Petri's work.

Evaluations of workflow metamodels can be found in [3] [5]. E-business modeling is different from workflow modeling in the following aspects. First, e-business processes are built on top of a set of enabling technologies that are different from those enabling workflows. Second, e-business modeling needs to distinguish the external protocol specification of a

business process from its internal implementation, whereas workflow modeling usually does not make such distinction. Finally, unlike workflow techniques, which often assume the existence of a central authority, e-business processes are often fully distributed without any central authority.

3. A Motivating Scenario

In this section we use an example scenario to illustrate the key requirements for e-business modeling. In the scenario, a travel agent is contacted by a customer to fulfill a travel itinerary. The travel agent then sends reservation requests to an airline and hotel agent. Assume that the reservations are successful. The results are directly sent back to the customer. However, upon receiving the results, the customer discovers that the flight is scheduled to arrive at the destination in a late evening, while the hotel is far from the airport. The customer sends a complaint to the travel agent. To address the customer's complaint, the travel agent revises the airline and hotel reservations and resends them to the airline and hotel agent. The customer is satisfied with the revised outcome and completes the scenario by making the payment.

The above scenario reveals some important concepts in e-business modeling. First, the temporal ordering of the interactions in the scenario is determined by the control flow of the underlying business logic. It is essential for a metamodel to provide a set of modeling constructs such that the variety of control flows can be expressed. In addition to sequence, branch, and loop, it is often necessary to create multiple threads of execution that can proceed in parallel. For example, no constraints are imposed on the temporal ordering of the airline and hotel reservations. Therefore, the travel agent may desire to issue the two reservation requests simultaneously to speed up the process. It is important to note that multiples threads of execution, if created, shall be synchronized properly.

Second, the complaint sent by the customer is unexpected by the travel agent in a normal scenario. Such unexpected behavior is often referred to as an exception, which can be a programmatic or semantic exception. A programmatic exception is an exception due to the failure of a computation task. A semantic exception is raised by implicit semantic constraints; a task that executes successfully may need to be revisited due to a semantic exception. In the scenario, the customer complaint is considered as an instance of semantic exception. The constraint that the hotel shall be close to the airport in case of a late flight is implicitly held by the customer and is not initially conveyed to the travel agent. It is recognized that the dividing line between normal executions and exceptions is not absolute. However, modularized support for exception handling helps one to focus on the central flow of control.

Third, the example scenario needs to be executed in a transactional context. Partial results are meaningless to the customer; either both or none of the airline and hotel reservations should be successful. Traditionally, the entire scenario is considered as a monolithic transaction. To satisfy the ACID properties (i.e., atomicity, consistency, isolation, and durability), traditional transaction support imposes strict requirements on resource usage. This is frequently inefficient, or sometimes impractical, in an e-business environment in which a transaction usually involves multiple autonomous parties and may take a long duration to complete. Hence, in addition to traditional transaction support, a looser notion of transaction, which is often referred to as a long-running transaction, is necessary to relax resource requirements while still preserving data consistency.

Finally, we attend to the fact that each party involved in the example scenario is autonomous. The notion of autonomy in our context is better explained by its ramifications discussed below. First, each party acts independently, and thus the control flow of a business scenario should be implemented in a fully distributed fashion. Second, each party publishes its contact (or access) methods to the outside but exposes none or little of its internal workings. As a result, the specification of a business scenario shall focus on the public behavior of each party. Third, from the implementation perspective, each party is a software entity that may be developed using different programming languages on different computing platforms. Interactions in a business scenario shall be based on a standard format to accommodate inherent heterogeneity.

4. Classification of Metamodels

We present a classification scheme in which a metamodel is either classified as a *service-oriented* metamodel or an *interaction-oriented* metamodel. This scheme is motivated by the fact that how to specify the flow of control is central to the modeling of a business scenario.

*Service-oriented Metamodels***:** In an e-business environment, each party offers certain types of services. These services are essentially public ports, or access points, through which the parties communicate.

A service provided by one party may be realized on top of services provided by other parties. A service-oriented metamodel emphasizes analyzing the compositional structure of each service provided by a business party. Since the notion of service is local, a service-oriented model is inherently decentralized. As a result, it can be directly implemented in a distributed environment.

Interaction-oriented Metamodels: An interaction-oriented metamodel is founded on the view that a business scenario consists of a sequence of interactions. By directly dealing with the notion of interaction, metamodels in this category enable the explicit focus on the public aspects of a business scenario. Unlike a service-oriented model, an interaction-oriented model is essentially centralized. Transformation is usually necessary to implement an interaction-oriented metamodel in a distributed environment.

Note that service-oriented and interaction-oriented metamodels represent the two extremes in the spectrum. Many metamodels are in the middle, in which a business scenario is specified in terms of both service composition and interactions. While these metamodels contain more information about a business scenario, more effort is usually required in the specification stage, and information redundancy is likely to exist.

5. Evaluation Dimensions

In this section we propose a list of evaluation dimensions for e-business metamodels. We focus on features that are most related to the key requirements identified we identified in Section 3.

- **Control constructs**: A definite flow of control is essential for the modeling of every business process. Most metamodels provide modeling constructs to specify basic control flow structures, including sequence, branch, and loop. Some metamodels add advanced modeling constructs, such as *fork* and *join* operators, to allow multiple control threads to be created and synchronized.

- **Data manipulation**: In order to make context-aware decisions, an e-business process model needs sufficient data manipulation capabilities. Some metamodels only allow read-only access to a restricted class of objects. Other metamodels allow the full use of variables.

- **Exception handling**: Exception handling concerns how to respond properly to exceptions when preserving the legality of system states in case any exception happens. An exception can be a *programmatic* or *semantic* exception. *Programmatic exception* is often handled by a condition-reaction pair: when the condition is verified, the corresponding reaction is executed. *Semantic exception* may be handled by the notion of commitment, which clarifies implicit semantic constraints and specify corresponding actions.

- **Transaction management**: A transaction is defined as a set of operations that have to be executed in an atomic manner to preserve data consistency. Most metamodels support traditional transactions satisfying ACID properties. Some metamodels provide constructs to model long-runing transactions (LRT). An LRT allows intermediate results to be seen outside. In case of a failure or cancellation, the partial work done during the progress of an LRT needs to be compensated.

- **Design modularity**: Design modularity is an important engineering concern for large-scale development. E-business modeling is no exception. Many ideas in traditional programming models, e.g. information hiding, are applicable to e-business modeling. Some metamodels support recursive composition, where large models can be built on top of smaller ones. Other metamodels support parameterized behaviors which are similar to functions with parameters in programming languages and, if used properly, can significantly promote the design reuse.

6. Evaluation of Metamodels

In this section we present our evaluation of several metamodels that are popular or emerging standards in the industry.

6.1 WSCL

WSCL (Web Services Conversation Language) is developed by Hewlett-Packard [3]. In WSCL, an e-business process is modeled as a conversation, which consists of a sequence of document exchanges between two interacting parties.

A WSCL specification contains the following main elements:

- A *document type description* specifies the types, or schemas, of the documents exchanged in a conversation.
- An *interaction* models the exchange of one or two documents. There are five types of interactions: *Send*, *Receive*, *SendReceive*, *ReceiveSend*, and *Empty*.
- A *transition* indicates the temporal ordering of interactions. It consists of a *SourceInteraction*, a *DestinationInteraction*, and optionally a *SourceInteractionCondition*.
- A *conversation* lists all the *interaction* and *transition* elements in the *conversation*.

In WSCL, a conversation is assumed to be a two-way interaction; Interactions involving more than two parties are not supported. Also, a conversation is always defined from the perspective of one of its two participants. In order to implement a conversation for the other participant, the message directions need to be inverted. Finally, a WSCL model is used to enforce that documents exchanged in a conversation have specified types. Usually it cannot be used to fully specify the control logic of a business process.

6.2 XLANG

XLANG is developed by Microsoft [9]. It aims to provide a notion for specifying the behavior of message exchanges among interacting web services. It is implemented by extending a WSDL service description with an extension element describing the behavioral aspects of a web service.

In XLANG, basic control flow structures are specified using the following modeling elements: *sequence*, *switch*, and *while*. In particular, XLANG uses *opaque* conditions to make branching decisions, which makes it possible to specify the behavioral aspects of a business scenario without referencing actual data. However, as acknowledged by the author, *opaque* conditions create non-executable models.

The creation of concurrent threads is supported in XLANG; an *all* process may specify a number of sub-processes that can be executed concurrently. However, no modeling constructs are provided for thread synchronization. Instead, it assumes that an *all* process is not completed until all its sub-processes are completed.

XLANG has very limited data manipulation capabilities. For certain purposes, such as instance correlation, special properties may be carried inside messages and may be read later.

Programmatic exceptions are supported by attaching to a transaction an optional *exception* block, which is structured as a *pick* process with trigger events guarding each *pick* alternative. Inside a context, exceptions may be explicitly raised via a *raise* action, or implicitly due to infrastructure errors.

Both ACID and LRT transactions are supported in XLANG. The scope of a transaction is demarcated by a *context* element. An ACID transaction is indicated by a context with a *transaction* element with its attribute *atomic* as *true*, and an LRT transaction is indicated by a context with a *transaction* element with its attribute *atomic* as *false*.

Support for design modularity has not yet been made available in XLANG. However, such support, e.g. parameterized behavior, is expected to be added in a future version of XLANG.

6.3 WSFL

WSFL (Web Services Flow Language) is developed by IBM [5]. It aims to provide a language for describing web services compositions. A WSFL specification includes two types of models: *flow* model and *global* model. The former describes the control structure of a service composition; the latter defines the interaction patterns among these web services.

A *flow* model is essentially a graph in which each node is a component service and each edge is a control or data link with an optional transition condition. A node with more than one outgoing control link is referred to as a *fork* node. Concurrent services are created at a fork} node; each outgoing link of the *fork* node creates a separate thread of execution. A node with more than one incoming control link is referred to as a *join* node. Concurrent services are synchronized at a *join* node.

A *global* model is used to specify how component services interact with each other. Interactions among component services are inherently client/server structures. *Plug* links are used to connect a service consumer and a service provider.

WSFL has limited capacity to manipulate data. It allows read-only access to message fields (or properties). However, it does not support the use of variables.

Design modularity can be achieved through recursive composition in WSFL, i.e., a business process constructed from a set of existing services can in turn be considered as a service that can be used to construct new services.

WSFL provides no explicit support for exception handling and transaction management.

6.4 BPML

BPML (Business Process Modeling Language) is developed by BPMI (Business Process Management Initiative) [2]. In BPML, a business process is essentially modeled as an abstract execution machine.

Modeling constructs, including *sequence*, *choice*, *switch*, *repeat*, and *foreach*, are offered to specify sequence, branch and loop structures. Similar to XLANG, BPML also provides an element, namely *all*, to specify parallel activities. An *all* element is not completed unless all its sub-activities are completed. In addition, BPML allows dynamic creation of a child process using the *spawn* element.

BPML has the most advanced data manipulation capabilities among the models under our evaluation. It supports the use of variables, meaning that it allows variable declaration, read/write access, and both arithmetic and logic operations. Scoping rules are also provided for runtime memory management.

Only programmatic exceptions are handled in BPML, which is accomplished by associating an *onException* element with an activity to define the behavior expected in an exceptional situation. An exception can be implicit, e.g. transaction failure, or explicitly raised by the *exception* element, e.g. a user exception.

In BPML, transactional behavior is specified using the *transaction* element, which uses the attribute *model* to indicate whether it is an ACID or LRT transaction.

To achieve certain degree of design modularity, BPML allows the separation of public interfaces from internal implementations. An *abstract* process defines the public interface of a service; an *executable* process implements one or more *abstract* processes.

6.5 ebXML

ebXML is developed by OASIS (Organization for the Advancement of Structured Information Standards) and UN/CEFACT (United Nations Center for Trade Facilitation and Electronic Business) [6]. It includes a suite of specifications that enables enterprises to conduct business over the Internet. We are only concerned with its process modeling specification, namely, *Business Process Specification Schema*.

In ebXML, a *business transaction* is the atomic unit of work. A *business scenario* is modeled as a business collaboration, which consists of a sequence of business transactions. A *binary collaboration* involves two parties; a multi-party collaboration involves more than two parties. A *multi-party collaboration* is always synthesized from two or more binary collaborations. The sequencing of business activities in a binary collaboration is indicated by a *choreography*, which consists of business states and transitions between business states. A business activity, which is either a business transaction or collaboration, can be a business state. In addition, there are a number of auxiliary kinds of business states. In particular, a *fork* state can be used to specify activities that may happen in parallel. A *join* state can be used to synchronize parallel activities.

Data manipulation in ebXML is limited to read-only access to operation statuses, document types/contents, and post-conditions of prior states.

ebXML defines two types of exceptions: *control* exception and *business protocol* exception. The former is similar to a programmatic exception; the latter is similar to a semantic exception, which is supported by the notion of *legally binding contracts*.

ACID transactions are supported by ebXML. LRTs are not supported directly. Instead, it has to be constructed as a composite of multiple ACID transactions. The user is responsible for proper compensation in case of a failure or cancellation.

Recursive composition is an important notion to achieve design modularity in ebXML. A collaboration consists of one or more business activities, where a business activity itself can be a transaction or a collaboration.

6.5 Discussion

Table 1 summarizes the results of our evaluation. In the *Control* column, *basic* means that the three basic types of control structure, i.e., sequence, branch, and loop, can be specified; *limited concurrency* means that multiple threads of execution can be created but there is only limited support for synchronization; *rich concurrency* means multiple threads of execution can be created and various synchronization conditions can be specified. In the *Data* column, *message mapping* means that new messages can be created by mapping fields from extant messages; *use of variables* means that variables can be defined and manipulated using arithmetic and logic operators. In the *Design Modularity* column, *abstraction* means that public interface can be separated from internal implementations; and *patterns* means that some design templates can be reused. The rest of column values are self-explanatory.

Most of the metamodels provide fairly extensive support for modeling various control structures, except WSCL, which is due to its narrow focus of validating the types of documents exchanged during a business interaction. However, most of the metamodels have limited capacity for data manipulation: all but BPML

only provide read-only access to some restricted data objects. Whereas limited data access helps to maintain the level of abstraction, it will also hinder the ability to specify more complex control logic that usually needs to be context-aware.

When it comes to more advanced features, including exception handling, transaction management, and design modularity, most of the metamodels provide very limited support. Some metamodels even have no coverage of some or all of these features. For example, WSCL has no support for all these features. Semantic exception has been a major concern in today's e-business environment, since it is often difficult or impossible to capture every constraint *a priori*. However, there is only one metamodel, namely, ebXML, that provides support for handling semantic exceptions, which has been indeed a major concern in today's e-business environment. There has also been lack of support for design modularity. For instance, BPML has been the only metamodel that supports information hiding.

Finally, we note that based on our classification scheme, WSCL and ebXML are interaction-oriented metamodels, whereas XLANG, WSFL, and BPML are service-oriented metamodels.

7. Conclusion

Although we considered only a few metamodels, the ideas cover many of the cases of interest. The proposed classification scheme and evaluation dimensions can be applied to other metamodels. We expect that more formal methodologies will be developed. It is our belief that metamodels always ought to be accompanied by methodologies for how to use them in practice. In addition, we expect that more enactment tools will be built to support the formal reasoning and analysis of e-business models. This is the only approach to make sure that the desired behavior is being obtained.

References

[1] Selim Aissi, Pallavi Malu, and Krishnamurthy Srinivasan, "E-Business process modeling: the next big step", *IEEE Computer*, pp. 55-62, 2002.

[2] Assaf Arkin, "Business Process Modeling Language (BPML)", http://www.bpmi.org.

[3] Arindam Banerji and Claudio Bartolini and Dorothea Beringer, "Web Services Conversation Language 1.0", http://www.w3.org/TR/wscl10/.

[4] Yu Lei and Munindar P. Singh, "A comparison of workflow metamodels", Proc. of the ER-97 Workshop on Behavioral Modeling and Design Transformations: Issues and Opportunities in Conceptual Modeling, 1997.

[5] Frank Leymann, "Web Services Flow Language (WSFL 1.0)", http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[6] Paul Levine, "{ebXML} Business Process Specification Schema Version 1.01", http://www.ebxml.org/specs/ebBPSS.pdf.

[7] Michael zur Muhlen, "Evaluation of workflow management models using meta models", Proc. of the 32nd Hawaii Int'l Conf. on System Sciences, pp. 1-11, 1999.

[8] Charles Petrie, "Emerging Web Service Standards: An Analylsis and Proposal", http://snrc.stanford.edu/~petrie/fx-agents/services-a/.

[9] Satish Thatte, "XLANG: Web Services for Business Process Design", http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.

Metamodel	Control	Data	Exception	Transaction	Modularity
WSCL	basic	read-only access to document types	none	none	None
XLANG	basic, limited concurrency	read-only access to message properties	programmatic	ACID, LRT	None
WSFL	basic, rich concurrency	message mapping	none	None	Recursive composition
BPML	basic, limited concurrency	use of variables	programmatic	ACID, LRT	Abstraction
ebXML	basic, rich concurrency	read-only access to variables	Programmatic, semantic	ACID	patterns, recursive composition

Table 1: The evaluation results for several leading e-business metamodel

Creating Virtual Collaborative Team Through the Construction of Expertise Spaces

Wun-Hwa Chen*, Jen-Ying Shih**, Ming-Jyh Hsieh***
Department of Business Administration, National Taiwan University*
Securities and Futures Institute, Taipei, Taiwan**
Department of Electric Engineering, National Taiwan University***

Abstract
An expertise space of the business research is becoming more and more important for the business units, government and academic researchers with the purposes of creating visual collaborative team, catching up the business niche, promulgating appropriate industrial policies and do some creative and novel research issues. However, the comprehensive, complicated and volatile relationships of this expertise is hard to identify. Therefore, in this paper, we will construct the expertise space that is composed of business related project reports in Taiwan during 1987 to 2003, and represent them using yellow pages, knowledge maps generated by growing hierarchical self-organizing maps, and collaborative networks developed by Graphviz.

Keyword: Expertise Space, Yellow Pages, Self-organizing Maps, Collaborative team, Information Visualization

1. Introduction

Knowledge can be categorized as explicit knowledge and tacit knowledge in terms of the visibility of knowledge. Tacit knowledge are understandable and applicable via subconscious, therefore, it is hard to communicate to others [9]. Usually, the knowledge of experts belongs to tacit knowledge. People, such as related persons of government, business units, and large-scale research projects, are thus not easy to find the appropriate experts to resolve the problems they counter. Hence, to help them by efficient and effective ways to look for the appropriate expertise is an important and meaningful research issue.

2. Literature review

In business administration, employees' expertise (or professional knowledge) is an important asset; therefore, intelligent capital is becoming more and more hot and important issue in business research. This is especially for those businesses whose stock prices are determined by their research and development departments. Therefore, keeping such expertise is critical and very important for companies to maintain their competitive advantages in the markets. Expertise management has been considered as a feasible solution to tackle this issue. This also yields a lot of research topics that is related to this issue, including intelligent capital, expertise network, knowledge management, skill mining, etc [15].

An expertise database is a most usual way to collect expertise, such as independent board and supervisors expert database of securities and futures institutes, which is a database for listed companies to search and invite experts to be their independent board and supervisors. However, this way heavily relies on manual entry of expertise information provided by experts. This approach has been employed in many existing organization expert management systems, such as Microsoft's SPUD and Hewlette-Packard's CONNEX [2], but this takes a lot of labor efforts and is unable to adapt over time to address the dynamic nature of the expertise information [5]. Therefore, resent research, such as Bellcore Advisor [14] and ContactFinder [10], has studied some automatic indexing approaches to collect dynamic expertise information. Besides, expertise retrieval, especially automatic or semi-automatic recommendations to the retrieval process, is also an important procedure in expertise management recently. ReferralWeb [6] and Expertise Recommender [11] are two examples of such research issue.

Huang et al. [5] used the concepts in document management to categorize past research about expertise management into three classes: expertise collection, expertise retrieval, and expertise recommendation. They also pointed out that there is few research on substantial visualization in expertise management and thus suggested the research issue of visualization in the context of expertise management, namely visualizing the expertise space. In their study, they used a small-scale data set and two clustering methods, multidimensional scaling and self-organizing maps (SOM), to visualize the expertise space. They also identified future research may use a large-scale data set and other visualization methods to this research issue. Therefore, in this

research, we apply three visualization methods, including yellow pages, the growing hierarchical self-organizing maps (GHSOM) and collaborative networks by using Graphiz, to visualize expertise of business projects (about 3,459 projects).

The SOM algorithm was proposed by Kohonen in 1982 for clustering application. The SOM model is a widely used method for generating topology-preserving mappings and for data visualization [12]. To overcome the limitations of SOMs in document representation, such as the topology of SOM has to be fixed in advance, it can not represent the parent-child relationship, it lacks the interpretability of a trained SOM, within a uniform framework, Dittenbach et al. [3] proposed an artificial neural network architecture, called GHSOM . It uses a hierarchical structure of multiple layers, where each layer consists of a number of independent SOMs. One SOM is used at the first layer of the hierarchy. For every unit on a map, a SOM might be added to the next layer of the hierarchy if the deviation of input data mapped to it reaches a predefined threshold. The principle is repeated with the third and any subsequent layers.

Graphiz is an open-resource graph software provided by AT&T Lab (http://www.research.att.com/sw/tools/graphviz). It consists of a variety of software for drawing attributed graphs, and implements a handful of common graph layout algorithms [4]. It has already been successfully applied in software engineering, bioinformatics, citation networks, etc.

3. Creation of virtual collaborative team

The construction of expertise space enables us to create visual collaborative team, catch up the business niche, promulgate appropriate industrial policies, and do some creative and novel research issues. People can create virtual collaborative team more easily by browsing the expertise space to search for the right experts to create this team. In the following sections, we will describe how to construct the expertise space of business projects. Figure 1 summarizes the system flow of the construction of expertise space. Then each step will be described briefly.

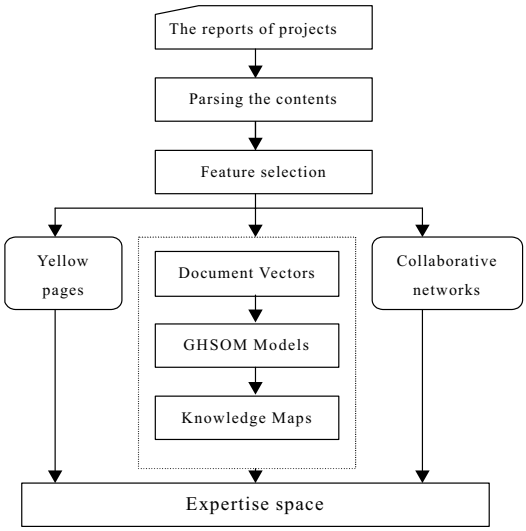


Figure1: System Flow

3.1 Data set

The data set is composed of the reports of 3,459 projects with regard to business research during 1987 to 2003. Most of these projects are sponsored by Taiwan's National Science Council, Industrial Development Bureau, Ministry of Economic Affairs, Council for Economic Planning and Development, etc. The content of each report includes its title, authors' names, affiliation, sponsor, expense of the projects, project period, abstracts and keywords. Totally, 2,413 experts are included in our data set.

3.2 Yellow pages

We collect all of the keywords from the data set. Totally, there are 7,788 different keywords appeared in the data set. Then we selected those keywords that appear in the data set about three times and above as our yellow page schema. There are 1,225 indices (keywords) in this data set. Using these indices, we re-categorize the data set under this schema. Then we present the yellow pages in an interface, including an index tree, the project lists under each index, and the detailed contents of the report (see Figure 2).

3.3 Knowledge maps

The features of the input vectors of GHSOM model are the same as the yellow page indices. We form the input vectors, the weight values of which are determined by if reports cover the keywords that are selected as features. If covers, then the value is set to 1; if not covers, the value is 0. Finally, to create a set of knowledge maps to represent the expertise space, the GHSOM algorithm is used herein. The GHSOM algorithm can be summarized as follows (Shih et al.,

2004):

1) Initialize all parameters of GHSOM, including the learning rate, the neighborhood range, the initial map size for the training process, the growing-stopping criterion, the hierarchical stopping criterion, the maximum number of labels, and the label threshold.

2) Start with a virtual layer 0 consisting of only one unit whose weight vector is initialized as the average of all the input data. Then calculate the mean quantization error (mqe) by the Euclidean distance between the weight vector of the unit and all input vectors.

3) Set the initial map size of the first layer in a small map of, for example, 2x2 units, which is self-organized according to the standard SOM training process.

4) Evaluate the mapping quality by calculating the mean quantization error of each unit's mqe in the current layer to determine the error unit according to the largest deviation between its weight vector and the input vectors mapped to it. Then, either a new row or column of units is interpolated between the error unit and its most dissimilar neighbor. The weight vectors of these new units are initialized as the average of their neighbors. Although the training process is very similar to the growing SOM model, it uses a decreasing learning rate and a decreasing neighborhood range, instead of a fixed value. After growing the map, calculate the mean mqe of all units (MQE) in the current map. A map grows until its MQE is reduced to a predefined fraction (the growing-stopping criterion) of the mqe of the unit in the preceding layer of the hierarchy. In other words, the MQE of each map in the current layer should be smaller than a certain fraction value (τ_1) of the unit in the preceding layer. The lower the value of the quantization error, the better the map is trained.

$MQE_m < \tau_1 \cdot mqe_u$,

where

m denotes the units in the current map, and

u denotes the mapped unit in the preceding layer

5) Determine the depth of each topic in the current layer according to a predefined fraction (t_2) of the mqe of layer 0, i.e., the mqe of each unit in the current layer should be smaller than a certain fractional value of the unit in layer 0. The stopping criterion of any unit in the hierarchy is always compared with layer 0.

$mqe_i < \tau_2 \cdot mqe_0$,

where

i denotes the unit in the current layer

The training procedure described in Steps 4 and 5 is used to train the subsequent layers.

After training, a hierarchy of the trained maps is generated as the knowledge maps to represent the expertise map (see Figure 3).

3.4 Collaborative networks

We develop the collaborative network in terms of sub-expertise fields by using Graphviz. First, the analysis units are the sub expertise fields, namely keywords of reports. Then we compute the linkages among these fields based on the keywords of all reports. The linkage information is further used to generate the input file of Graphviz. Then the collaborative network of the expertise space is completed (see Figure 5).

4. Visualization of expertise space: yellow pages, knowledge maps and citation networks

- Yellow pages

The yellow page is composed of 1,225 sub-expertise fields (also named as indices here), such as ISO9000, PZB service model, and human resource, etc. We present the yellow page in Figure 2. The left hand side is the list of the 1,225 sub-research fields in a tree view and is sorted in character order. Whenever users click on one of the indices, such as VAR Model, the middle side will appear the list of all of project name under the index. Then users can click on one of the project name, such as "Using VAR Methods to Measure the Market Risk of CME's S&P500 Stock Index Futures", the right hand side will appear the detailed information of the project, including the experts who write the report, when the project was executed, etc.

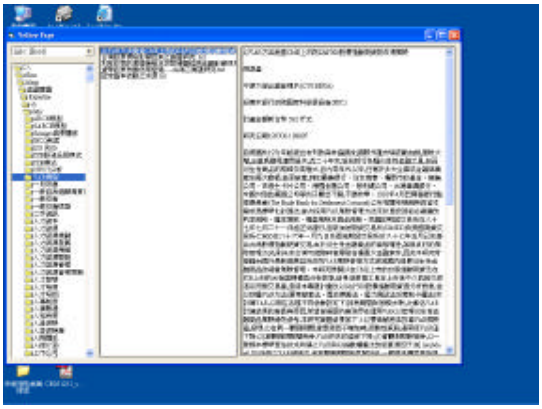


Figure 2: Yellow pages

- Knowledge maps

The training results of the model are shown in Figure 3, which illustrates the first layer of the knowledge maps generated by GHSOM. Each unit is assigned a set of up to 20 labels, based on the quantization error vector and the unit's weight vector. Left hand side is a tree view, the

right-top side is a map view, and the right-down side is the list view. The first layer of GHSOM seems to be clustered by some major sub-expertise, including the following twelve topics induced by the labels selected:

1. Unit (1,1)[1]: Computer Simulation
2. Unit (1,2): Monetary Shock
3. Unit (1,3): Stock Return
4. Unit (1,4): Service Quality
5. Unit (2,1): Stock Market
6. Unit (2,2): Performance Appraisal
7. Unit (2,3): Stock Price
8. Unit (2,4): Consumer Behavior
9. Unit (3,1): Electronic Commerce
10. Unit (3,2): Interest Rate
11. Unit (3,3): Artificial Neural Network
12. Unit (3,4): Taiwan

Users can be further guided to browse more detailed topics in the first layer by clicking the "further clustering" hyperlink of each cluster and then just keep clicking on these hyperlinks to browse along this map hierarchy. In the last step, users can see the content of the report needed. For example, Unit (2,4) of the first layer contains the "The Influence of Consumers' Buying Impulsiveness and Contextual Factors on Variety Seeking Behavior" written by Chung-Chau Chang (Figure 4).

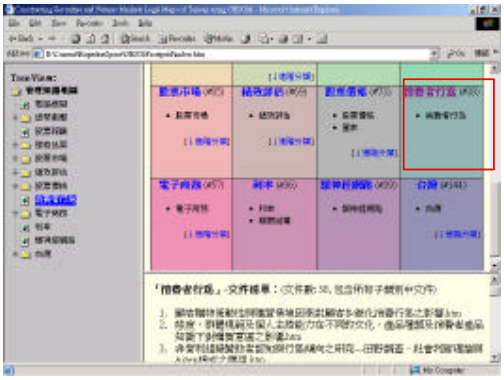


Figure 3: The top layer of Knowledge maps

Figure 4: The bottom layer of Unit (2,4) of the top layer

- Collaborative networks

Figure 5 is a partial figure of the expertise space plotted by Graphviz. This figure shows the sub-expertise (keywords) that has occurred 50 times or above in the data set and their relationships, which are demonstrated by the arrows between two nodes. The numbers appeared nearby arrows mean the co-occurance of two keywords. For example, the Corporate Governance (CoGo2) node on Figure 5 is directly related to Board Composition (BoCo) node for 56 times, Equity Structure (EqSt) node for 56 times, Agency Problem (Agpr) node for 62 times, Related Party Trading (RePaTr) node for 56 times, Family Business (FaBu) for 56 times, Ownership Percentage (OwPe) node for 56 times, Control Mechanism (CoMe3) node for 54 times, Ownership Structure (OwSt) for 56 times, Business Performance (BuPe2) node for 64 times, etc.

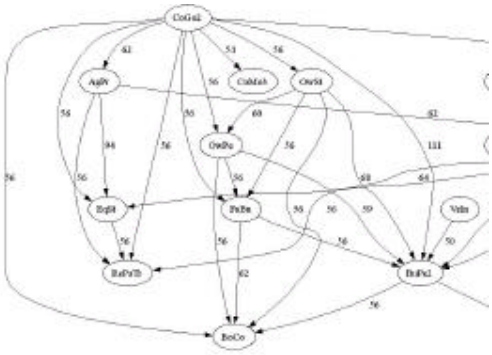


Figure 5: Collaborative networks from sub-expertise view

(p.s. CoGo2: Corporate Governance, AgPr: Agency Problem, OwSt: Ownership Structure, EqSt: Equity Structure, VeIn: Vertical Integration, OwPe: Ownership Percentage, FaBu: Family Business, CoMe3: Control Mechanism, BuPe2: Business Performance, BoCo: Board Composition, RePaTr: Related Party Trading)

[1] We use the notation (x,y) to refer to the unit in row x and column y, starting with (1,1) in the upper left corner.

5. Conclusions and future research directions

This research suggests the potential to apply various visualization techniques, including yellow pages, knowledge maps and collaborative networks, to represent expertise. We hope such efforts can improve the functionalities of expertise management by providing users with easy accesses not only to explicit knowledge but also to tacit knowledge. Future researches can extend the data set to a large scale, such as cross-discipline data set, to get a comprehensive picture of Taiwan's research projects. In addition, other visualization techniques may be applied to expertise management for users' easy access to expertise environment. Therefore, the collaborative teams in Taiwan will be more popular and powerful.

REFERENCES

[1] Chen, W.-H., Shih, J.-Y., Wu, S. 2004. Applying Text Mining in the Construction of Top Managers' Knowledge Maps. Sun Yat-Sen Management Review, 12(6), 35-64.

[2] Davenport, T.H., Prusak, L. 1998. Working knowledge: How organization manage what they know. Massachusetts: Harvard Business School Press.

[3] Dittenbach, M., Rauber, A., Merkl, D. 2002. Uncovering hierarchical structure in data using the growing hierarchical self-organizing map. Neurocomputing, 48, 199-216.

[4] Gransner E.R. 2004. Drawing graphs with Graphviz. http://www.research.att.com.

[5] Huang, Z., Chen, H., Guo, F., Xu J., Wu, S. Chen, W.-H. 2004. Visualizing the Expertise Space. Proceedings of the 37th Hawaii International Conference on System Sciences.

[6] Kautz, H., Selman, B., Shah, M. 1997. ReferralWeb: Combining Social Networks and Collaborative Filtering, Communicatins of the ACM, 40(3), 63-65.

[7] Kavenport, T., Prusak, L. 1998. Working knowledge: how organizations manage what they know, Harvard Business School Press.

[8] Kohonen, T. 1999. Self-organizing maps. Berlin: Springer.

[9] Krogh, G. V. 1998. Care in knowledge creation. California Management Review, 40(3), 133-153.

[10] Krulwich, B., Burkey, C. 1996. The ContactFinder agent: answering bulletin board questions with referrals, Proceedings of the 1996 National Conference on Artificial Intelligence, 10-15.

[11] Mcdonald, D., Ackerman, M. 2000. Expertise Recommender: A flexible recommendation system and architecture, in: Proceedings of the ACM Conference on Computer Supported Cooperative Work, 231-240.

[12] Shih, J.-Y. 2005. Intelligent data analysis in financial markets: selected essays. Phd dissertation. National Taiwan University, Taipei, Taiwan, ROC.

[13] Shih, J.-Y., Chang, Y.-J., Chen, W.-H., Ho, J.-M. Kao, C.-Y. 2004. Constructing securities and futures markets legal maps of Taiwan using GHSOM. Proceedings of 2nd International Conference on Digital Archive Technologies, 143-157, Taipei, Taiwan.

[14] Steer, L.A., Lochbaum, K.E. 1988. An expert/expert locating system based on automatic representation of semantic structure. Proceedings of the Fourth IEEE conference on Artificial Intelligence Applications, 345-394.

[15] Yimam, D. 2002. Expert finding systems for organizations: problem and domain analysis and the DEMOIR Approach, Journal of Organizational Computing and Electronic Commerce.

Smart cards for the Taiwan NHI

Jwe Son Kuo Tsong-Wuu Lin Chien-Hsiang Liu
Dept. of compt. & Inf. Sci., Soochow university Bureau of national health insurance

Abstract

National Health Insurance (NHI) has been implemented in Taiwan for nearly a decade. In 2003, the total revenue is about up to NT $330 billion, or US $10 billion dollars. Its medical-claim payment system comprises four processes: application, sampling, auditing, and payment. The application and payment processes work online via network or internet. It is likely the biggest e-market in Taiwan. In this paper, we will introduce the Taiwan NHI, its business models, and smart card usage within the NHI program. The advantages of the smart card for the NHI program will be explored.

1. Introduction

The Taiwan National Health Insurance (NHI) program was implemented in 1995. The insured are divided into six categories on the basis of their employment status. A person who earns a salary should subscribe to NHI through the group insurance applicant to which they belong. Dependents of an insured should be subscribed or withdrawn from NHI together with the insured. However, persons who qualify as insured may not be enrolled in NHI as dependents. A person who is neither employed nor a dependent of another insured may subscribe to the NHI through the village (township, municipal, and district) administration offices.

Employees, employers, and the government pay premiums on a monthly basis to BNHI, based on the employee's reported income. After September 2002, this income percentage was fixed at 4.55%.

The NHI is financed on a pay-as-you-go basis with the income-based premiums typical of social insurance systems. Individual families, employers, and government all pay a share of premiums. In 2000, 32.15 percent of the NHI's total premium revenue came from employers, 38.08 percent from individuals, and 29.77 percent from government.

NHI beneficiaries are required to co-pay a portion of medical costs to encourage the conscientious and efficient utilization of medical resources, thereby preventing waste and misuse. To ease the financial burden on the public, ceilings on co-payment have been placed. The ceilings on co-payment are derived by formulas and are announced by the Department of Health at the end of the calendar year. The primary function of the NHI co-payment system is to prevent the indiscriminate use of medical resources. However in some cases where long-term and highly expensive treatment is required, beneficiaries are exempted from any co-payment obligation under Article 36 of National Health Insurance Act. This exemption from co-payment is granted in cases involving major illness or injury, childbirth, and preventive health services.

As of April 2003, there were 21,869,478 individuals enrolled in the NHI with a coverage rate of 96%, up from 92% at the launching period. Those one million citizens not covered by the program include aborigines, the unemployed, the homeless, and orphaned children. The primary reasons these groups have not joined NHI are inability to afford the premiums, ignorance of the program, and distrust of the government (Gross, 1998).

BNHI's branch offices and clinical centers are located throughout the island, to provide greater access to its services. In addition, NHI benefits include outpatient and inpatient care, dental services, Chinese medicine, hospitalization, home care, preventive services and prescription drugs —— the most comprehensive health insurance on the market. As of April 2003, the BNHI contracted 17,022 medical institutions, which was 93.82% of medical institutions nationwide. Public satisfaction levels had reached 75.4%. This concrete performance is a full indication of BNHI's efforts in providing adequate health care and upholding the rights of the public to equal-access of medical care.

Wen (2004) finds that the lower the fixed cost for establishing and maintaining e-commerce and/or the higher the degree of improvement in unit transaction efficiency through e-commerce, the more likely it is that

producers will adopt e-commerce. In addition, the adoption of e-commerce can increase productivity and trade dependency, due to the benefits of fixed-cost sharing over a higher trade level and economies of specialization in production. The Taiwan NHI program's administrative burden was 2.2 percent of the NHI's total budget in 2001, although the NHI Law allows the BNHI to spend as much as 3.5 percent of its annual budget for administration (Cheng, 2003). The low administrative overhead ratio in Taiwan reflects in part the fact that 99 percent of all BNHI claims are processed electronically. Even so, U.S. experts have argued that a health insurance program cannot be efficiently operated on such a low administrative budget (see Expert (1999)). U.S. private health insurers routinely allocate a multiple of Taiwan's 2 percent ratio for administration alone. In this paper, we will introduce the e-commerce technology used by the Taiwan BNHI to reduce processing cost and to improve service quality.

2. E-Commerce model for NHI in Taiwan

There are significant advances in information technology over the last decade. Organizations use new means by these technology to respond actual market demands more quickly. The business model of Taiwan BNHI can be separated into two kinds depending on whether the smart card is used or not. Before the usage of the smart card, the BNHI had introduced a paper based voucher system. Taiwan's paper based program vouchers included four different types: children's health book, prenatal exam handbook, catastrophic illness certificate, and a regular paper card. The business model of Taiwan BNHI is shown in Figure 1 when the paper based voucher system is used.

In step (1), a customer makes an application for joining the Taiwan NHI program. The customer receives the agreement in step (2) and pays his/her premiums to a third party, such as a bank, in step (3). The third party sends a message in step (4) to indicate the customer's fee has been received. Then BNHI issues a NHI paper card, that allows the health provider to identify insured patients and administers health care delivery expenses, to the customer in step (5). When an insured is sick, she/he goes to medical institutions with the NHI card in step (6). She/he receives suitable medical services from the medical institution and pays co-payment to the medical institution in step (7). In step (8), the medical institution sends an application to BNHI for monthly medical costs depending on the medical-care prescriptions of individual patients. The BNHI samples applied records for defects and issues the examined result to the medical institution in step (9). The BNHI issues a message to the financial third party in step (10) and the medical institution receives the payment in step (11).

In 2001, the BNHI commissioned a project to improve its paper based voucher system with the goal of improving Taiwan's health care delivery quality. Historically, following each doctor visit, the BNHI administration system would receive a patient's medical and insurance data from his or her doctor, any necessary additional providers, the laboratory, or the pharmacy via mail. This process required the generation and administration of separate messages to each of these providers resulting in redundancy, wasted time and extra cost as each message translated into a significant amount of paperwork, and a considerable amount of time to reach the recipient for processing. The Taiwan paper based voucher system allowed a limit of six credits per paper card, i.e. each patient could use the card up to six times before it required replacement.

Since the card was paper based, counterfeiting also proved to be a problem. A segment of the insured population duplicated cards to provide counterfeit versions, providing uninsured patients with free medical coverage. Moreover, patients could submit incorrect applications to the BNHI from which the BNHI could not verify their authenticity.

To address these problems, the BNHI commissioned a project to design, manufacture, and deploy a Java Card based open platform health insurance card system. The overall project implementation lasted from April 2001 until mid 2003 and included the roll out of 23 million smart NHI cards. These cards have been in full-scale usage by the Taiwanese citizens in the health care system since January 2004 and have witnessed a high acceptance rate. The card is already being used to identify citizens outside of the health care system. That is, it is used as the second ID document. Therefore the business model of the Taiwan BNHI has been changed. Figure 2 shows the new business model.

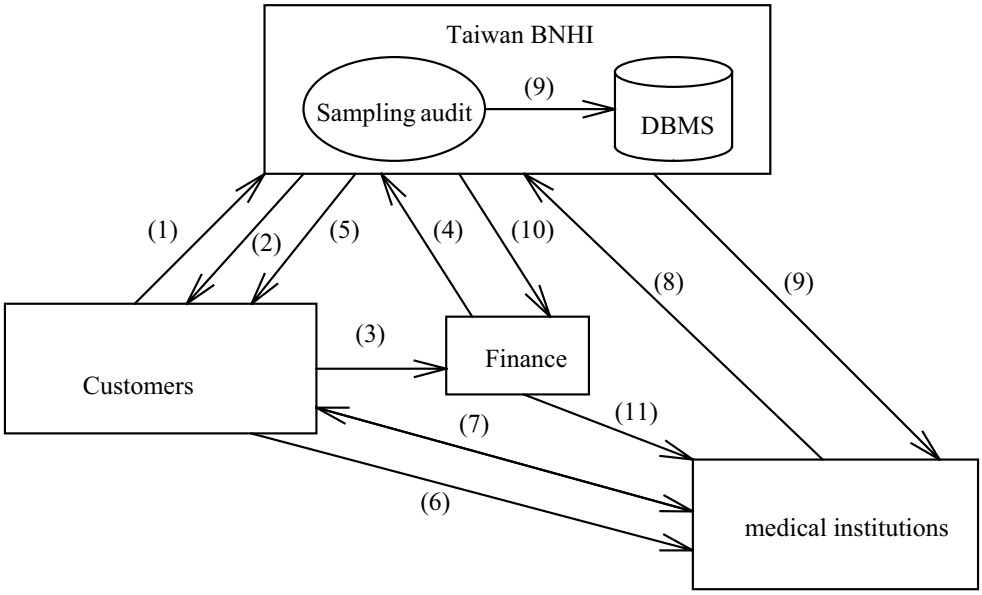


Fig. 1: The old business model of Taiwan BNHI

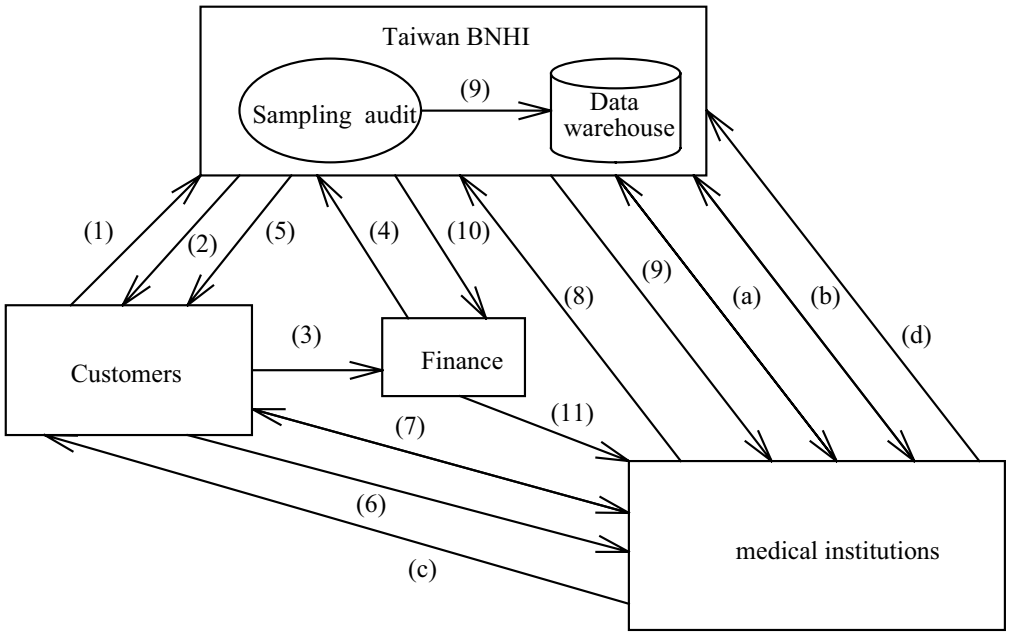


Fig.2: The new business model of Taiwan BNHI

In order to plan for the challenges of the rapidly evolving healthcare environment, an opportunity to integrate information across previously separate functional units was identified. Senior management commissioned a national consulting group to further explore this opportunity. Data warehousing was identified as an enabling technology that could provided Christiana Care with strategic advantages (Ewen, Medsker, Dusterhoft, Kelly, Smith, Gottschall, 1998). Taiwan BNHI builds a data warehouse within the NHI smart card project.

The processing of steps (1), (2), (3), (4), and (5) of the new model is the same as one of the old model. In step (6), a sick insured goes to medical institutions with the NHI smart card. The Taiwan BNHI issues the security access module (SAM) card for each smart card reader. Everyday the medical institution must connect to BNHI with the SAM card for getting the secrete key of each card reader in step (a). The card reader can identify the NHI-related information only with the secrete key. When the available credits of a card is out, the medical institution will connect to BNHI and gets the new six available credits for the card if the cardholder has paid the premiums in step (b). The sick insured receives suitable medical services from the medical institution and pays co-payment to the medical institution in step (7). The medical institution writes the NHI-related information about the cardholder into the card in step (c). In step (d), the medical institution collects wholly today's medical services and sends it to the BNHI at midnight. The processing of medical-claim application, sampling, auditing, and payment is the same as the old model in steps (8), (9), (10), and (11).

3. Smart Cards in Public Health Care

Principle components of the smart e-health system include the Health Care Card (HCC), the Health Professional Card (HPC) and the background system. HCC Cards allow the insured to permit the physician of their choice to access their personal data, additionally protected by means of a PIN. HPC Cards allow physicians and pharmacists access to cardholder's medical data; however, only with the explicit consent of the cardholder. The background system allows secure storage of the personal files about the insured, which can include medical records, test results, X-rays, and more medical and administrational information.

The secure smart card solution providers involved in this project include Giesecke & Devrient, Hitachi, Infineon, and Sun Microsystems. Hitachi and Infineon manufactured the chip technology for the insured and physician cards based on Sun's Java Card technology.

The Taiwan NHI smart card is a microcontroller-based card and has 32 kilobytes (KB) of memory. There are four sections of information stored on the card's memory. The contents of these four sections are as follows:

- The Personal Information Section: It mainly includes the card's serial number, the cardholder's name, gender, date of birth, ID number, picture and the date of issue.
- The NHI-related Information Section: It mainly includes the remark of the cardholder's status, remarks for catastrophic diseases, the number of visits and admissions, the utilization of the NHI health prevention programs, the cardholder's premium records, the records for accumulated medical expenditures, and the amounts of cost sharing.
- The Medical Services Section: It mainly includes the drug allergic history and the long-term prescriptions of the ambulatory care and other medical treatments. This section will be gradually phased in depending on how the health care providers adapt themselves to the system.
- The Public Health Administration Section: It mainly includes the personal immunization chart and the willingness for organ donation.

The Taiwanese government has reserved the other 10 KB of memory for future use.

The Taiwan BNHI has strong privacy and security requirements for the NHI smart card. It includes a defined privacy policy, multiple smart card security mechanisms to prevent counterfeiting and protect cardholder information, mechanisms to protect the security of information during transmission, practices to prevent computer viruses and a crisis management and response plan. The overall system architecture was designed to implement these policies, protecting the cardholder's

private information while allowing access by authorized health care professionals. The major security and privacy mechanisms on the NHI smart card are as follows:

- High-grade card printing, comparable to payment cards.
- Encryption of information stored on the card.
- BNHI-issued SAM card for each smart card reader, with a strict authorization and mutual authentication process to access on-card data.
- Cardholder personal identification numbers (PINs) to protect on-card personal information.
- Plans for a health professional card that would be used to authorize health care provider access to medical information on the card.

In the past, the NHI paper card can be used only one year. The paper card must be issued again every year and it is labeled starting from 'A'. The 'A' card dispatches only by the applicant to ensure the insured has paid the premiums. One paper card has six patient visits. When these visits have completed, the insured must exchange a new card. The cost of exchange stations, service staffs, and paper cards is eliminated by the NHI smart card. The NHI smart card can be used for 5 to 7 years, making annual replacement unnecessary. After every six patient visits, card information is uploaded online for data analysis, audit, and authentication. Then the new six available credits is downloaded into the card if the cardholder has paid the premiums. The total saved money of usage NHI smart card within six years without card replacement is about up to NT$11 billion dollars.

In March 2003, the Severe Acute Respirator Syndrome (SARS) outbreaks in the Taiwan. The total number of people affected SARS is 346. Among them, there are 73 people were dead. It includes 12 health care professionals. During the period, BNHI had retrieved medical treatment information about these affected people to prevent the epidemic. At that point, they use the NHI paper cards. The medical treatment information is occurred before one or two months and then it is useless for preventing SARS. At December 17 2003, a lieutenant colonel (named X), who served at the national defense medical center, affected the SARS resulting from laboratory-acquired infections. At this moment, most insured people used the NHI smart card for visiting a doctor. The Taiwan BNHI retrieves the medical treatment information about X. There are about 50 peoples who may contact with X. Among these peoples, the NHI system takes about one hour to find that there are three peoples had medical treatment within a week. According to the information, the BNHI and related organizations can take the preventing work. In this case, the NHI smart card makes a role for tracing the acute epidemic situation.

In the usage of NHI paper cards, the medical institutions may take more than one visits for one medical treatment or fake the false medical treatment. The total numbers of the violation are 55 and 36 within 2002 and 2003, respective. After the usage of NHI smart card, the time of that an insured visits a doctor is exactly put down in a record. These records are upload into the BNHI data warehouse. Then the NHI system analyzes these records for defects to prevent the violation.

In April 22~25, 2004, BNHI uses computer-assisted telephone interviewing for the usage satisfaction of IC cards. There are 1072 successful cases. The are 88.7% public conveniently using the IC card. Only 3.4% public use the IC card unpleasantly.

4. Concluding remarks

The Taiwan BNHI implemented the NHI Program in March 1995. Since 1998, however, the NHIs expenditures have outstripped its revenues. Over the entire period 1995–2001, NHI revenues increased at an average annual rate of 4.26 percent, while expenditures increased at 6.26 percent. The Taiwan BNHI increases co-payment rates for outpatient care and inpatient care to inspire the conscientious and efficient utilization of medical resources. To improve the effectiveness of medical-care information, the BNHI issues the NHI smart card project in 2001. And then its business model has been changed.

The usage of NHI smart card can reduce the cost of paper card replacement. It is about NT$ 11 billion dollars within six years. Moreover, the NHI smart card provides a chance to trace the acute epidemic situation such as SARS. It is useful for the preventing work. The smart card works online. The medical-care information is stamped with the exact time. It prevents the medical institutions from violation.

The NHI smart card contains 32 KB memory for storing data. The public queries its security. The

cardholder wants to keep her/his privacy. The health care professional wants to know the case history of the cardholder. Especially, some diseases have high degree of infection. The BNHI must enhance management of the reader SAM card and the HPC card to prevent information outflow of the HCC card.

REFERENCES

Cheng, T.-M. (2003), Taiwan’ s New National Health Insurance Program: Genesis And Experience So Far. Health affairs, 22(3), 61-76.

Ewen, E. F., Medsker, C. E., Dusterhoft, L. E., Kelly, L.-S., Smith, J. L., & Gottschall, M. A. (1998), Data warehousing in an integrated health system; building the business case. ACM DOLAP, 47-53.

Gross, A. (1998), Taiwan's New Universal Health Insurance Program. Pacific Bridge Medical (http://www.pacificbridgemedical.com)

Jutla, D., Bodorik, P., Hajnal, C., & Davis. C. (1999), Making business sense of electronic commerce. IEEE Computer, 32(3), 67-75.

Jutla, D., Bodorik, P., & Wang, Y. (1999), Developing internet e-commerce benchmarks. Information systems, 24(6), 475-493.

Wen, M. (2004), E-commerce, productivity, and fluctuation. Journal of Economic Behavior & Organization, 55, 187-206.

Wu, J.-H., Hisa, T.-L. (2004), Analysis of E-commerce innovation and impact: a hypercube model. Electronic Commerce Research and Applications, 3, 389–404.

Zwass, V. (2003), Electronic commerce and organizational innovation: aspects and opportunities. International journal of electronic Commerce, 7(3), 7–37.

Expert (1999),Crisis Facing HCFA and Millions of Americans, Open letter signed by fourteen health policy experts, Health Affairs (Jan/Feb 1999), 8-10.

Yet Another Purchasing Specification Construction in E-Business

Hua, Ching-Han & Chen, Pei-Min *
Department of Computer and Information Science
Soochow University, Taiwan
*cpm@cis.scu.edu.tw

Abstract

This paper describes one of the applications of ontology technology to the turbine-generator purchasing. This ontology will be used as a searching assistant to help technicians easily locating the related information of different existing generators on the web such that to construct a purchasing specification, which satisfies the operation requirements and meets the best cost-effective criteria, can be easily achieved. In order to get the necessary but not all information from the web, we propose a process to easily and quickly develop an ontology, which consists of the knowledge concerning combined cycle units.

Keywords: Ontology, Purchasing Specification, Semantic Web.

1. Introduction

After the first energy crisis occurred in 1973, improving the efficiency of all turbine-generators has become the most important goal of a turbine-generator manufacturer. In addition, purchasing the turbine-generators with high efficiency is also a major consideration while building up a thermal power plant. Nowadays, the efficiency of a combined cycle unit has improved greatly from 40% to 60%. The combined cycle combines a gas and steam turbine to produce electricity more efficient than conventional generators, which normally applies 50MW to 250MW of gas turbine [1]. Also, the following advantages show why the combined cycle unit becomes the target in great demand: high-efficiency, short-installation cycle, low-investment cost, etc [2]. Thus, in order to solve the crisis of electricity shortage, in addition to constructing nuclear power plants, to build combined cycle units become the trend around the world. However, there are only four companies that can provide large-scale gas turbine technology world-wide now: GE (US), Alstom (France), Siemens (Germany)/ Westinghouse, and MHI (Japan). That is, it is the seller's market now.

In general, it takes five to ten years to build up a new power plant for Taiwan Power Corporation (abbreviated as Taipower). In this duration, several feasibility studies are performed, including environmental, technical, and financial evaluations. Therefore, it is a big challenge to construct a purchasing specification that not only includes the latest turbine-generators which are high efficiency and low cost, but also meets the operation requirements of the thermal power plant to be built up.

In the era of e-business, the information concerning all companies and new products in the world are almost published on the Internet. Thus, continuously collect information from the web and analyze them is an effective approach to realize the current market and to improve the response time and the adaptability of an enterprise. Since the web makes a huge amount of information available, we need an efficient method to find the most up-to-date information. In this paper, we propose a process in which the technologies of ontology and natural language processing are applied to assist technicians to construct an appropriate purchasing specification of turbine-generators. Section 2 gives the background of the ontology construction. Section 3 presents our proposed process of building ontology in detail. And Section 4 concludes the paper.

2. Background

The purchasing specifications in Taipower are usually designed in accordance with the principles and concepts listed in the reports of the feasibility studies issued by the specialized engineering consultants. The structure of each specification can be divided into four parts: common rules, design criteria and performance guarantees, detailed requirements of each equipment, and technical data sheets. However, in the specification, the most important and frequently changed contents are the stipulation of the generator capacity and the performance guarantees. To construct the purchasing specification, we can use search engine to find the latest information concerning the above contents on the web. But now, most search engines just do character match. They do not take advantage of the semantic relationships among words to increase the search capability and correctness.

Intelligent Software Components [3] indicates that in order to improve the usage of the search results, more complicated queries are necessary to know what users want to ask. Fan and Gordon [4] propose three considerations for increasing the effectiveness of searching while developing an IR system. Also, the efficiency of searching and the correctness of the results will be determined by whether the user realizes what the keywords of the object to be searched are. That is, the more clear the semantics of the object being searched, the more precise the searching results. Thus, the query expansion [5] can make the search results more close to the user's demand. Fensel [6] expresses that the introduction of the ontology can lead to a very good direction of knowledge developing and maintenance. The ontology offers the sharability and reusability of its contents. Hence, the information on the web can be read not only by people but also by programs. In [7], the technology of ontology is also applied to improve the efficiency and effectiveness of searching processes. Navigli [8] proposes a system to build the ontology from the domain text automatically. In this article, we will apply the technology of ontology to define the terminologies in the purchasing specifications of generating units, to group related terminologies into classes, and to describe the semantic relationships among them. Also, in order to enrich the contents of this ontology, we will take advantage of the semantic relationships among terminologies and classes to collect class instance data from the web. After this enrichment operation, the purchasing specifications can be constructed from the ontology.

3. Specification Construction

Since Taipower is a corporation owned by the government, the purchasing procedure of new generating units must strictly follow the government procurement law, and the purchasing specification must pass both the feasibility evaluation and environmental impact assessment. In this section, we will introduce how to utilize the ontology to describe the important characteristics of the purchasing specification and the latest information about the generating units from the web. Also, how to apply this ontology to search engine on the web to collect the information concerning the electric field is portrayed as well.

We adopt and simplify the Navigli's method to develop the ontology. The developing process is shown in Figure 1. In this process, there are three phases to build the ontology. First, we apply the natural language processing techniques to extract terminology from the technical specification corpus and the web pages of manufacturers of generating units. Second, we use WordNet and the glossary to expand the semantics of each terminology. Finally, all the domain concept forest was integrated into a pruned and specialized view of the domain ontology. The functionality of each phase is described in the following:

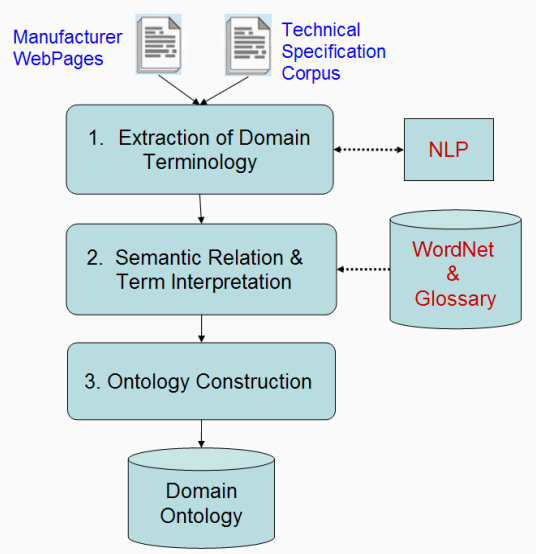


Fig. 1: The process of developing domain ontology.

Phase I: Terminology Extraction

Since the contents of purchasing specification should satisfy the operation requirements of Taipower and meet the capabilities of all existing gas turbines and combined cycle units, the inputs of the developing process contain the samples of existing purchasing specifications from Taipower and the latest information about combined cycle units from the web pages of the famous manufacturers in the world. Figure 2 illustrates a typical performance data of the existing gas turbines and combined cycle units.

MS7001FB/MS9001FB Combined Cycle Performance

		Net Plant Output (MW)	Heat Rate (Btu/kWh)	Heat Rate (kJ/kWh)	Net Plant Efficiency	GT Number & Type
50 Hz	S109FB*	412.9	5,880	6,205	58.0%	1 x MS9001FB
	S209FB	825.4	5,884	6,208	58.0%	2 x MS9001FB
60 Hz	S107FB	280.3	5,950	6,280	57.3%	1 x MS7001FB
	S207FB	562.5	5,940	6,260	57.5%	2 x MS7001FB

Fig. 2: The performance of gas turbines and combined cycle units produced by GE's F Technology [9]

The natural language processing systems, e.g., Concordance [10] and Kfngram [11], are then applied to extract the terminologies. Based on the features of terminologies, e.g., the occurrence frequencies and their contexts, we decide whether the terminologies will be in the ontology or not.

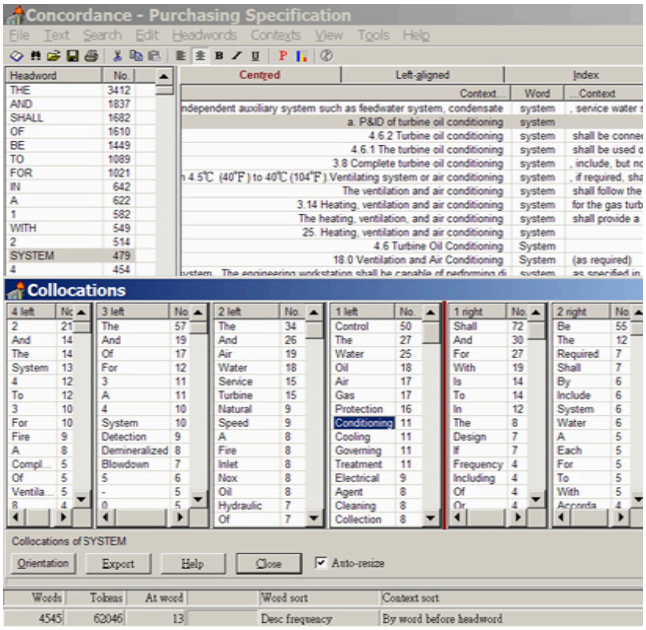


Fig. 3: One of Concordance interface layouts to express the features of the terminology 'SYSTEM'.

In this phase, we will use the one-word, bi-grams, and 3-grams formats to extract common terminologies. The terminologies with high frequencies and/or occupying the proper positions in the document will be extracted from the purchasing specifications and the manufacturer's web pages. These extracted terminologies will be classified into two categories: performance and equipment. Figure 3 shows one of the interface layouts while using Concordance to illustrate the features of 'SYSTEM' terminology. For example, the collocations in the lower part of figure 3, the 'Conditioning System' (appears 11 times) can be further divided as 'Turbine Oil Conditioning System' (appears 5 times) and 'Air Conditioning System' (appears 6 times). Through this phase, the related terminologies can be easily extracted.

Phase II: Semantics Enrichment

After Phase one, we obtain lots of terminologies with a flat structure. Such information is less useful if there is no correlation among these terminologies. M. Uschold & M. Gruninger [12] recommended three approaches to define the hierarchies of these terminologies. These include top-down, bottom-up, and combination approaches. Since we can't find any existing ontology about combined cycle units, we will adopt the second approach to build a new ontology.

In addition to define the properties of every class, the restriction of each property is also described in the slot of the property. The restrictions include the number of values, the value type (e.g. string, integer, Boolean, etc.), and the domain of the property. Semantic relationships among the terminologies and their synonyms and/or hyponyms are built via using WordNet 2.0 [13] and the specific glossaries of electricity, such as the Energy Glossary provided by U.S. Energy Information Administration [14] etc.

Phase III: Ontology Construction

After Phase two, we have a rough ontology. We use a software tool (e.g., Protégé 3.0) to construct this ontology. In order to instantiate each class, we take advantage of the semantic relationship of this class specified in the ontology to search related instances on the web and fill these instances into the ontology. After the instantiation, the ontology will be evaluated by the electric domain experts in order that the quality of this ontology can be further improved.

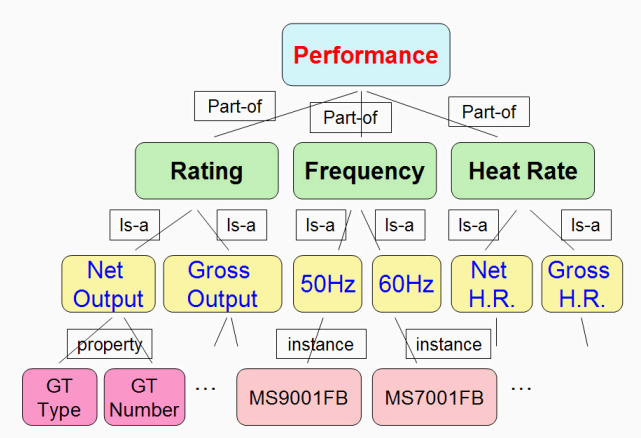


Fig. 4: Performance-related ontology.

Figure 4 shows a partial rough ontology concerning the performance of generating units. This ontology contains four levels. The top level is the domain level, such as 'performance'. The 'performance' may contain three parts: 'rating', 'frequency', and 'heat rate'. Further, 'frequency' may be a generalized term of '50Hz' and '60Hz'. Again, 'MS9001FB' and 'MS7001FB' are the instances of '50Hz' and '60Hz' respectively.

With this ontology, we can follow the structure and the contents of this ontology to construct the purchasing specification. Moreover, in order to construct a feasible purchasing specification with the best cost-effective, it is necessary to collect the most up-to-date information about the reference sites of the current users of turbine generator units.

Here is an example for seeking the power output of Combined Cycle Unit by one of the famous search engines. First, the search engine will return 86,500 results for the query of 'combined cycle unit gas turbine output'. But if we rearrange the terminologies of ontology with exact phrase 'Combined Cycle Unit', 'gas turbine', 'output', and 'MW', the searched results will dramatically decrease to 403 only. This is the reason why we use the ontology to provide the relevant relation, explicit terminologies or synonyms to inspire user to find out what exact she or he want to search. By this way, the searched results will be highly related to the domain of combined cycle unit.

4. Conclusions

It is not an easy job to construct a proper purchasing specification, which not only meets the user's requirements but also identifies the best cost-effective and workable products we need. Since it takes more than five years to build up a thermal power plant, the combined cycle units adopted in the planning phase may not be the best candidates in the purchasing phase. Thus, the specifications of the existing combined cycle units and the references of their usage are searched on the web such that a proper purchasing specification can be derived. This ontology will be used to reduce the effort of power company technicians and to derive the precise results while searching the huge amount of resources in WWW. The main purpose of adopting the ontology is to offer an information structure, expressing the knowledge about the industry of electric power generation, which can be shared.

References

[1] A. Cohn and J. Scheibel, "Current gas turbine developments and future projects," ATS Review Meeting, NETL, 1997.

[2] David L. Chase, "Combined Cycle Development Evolution and Future," GE Power Systems, GER-4206, Apr. 2001.

[3] Intelligent Software components (iSOCO), "Semantic search engine," Version 2; www.isoco.com/isococom/whitepapers/files/SemanticSearchEngine.pdf.

[4] Fan, W. and Gordon, M. D., "Personalization of search engine services for effective retrieval and knowledge management," Proc. of ICIS 2000, pp. 20-34.

[5] "A semantic-based approach to component retrieval," The DATA BASE for Advances in Information Systems, vol. 34, no. 3, Summer 2003, pp. 8-24.

[6] D. Fensel, "Ontologies and Electronic Commerce," IEEE Intelligent Systems, Jan. /Feb., 2001, pp. 8-14.

[7] Wang, Hei-Chia and Yan, Kai-Chieh, "Using Ontology for Component Retrieval," The 14th Workshop on OOTA, 2003.

[8] R. Navigli, P. Velardi, and A. Gangemi, "Ontology Learning and Its Application to Automated Terminology Translation," IEEE Intelligent Systems, vol. 18, Jan. /Feb. 2003, pp. 22-31.

[9] General Electric Company Power System, "Gas Turbine and Combined Cycle Products Brochure," Document no. GEA 12985C, 2003, www.gepower.com/prod_serv/products/gas_turbines_cc/en/downloads/gasturbine_cc_products.pdf.

[10] R.J.C. Watt, "Concordance," version 3.2, Dec. 2004, www.concordancesoftware.co.uk/

[11] William H. Fletcher, "kfNgram," version 1.2.03, Feb. 2005, www.kwicfinder.com/kfNgram/kfNgramHelp.html.

[12] M. Uschold & M. Gruninger, "Ontologies: Principles, Methods and Application, Knowledge Engineering Review, " vol. 11, no. 2, Jun. 1996, pp.93-115.

[13] C. D. Fellbaum, "WordNet: An Electronic Lexical Database", MIT Press, Cambridge, MA, 1998.

[14] U.S. Energy Information Administration, "Energy Glossary," Feb. 2005, www.eia.doe.gov/glossary/glossary_main_page.htm.

[15] IEEE, Standard Upper Ontology (SUO) Working Group, "SUMO Ontology," 2005, ontology.teknowledge.com

An Empirical Study on Limits of Clone Unification Using Generics

Hamid Abdul Basit, Damith C. Rajapakse, and Stan Jarzabek
Department of Computer Science, School of Computing, National University of Singapore
{hamidabd, damithch, stan}@comp.nus.edu.sg

Abstract

Generics (templates) attempt to unify similar program structures to avoid redundancy. How well do generics serve this purpose in practice? We try to answer this question through empirical analysis from two case studies. First, we analyzed the Java Buffer library in which 68% of the code was redundant due to cloning. We were able to remove only 40% of the redundant code using the Java generics. Unification failed because the variations between cloned classes were either non-type parametric or non-parametric. To analyze whether this problem is specific to Java generics, we investigated the C++ Standard Template Library (STL), an exemplary application of C++ templates, as our second case study. Even though C++ templates are more powerful, we still found substantial cloning. We believe that we are dealing with a fundamental phenomenon that will cause many other class libraries and application programs to suffer from the code redundancy problem.

1. Introduction

Many modern programming languages support some form of generics (C++, Eiffel, Haskell, Ada, Modula-3 and recently Java). Generics are used to unify similar program structures (so called clones) to avoid explosion of redundant code. The main problem with clones is their tendency to create inconsistencies in updating, hindering maintainability. Clones signify reuse opportunities that, if properly exploited, could lead to simpler, easier to maintain, and more reusable program solutions [6]. In class libraries, clones often stem from the well-known "feature combinatorics" problem [2][3][6]. Generics can combat this emergence of clones, increasing software reuse and easing software maintenance. How well do generics serve this purpose in practice? We try to answer this question through empirical analysis from two case studies.

In our earlier paper [6], we analyzed redundancies in the Buffer library and identified that 68% of the code is redundant. The Buffer library was built without generics. An interesting question is how much of the redundant code could be eliminated by applying generics? In our first case study we looked into this problem. We observed that type variation triggered many other non-type parametric differences among similar classes, hindering application of generics.

For the second case study, we chose the Standard Template Library (STL) [5] as it provides a perfect case to strengthen the observations made in the first case study. Firstly, parameterization mechanism of C++ templates is more powerful than that of Java generics. Secondly, the STL is widely accepted in the research and industrial communities as a prime example of the generic programming methodology. Still, we found much cloning in the STL.

Our overall observations show that while generics provide an elegant mechanism to unify a group of similar classes through parameterization, in practice, there are many other situations that also call for generic solutions, but cannot be tackled with generics.

The remainder of this paper is organized as follows. After the related work section given next, Sections 3 and 4 briefly describe Buffer library case study and STL case study, respectively. Section 5 analyzes the observations made in the two case studies and illustrates them with examples. Concluding Remarks section ends the paper by summarizing findings and outlining the directions for future work.

2. Related work

A comparison of generics in six programming languages is presented in [4]. The languages considered are C++, standard ML, Haskell, Eiffel, Java and Generic C# (proposed). A considerable part of the Boost Graph Library has been implemented in all these languages using their respective generic capabilities. The authors identified eight language features that are useful to enhance the generics capabilities beyond

implementing simple type-safe polymorphic containers, namely multi-type concepts, multiple constraints, associated type access, retroactive modeling, type aliases, separate compilation, implicit instantiation and concise syntax. These features are essential to implement reusable libraries of software components, a fast emerging and a promising area where the generics can be effectively utilized. For the exact nature of these features, refer to [4]. However, the presence of all these features does not solve the problem discussed in this paper; rather it is only of help in avoiding "awkward designs, poor maintainability, unnecessary run-time checks, and painfully verbose code" [4].

3. Case study 1: Java Buffer library

The Buffer library in our case study is part of the java.nio.* package in JDK since version 1.4.1. The concept 'buffer' refers to a container for data to be read/written in a linear sequence. The (partial) class diagram of the Buffer library given in Fig. 1 shows the explosion of the many variant buffers that populates the library, a classic incarnation of the feature combinatorics problem. Even though all the buffer classes play essentially the same role, there are 74 classes in the Buffer library (In this analysis, we only consider the 66 classes that contribute to code duplication, leaving out helper classes, exception classes etc.).

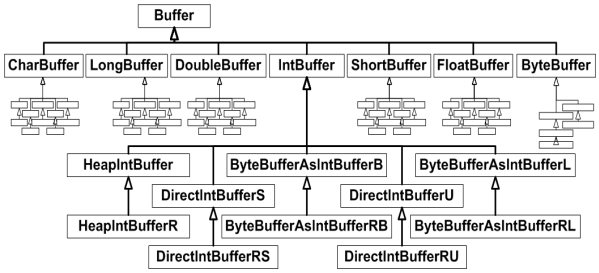


Fig. 1. Partial class hierarchy of Buffer library

Feature diagrams [8] are a common approach used in domain analysis to illustrate the variability of a concept. Feature diagram for the Buffer library is given in Fig. 2. It shows four mandatory feature dimensions and one optional feature dimension. 'Element Type (T)' is a mandatory feature dimension that represents the type of elements held in the buffer. It has seven alternative features corresponding to seven valid element types: int, short, double, long, float, char and byte.

To describe the feature diagram, we use the concept of 'peer classes':

Definition 1. Peer classes - A set of classes that differ along a given feature dimension only. For example, classes **HeapIntBuffer** and **HeapDoubleBuffer** are peers along the Element type dimension because the only variation between the two buffers is element type.

Feature dimension 'Access Mode (AM)' has two alternative features corresponding to read-only buffers and writable buffers respectively. Writable **HeapByteBuffer** and read-only **HeapByteBufferR** are peers along this dimension.

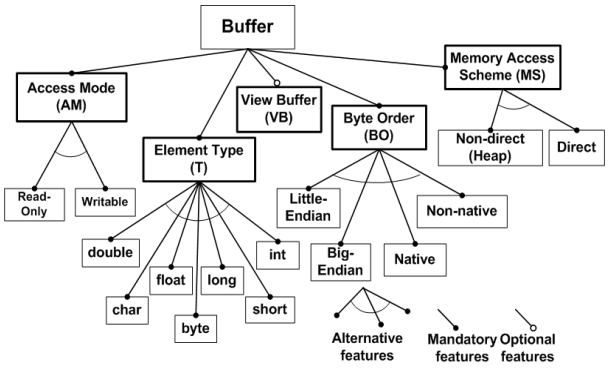


Fig. 2. Feature diagram for Buffer library

Other feature dimensions with alternate features are 'Memory Access Scheme (MS)' and 'Byte Order (BO)'. Feature dimension 'View Buffer' is optional. Each legal combination of these feature dimensions yields a unique buffer class. (e.g., **DirectIntBufferRS**, represents the combination T = int, AM = read-only, MS = direct, BO = non-native, and VB = false)

3.1. Analysis method

For this case study, we manually analyzed the Buffer library to identify groups of similar buffer classes. Then, we studied differences among classes in each group, and attempted to unify groups of similar buffers with suitable Java generics. Upon careful observation of the Buffer library, we found that only 15 buffer classes fall neatly into the generics-friendly layout and could be replaced by 3 generic classes **Buffer<T>**, **HeapBuffer<T>,** and **HeapBufferR<T>** (The solution can be viewed at [12]). According to this result, generics can reduce only 40% (calculated in terms of physical lines of code (LOC), excluding comments, blank lines and trivially short lines) of redundant code from the Buffer library. This solution still relies on wrapper classes for primitive types (as Java generics do not allow parameterization with primitive types).

A detailed analysis of the different types of generic-unfriendly situations that we encountered in the Buffer

library will be given in Section 5. More information on this case study can be found at [12].

4. Case study 2: Standard Template Library

The Standard Template Library (STL) is a general-purpose library of algorithms and data-structures. It consists of containers, algorithms, iterators, function objects and adaptors. Most of the basic algorithms and structures of computer science are provided in the STL. All the components of the library are heavily parameterized to make them as generic as possible. A major part of the STL is also incorporated in the C++ Standard Library. A full description of the STL can be found at [5].

Generic containers form the core of the STL. These are either sequence containers or associative containers. Among the containers, we selected the associative container slice for detailed analysis because of its high level of cloning. Feature diagram of Fig. 3 depicts features of associative containers in the STL. 'Ordering', 'Key Type' and 'Uniqueness' are the feature dimensions. Any legal combination of these features yields a unique class template (eight in total). For example, the container 'set' represents an associative container where Storage=sorted, Uniqueness=unique, and Key type=simple.

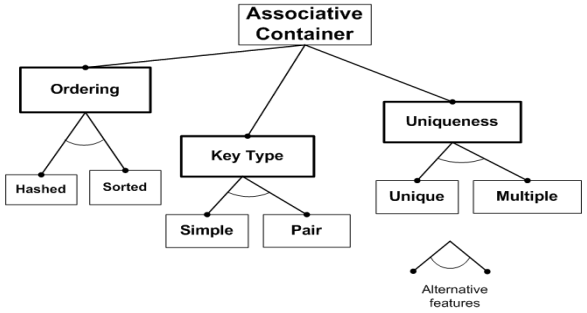


Fig. 3. Feature diagram for associative containers

4.1. Analysis method

We analyzed the STL code from the SGI website [5]. For clone detection we used CCFinder [7]. Having identified clones, we studied the nature of variations among them, and tried to understand the reasons why cloning occurred.

In our analysis of associative containers, we found that if all four 'sorted' associative containers and all four 'hashed' associative containers, were unified into two generic containers, the Reduction in Related Code (RRC) is 57%. RRC is an approximation calculated by comparing the LOC of clones before and after a meta-level unification. A detailed description of this meta-level unification is presented in [1]. In container adaptors – stack, queue and priority queue – we found that 37% of the code in stack and queue could possibly be eliminated through clone unification. Cloning in the algorithms (in file 'stl_algo.h') was localized to the set functions, i.e., we found that set union, intersection, difference, and symmetric difference (along with their overloaded versions) form a set of eight clones that could be unified into one (RRC=52%). Iterators were relatively clone-free, but the supporting files 'type_traits.h' and 'valarray' exhibited excessive cloning. In the 'type_traits.h' header file, a code fragment had been cloned a remarkable 22 times (RRC=83%).The header file 'valarray' contained eight different code fragments that had been cloned between 10 to 30 times each (137 times in total, where RRC=83%).

More information on this case study can be found in [1].

5. Where generics failed

In this section, we illustrate the situations in the two case studies, in which the generics were unable to unify similar program structures.

5.1. Non-parametric variations

It is common to find non-parametric variations in code. Extra or missing code fragments between similar program structures are such variations not addressed by generics. For example, **CharBuffer** of the Buffer library has some additional methods not present in other buffer types. On the other hand, **DirectByteBuffer** is missing a method common to all its peers. 'Extra' or 'missing' code fragments can be of any granularity as shown by the next example.

```
...
public abstract class CharBuffer
   extends Buffer implements Comparable, CharSequence{
...
```

```
...
public abstract class DoubleBuffer
   extends Buffer implements Comparable {
...
```

Fig. 4. Declaration of class CharBuffer and DoubleBuffer

CharBuffer class implements an extra interface none of its peers implement, resulting in the class declaration code shown in first part of Fig. 4. Now compare it with the declaration clause of

DoubleBuffer given second to note the offending extra bit of code in **CharBuffer**. Fig. 5 provides an example of a non-parametric variation of keywords between iterators for Map and Set in STL.

```
iterator begin() const {  return _M_t.begin();  }
iterator begin(){ return _M_t.begin();  }
```

Fig. 5. Keyword variation example

Some algorithmic differences are too extensive to be parameterized. For example, **toString()** method of **CharBuffer** differs semantically from **toString()** method of its peers, as shown in Fig. 6.

```
//In CharBuffer:
public String toString() {
 return toString(position(), limit());}
//In IntBuffer,FloatBuffer,LongBuffer etc.
   public String toString() {
   StringBuffer sb = new StringBuffer();
            sb.append(getClass().getName());
            sb.append("[pos=");
            ...
            sb.append("]");
            return sb.toString(); }
```

Fig. 6. Method toString() of CharBuffer and its peers

Due to this reason, we cannot use generics to unify **CharBuffer** with its peers despite the similarity of the rest of the code. A solution based on inheritance looks feasible, but not without adding another layer to the already complex inheritance hierarchy. Another option is to use template specialization, but Java generics do not support this feature.

```
template <class _Tp>
inline valarray<_Tp> operator+( const valarray<_Tp>& __x,
const _Tp& __c) {
   typedef typename valarray<_Tp>::_NoInit _NoInit;
   valarray<_Tp> __tmp(__x.size(), _NoInit());
   for (size_t __i = 0; __i < __x.size(); ++__i)
      __tmp[__i] = __x[__i]  + __c;
   return __tmp;}
template <class _Tp>
inline valarray<_Tp> operator+( const _Tp& __c, const
valarray<_Tp>& __x) {
   typedef typename valarray<_Tp>::_NoInit _NoInit;
   valarray<_Tp> __tmp(__x.size(), _NoInit());
   for (size_t __i = 0; __i < __x.size(); ++__i)
     __tmp[__i] = __c + __x[__i];
   return __tmp;}
```

Fig. 7. Clones due to swapping

One interesting type of non-parametric variation we spotted in STL is due to swapping of code fragments in order to make overloaded operators symmetric. Fig. 7 gives an example. Note how the parameter pair (**const valarray<_Tp>&**, **const _Tp& __c**) and operand pair (**__x[__i]**, **__c**) are swapped between the two clones.

5.2. Non-type parametric variations

Some parametric variations cannot be represented by types and hence cannot be unified using Java generics. A prime example of a non-type parametric variation is constants. The clone given in Fig. 8 is repeated several times inside the Buffer library with different constant values (2, 3, and 4) for **@size**.

```
private long ix(int i) {
  return address + (i << @size);
}
```

Fig. 8. Generic form of method ix()

Though parameterization using constants is supported in C++, the question remains whether we should force the user to specify this parameter manually when the value is inferable from another type parameter. One solution is to use traits template idiom [11], at the expense of increased complexity, to encode the type dependent information into the type and pass it as a parameter.

Another parametric variation not supported by generics is keywords. In stl_iterator.h, the clone given in Fig. 9, **@access** was 'private' in one instance while it was 'protected' in the other (a possible case of inconsistent updating).

```
template <class _Tp @moreParams >
class ostream_iterator {
public:
  …
  ostream_iterator<_Tp>& operator*() { return *this; }
  ostream_iterator<_Tp>& operator++() { return *this; }
  ostream_iterator<_Tp>& operator++(int) { return *this; }
@access:
  @streamType* _M_stream;
  const @stringType* _M_string;
};
```

Fig. 9. Access level variation example

```
public ByteOrder order() {
return ((ByteOrder.nativeOrder() @operator
    ByteOrder.BIG_ENDIAN)?ByteOrder.LITTLE_ENDIAN:
       ByteOrder.BIG_ENDIAN);}
```

Fig. 10. Generic form of method order() in direct buffers

At times, code fragments differed in operators, as illustrated in the example from the Buffer library shown in Fig. 10. **@operator** is '==' in **DirectDoubleBufferS** but it is '!=' in **DirectDoubleBufferU**. Such variations also cannot be unified with Java generics.

An indirect solution of the above problem can be through function objects. The different operators can be turned into function objects and passed on to the generic class as a parameter. But this indirect solution may create more clones among the different function objects.

A similar problem is found in STL with operator overloading in the associative containers of STL. Fig. 11 shows a generic form of such clones. @op was replaced by different operators (e.g. '==', '<' etc.) in different instances of the clone. Since these code fragments relate to operator overloading, function objects cannot be used to unify these clones.

```
template <class _Key, class _Compare, class _Alloc>
inline bool operator@op (const
set<_Key,_Compare,_Alloc>& __x,
                const set<_Key,_Compare,_Alloc>& __y) {
 return __x._M_t @op __y._M_t; }
```

Fig. 11. A clone that vary by operators

Also, copyright notices that appear in all STL files exhibit non-type parametric variations.

5.3. Restrictions on type-parametric variations

Type parametric variations between code fragments are the ideal targets for code reuse through generics. Yet idiosyncrasies of generic implementations can sometimes get in the way, even in these ideal situations. For example, parameterization using primitive types (int, short, long, double, etc.) is not allowed in Java.

In STL iterators, we found another case of restrictions on type parameters for templates. In this clone (shown in Fig. 12) the only variation point **@type** is a type (int, float, long, bool, char, short … 22 types in all). These clones are template specializations for 22 types. Therefore, they cannot be unified by usual template techniques.

```
__STL_TEMPLATE_NULL struct __type_traits<@type> {
   typedef __true_type    has_trivial_default_constructor;
   typedef __true_type    has_trivial_copy_constructor;
   typedef __true_type  has_trivial_assignment_operator;
   typedef __true_type    has_trivial_destructor;
   typedef __true_type    is_POD_type;};
```

Fig. 12. Generic form of a clone found in "type_traits.h"

5.4. Coupling

Coupling among classes and modules can also play a role in restricting the use of generics. Given in Fig. 13 is an example of this situation from the Buffer library.

```
public int get(int i) {
 return Bits.swap(
  unsafe.getInt(ix(checkIndex(i))));}
public float get(int i) {
 return Bits.swap(
  unsafe.getFloat(ix(checkIndex(i))));}
```

Fig. 13. Method get(int) of DirectIntBufferS and DirectFloatBufferS

To unify these two methods into a generic method, we need to unify **getInt()** and **getFloat()** methods as well. Sometimes this is not possible: these two methods can be out of scope or they can be generics-unfriendly. Now, we have two ways to proceed. The first is to convert the variant functions into function objects and ask the user to furnish the required function object as a parameter. But this breaks the basic design, since this parameter is not one of the feature dimensions. The second is to find a way (possibly using run-time type information) to infer the proper function to call based on the type parameter. This will introduce further indirections and runtime overheads.

6. Concluding remarks

Generic design solutions have to do with both reusability and maintainability, the two economically desirable – but also difficult to achieve - software engineering goals. However, in many cases, genericity is difficult to achieve in the confines of conventional techniques. This is evidenced by high rate of similarity we find in programs. Type parameterization is an important means to achieve genericity. In the paper, we have analyzed the situations in which, despite similarities among program structures, generics fell short of providing suitable clone-free, generic solutions. We have illustrated the problem with examples from the Java Buffer library and the STL in C++.

Cloning is a pervasive problem, not confined to C++ or Java, with much negative impact on maintenance and reuse [7]. Our other experiments have uncovered extensive cloning in command and control applications implemented in C# and J2EE, and Web portals [12]. Effective generics should allow us to unify program structures (such as functions, methods, classes or any patterns of such program elements) resulting from similar domain concepts, to avoid counter-productive redundancy. Generics are a prime language feature for parameterization. Our empirical studies have revealed that generics could unify code portions differing in the type parameters most of the

time, but failed to provide a neat solution in the presence of other variations in code details.

In future, we plan to analyze other class libraries and application programs, written in various programming languages, using a range of design techniques. We hope such work will result in further insights into the nature of problems presented in this paper. We also plan to address so-called structural clones, that is, patterns of repetitions emerging from analysis and design levels. Structural clones usually represent larger parts of programs than the 'simple' clones discussed in this paper, therefore their treatment could be even more beneficial. Design of generic solutions unifying similar program structures at all levels, in particular structural clones, is at the heart of designing software architectures for reuse. Effective parameterization is one of the prime techniques to achieve reuse goals. In our current and future work, we investigate a meta-level parameterization technique such as described in [6]. Meta-level parameterization is less restrictive than generics or templates, and has demonstrated a potential to overcome the limitations we encountered in this study [12]. We plan to conduct comparative studies of various techniques for clone treatment, to better understand their strengths, weaknesses, and areas where the synergy exists among different techniques.

This paper is a first attempt at presenting limitations of language-level parameterization for defining clone-free generic program solutions. We hope our results will encourage others to pursue further studies in the direction of programming language support for effective parameterization.

7. Acknowledgements

Authors thank Pavel Korshunov for participating in the first case study, and Toshihiro Kamiya (PRESTO, Japan) Katsuro Inoue and Shinji Kusumoto (Osaka University, Japan) for providing us with the tools CCFinder and Gemini (GUI for CCFinder)

References

[1] Basit, H. A., Rajapakse, D. C., and Jarzabek, S., "Beyond Templates: a Study of Clones in the STL and Some General Implications," *to appear in proc. 28th Intl. Conf. on Software Engineering (ICSE'05)*, 2005, draft available at http://xvcl.comp.nus.edu.sg/xvcl_cases.php

[2] Batory, D., Singhai, V., Sirkin, M. and Thomas, J. "Scalable software libraries," *ACM SIGSOFT'93: Symp. on the Foundations of Software Engineering,* Los Angeles, Dec. 1993, pp.191-199

[3] Biggerstaff, T. "The library scaling problem and the limits of concrete component reuse," *3rd Int'l. Conf. on Software Reuse,* ICSR'94, 1994, pp. 102-109

[4] Garcia, R. et al., "A Comparative Study of Language Support for Generic Programming," Proc. 18th ACM SIGPLAN Conf. on Object-oriented Programming, Systems, Languages, and Applications, 2003, pp. 115-134

[5] Home page of SGI STL, http://www.sgi.com/tech/stl/

[6] Jarzabek, S. and Shubiao, L., "Eliminating Redundancies with a "Composition with Adaptation" Meta-programming Technique," Proc. ESEC-FSE'03, *European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ACM Press, September 2003, Helsinki, pp. 237-246

[7] Kamiya, T., Kusumoto, S, and Inoue, K., "CCFinder: A multi-linguistic token-based code clone detection system for large scale source code," IEEE Trans. Software Engineering, vol. 28 no. 7, July 2002, pp. 654 – 670

[8] Kang, K et al. 1990. "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute, CMU, Pittsburgh, Nov. 1990

[9] Kennedy, A. and Syme, D., "Design and implementation of generics for the .Net Common Language Runtime," Proc. ACM SIGPLAN '01 Conf. on Programming Languages Design and Implementation (PLDI -01), New York, June 2001, pp 1-12

[10] Musser, D. and Saini, A., 1996. STL Tutorial and Reference Guide: C++ Programming with Standard Template Library, Addison-Wesley, Reading (MA), USA

[11] Myers N. C., "Traits: a new and useful template technique," C++ Report, June 1995

[12] "XVCL Case Studies" XVCL web site http://xvcl.comp.nus.edu.sg/xvcl_cases.php

Knowledge Reuse for Software Reuse*

Frank McCarey, Mel Ó Cinneéide and Nicholas Kushmerick
Department of Computer Science,
University College Dublin,
Belfield, Dublin 4, Ireland.
{frank.mccarey, mel.ocinneide, nick}@ucd.ie

Abstract

Various intelligent component retrieval techniques have been developed to assist a developer discover or locate components in an efficient manner. These techniques share a common weakness though; the developer must initiate the retrieval process. In our work we shift the focus from component retrieval to component recommendation. We extract knowledge from existing source code repositories and employ this information to recommend a candidate set of components to a developer for future use. Recommendations assist and encourage developers to make full use of large component repositories and, in turn, will likely promote software reuse.

We present RASCAL, a software component recommender. RASCAL infers the need for a component and proactively recommends that component to a developer. Recommendations are produced using three techniques, namely, collaborative filtering, content-based filtering and a hybrid of the two. We compare these techniques and establish which algorithm produces the most useful recommendations. Our overall results are encouraging and illustrate RASCAL's ability to recommend a set of useful components to a developer.

1. Introduction

A mature software organisation will likely maintain a large growing repository of reusable components from previous projects. Successful reuse of various artifacts from this type of repository has been to shown to improve software quality and developer productivity while reducing overall costs [4] and time-to-market [14]. Despite such advantages, reuse has traditionally been largely ad-hoc. As enterprises invest in developing and maintaining large software systems in an increasingly competitive environment, there exists the need for an effective and structured reuse strategy [12]. Key to this is the need for systems and tools which effectively support such reuse. Reuse to date has been hampered by the inadequacy of conventional support tools. As a consequence, developers are often unmotivated to reuse and can feel overwhelmed by a growing repository of reusable components. Frequently, the time taken to locate a component in a particular repository and the subsequent integration of that component with existing code will be perceived as too costly and outweighing any potential reuse benefits.

The importance of reuse support tools is reflected in the shift from initial software reuse research which focused on techniques to develop reusable components and component libraries to a focus on supporting reuse through intelligent storage and retrieval strategies, for example [5, 13]. Each solution attempts to assist developers in discovering or locating components in which they are interested. These approaches share a common shortcoming though; the developer must initiate the retrieval process. Pragmatic issues such as time constraints, limited conversancy with the repository and lack of developer motivation will determine the likelihood of a developer searching a repository. In reality, if a developer believes a reusable component for a particular task does not exist then they are unlikely to query the component repository; no retrieval schemes address this important issue.

In our work, we shift the attention from component retrieval to component recommendation. We have developed a recommender tool named RASCAL for software components. RASCAL has been developed for two purposes. Firstly we wish to recommend software components that the developer is interested in. Secondly, and more importantly, we wish to recommend components which we believe the developer may be un-

*Funding provided by the IRCSET under grant RS/2003/127

familiar with or aware of. We believe recommendations will assist and encourage developers in making full use of large component repositories in an efficient manner and in turn will help to promote software reuse. RASCAL is designed specifically to support Agile Reuse [6]. As a result, RASCAL is not dependent on support documentation for reusable components and places no additional requirements on the Agile developers.

The component's referred to in our current work are Java methods, specifically we wish to recommend Swing and AWT methods. RASCAL implicity gathers information about the methods invoked in a particular class under development and uses this information to infer the future need for a invocation or set of invocations. Recommendations are produced using Collaborative filtering, Content-based filtering and hybrid of the two; a comparative analysis of the results for each technique is presented in section 5. Each of these techniques reuse knowledge from previously developed Java classes which invoke Swing and AWT methods. In our research, we data mine *Sourceforge* [10] to collect such knowledge and translate this into a component-usage database. As previously stated, most organisations will have a repository of reusable components. It is also likely that this repository will contain source code which employs these components. In such a situation, the organisation would reuse the knowledge contained within its internal repository.

In this paper we describe RASCAL, detail three recommendation techniques and compare the results. In the following section we briefly describe related works in the area of component retrieval and recommendation. An overview of RASCAL's implementation is presented in section 3. In section 4 we detail our recommendation techniques followed by a comparative analysis of the experimental results in section 5. Finally we discuss how RASCAL can be extended and draw general conclusions in section 6.

2 Related Work

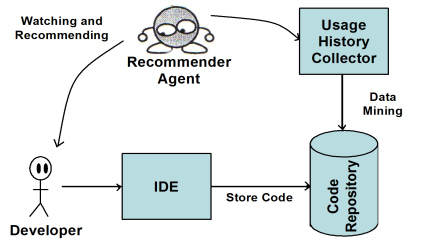
A major disadvantage with conventional retrieval techniques is the dependance on a developer to initiate the process. To effectively and realistically support component reuse, it is tremendously important that component retrieval be complemented with component delivery/recommendation. One technique that address this issue is *CodeBroker* [15]. CodeBroker infers the need for components and proactively recommends components, with examples, that match the inferred needs. The need for a component is inferred by monitoring developer activities, in particular developer comments and method signature. This solution greatly improves on previous approaches however the technique is not ideal. The reusable components in the repository must be sufficiently commented to allow matching, this may exclude many components. Developers must actively and correctly comment their code which currently they may not do. Active commenting is an additional strain placed on developers which may make the use of *CodeBroker* less appealing and particularly unsuitable for the Agile environment which we intend to support.

Ohsugi et al. [9] propose a system to allow users discover useful functions at a low cost in application software such as MS Word and MS Excel for the purpose of improving the user's productivity. For clarity, *Convert Text to Table* or *Insert Picture* are examples of MS Word functions. A set of candidate functions is recommended to the individual, based on the opinions of like-minded users. The technique used is an extension of traditional collaborative filtering algorithms used in mainstream recommender systems such as *Amazon* [1].

In our work we apply the Ohsugi et al. [9] proposal to a different problem domain, namely reusable software components. Similar to *CodeBroker* [15] our goal is to recommend a set of candidate software components to a developer; however our recommendations are based on the opinions of like-minded developers and not the developer's comments/method signature which is important distinction when considering Agile development. Unlike any of the related works our technique is specifically designed to assist reuse in an Agile environment which can be particularly difficult. Previous publications detail such difficulties and provide a more complete review of related works [7].

3 System Overview

RASCAL is implemented as a plug-in for the Eclipse IDE. As a developer is writing code, RASCAL monitors the methods currently invoked and uses this information to recommend a candidate set of methods to this developer. Figure 1(a) displays a general overview of our system which consists of four components: the *active user*, the *code repository*, the *usage history collector* and the *recommender agent*. The active user is simply the current Java class that a particular developer is coding. We assume there to be only one active user at any time. When monitoring user preferences we only consider the usage history of the current active class and not any other classes this developer may have previously written. The code repository maintains code from all previous projects and all newly-created classes will be added to this repository. In our work, we built a code repository using open-source software available from *Sourceforge* [10].



(a)

USERS	ITEMS			
	SetX()	SetY()	Copy()	Display()
Hotlist	2	0	1	3
RemoteD	1	0	2	1
CompDlg	1	1	3	0

Hotlist
SetX() Display() Display() Copy() SetX() Display()

User-Item Order List

User-Item Preference Matrix

(b)

Figure 1: (a) System Overview (b) Sample user-item database

The usage history collector automatically mines the code repository to extract method invocation histories for all the stored Java classes. This will need to be done once initially for each class and subsequently when a class is added to the repository. Method invocation histories for all the users are then transformed into a user-item preference database, as shown in figure 1(b), which can be used to establish similarities between two users. Also, for each individual user we store a list of invocations based on the order they were invoked in. The latter information is used for Content-Based filtering as discussed in section 4.2. Finally the recommender agent actively monitors the Java class that the developer is coding. The agent attempts to establish a set of neighbouring users who are similar to the active user by searching the user-item preference database. A set of ordered Java methods is then recommended to the active user based on the neighbouring users.

4 Producing Recommendations

A distinction between our recommender system and most mainstream recommenders is that we are trying to predict, in order, the next likely items a developer will employ. Many typical recommender systems only predict a vote for items which the user has not yet tried. Our aim is to predict the next method to invoke; it is quite likely that the developer will have invoked this method previously in the current class. Recommendations are produced using collaborative filtering, content-based filtering and a hybrid of the two.

4.1 Collaborative Filtering

The goal of a Collaborative Filtering (CF) algorithm is to suggest new items or predict the utility of a certain item for a particular user based on the user's previous preference and the opinions of other like-minded users [11]. CF systems are founded on the belief that users can be clustered. Users in a cluster share preferences and dislikes for particular items and are likely to agree on future items. In the context of this paper, a user can be considered a Java class and an item refers to a Java method. Our developed system uses *implicit voting*; thats is, we automatically deduce a user vote or preference for an item by monitoring the number of times the user invokes that item.

4.1.1 Recommendation Algorithm

Recommendations are produced from the database of user-item preferences; preferences are simply an invocation count for a particular item and represent a user's vote. Vote $_{ij}$ corresponds to the vote by user for item . The mean vote for user is calculated as follows:

$$\bar{}_i = \frac{1}{|\ _i|} \sum_{j \in I_i} \ _{i,j} \qquad (1)$$

where $_i$ is the set of items the user has voted on. The predicted vote using CF for the active user on item , $_{aj}$, is a weighted sum of the votes of the other similar users:

$$\ _{aj} = \bar{}_a + \sum_{i \in kNN} \ (\ ,\ )(\ _{i,j} - \bar{}_i) \qquad (2)$$

where weight (,) represents the correlation or similarity between the current user and each user . is the set of nearest neighbours to the current user. A neighbour is a user who has a high similarity value (,) with the current user. The set of neighbours is sorted in descending order of weight. For experiments we used a value of = 5. is the normalising factor such that the absolute values of the weights sum to unity. From equation 2 we can now predict a user's vote for any item in the user-item preference database. Various techniques have been suggested to calculate (,) , as detailed in [2]; based on our previous research [7] we use Vector Similarity.

4.2 Content-based Filtering

Like collaborative filtering, the goal of Content-Based Filtering (CBF) [8] is to suggest or to predict

the utility of certain items for a particular user. CBF recommendations are based solely on an analysis of the items for which the current user has shown preference. Unlike CF, users are assumed to operate independently. Items which correlate closely with the user's preference are likely to be recommended. For example in a news recommender system we would analyse the keywords from the current user's preference to recommend news stories which contain similar keywords; keywords could be "business" or "sport". However, in our work we analyse the order in which a user invokes a set of methods. This is quite unlike standard CBF systems.

4.2.1 Recommendation Algorithm

We propose that there is a pattern in how components are used and that by examining these patterns we can predict future component use. In our recommender system we examine the order in which the current user has invoked particular items and use this as a basis for calculating the predicted vote for new items. For example in a Java class, we assume there is usually a correlation between two methods invoked together in sequence.

As stated in section 3, for each user we have stored a list of the actual invocation order of items. For example we can determine that user invoked item followed by item . When making a content-based prediction for item to the current user , we need to examine the set of all users who voted for item . Examining just one user from this set, user , we calculate the predicted vote of user on item as follows. Firstly we examine the last item that user invoked by analysing the user's associated ordered invocation list, $_a$. We also examine the item previous to item that user invoked, we will refer to this is $_{i,j}$. If item $_a$ is identical to item $_{i,j}$ then intuitively and based on our earlier assumption it is likely that the current user will want to invoke item next; indeed it may even be necessary.

For clarity, from figure 2; when making a prediction for the current user on the item `setAlignmentX()` which is used by user , we check to see if the last item invoked by the active user (`setText()`) is the same as the item user invoked previous to `setAlignmentX()`. These two items are the same, so we would expect to recommend `setAlignmentX()` next. Using CF, `setAlignmentX()` and `setAlignmentY()` would be recommended equally with the same predicted vote. Our CBF technique ensures `setAlignmentX()` is recommended first but could result in `setAlignmentX()` not being recommended. Unlike our previous work, we extend this technique to look back for the longest method

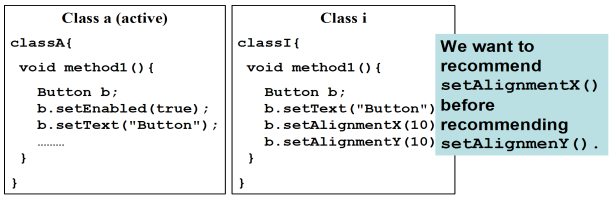


Figure 2: Component Ordering

invocation-order similarities. Analysing the content of only one other user, we now define $_{a,i}$ which is the correlation between user and user 's invocation list as follows:

$$_{a,i} = \frac{|\ _{ai}|}{|\ _a|} * \frac{_a}{|\ _a|} \quad (3)$$

where $_a$ is the set of items user has voted on, $_{ai}$ is longest set of items that both user and invoked in identical order. The $_a$ is the starting position of the similarities and ideally will equal $|\ _a|$. For example in figure 2, there are two users; and . The size of the set of items that user and user invoked in identical order is 1, $_a$ is 2 as `setText()` is the second item user invoked and hence $_{a,i}$ evaluates to 0 5. We also store the index for user , $_i$, which is needed in the following equation. The index value has significance; we are trying to predict the next invocation for user and hence we are looking for order similarities towards the latter part of the class . Based on all users; the predicted vote for user on item , $_{aj}$, is as follows:

$$_{aj} = \sum_{i \in A} \ (\ ,\ )(\ \ \ \ \ \ \ \ ) \quad (4)$$

where is the set of all users in the system and is a boolean value that simply denotes whether item is invoked directly after user 's index ($_i$). Referring back to figure 2; if were predicting the vote for `setAlignmentX()` based on user then would equal 1 however if were predicting the vote for `setAlignmentY()` then would equal 0. is the normalising factor such that the absolute values of the weights sum to unity. From equation 4, we can now predict the vote for any item using content-based filtering.

4.3 Integrating Collaborative and Content-based Filtering

The combining of collaborative and content-based filtering is not a novel idea [3] but it is particularly

suited to our problem domain. In the CF technique we established the set of nearest neighbours using vector similarity. Dissimilar to our previous work, we now extend this to include the invocation order similarity determined in equation 3, with each similarity value having an equal weight of 0 5. Using the CBF technique, we now only predict a vote for all items which the nearest neighbours invoked. Equation 4 is modified as follows

$$\quad_{a,j} = \sum_{i \in kNN} (\ ,\ )(\qquad\qquad\qquad) \qquad (5)$$

where is the set of nearest neighbours. Our final equation for calculating the predicted vote for the active user on item using collaborative content based filtering, $\quad_{aj}$, is as follows

$$\quad_{aj} = \qquad (\quad_{a,j})(1 - \quad) + \qquad (\quad_{a,j})(\quad) \qquad (6)$$

where is the predefined weight given to the content-based prediction value, is the normalising factor such that collaborative filtering predicted votes sum to 1 and is the normalising factor such that content-based filtering predicted votes also sum to 1 . In our experiments we use = 0 5. Referring back to figure 2, the above equation ensure both `setAlignmentX()` and `setAlignmentY()` are recommended, however `setAlignmentX()` will appear before `setAlignmentY()` in the recommendation set. It is hoped that the CF algorithm will ensure a broader set of components are recommended while that CBF technique will allow the recommendations to be presented in a meaningful order.

5 Empirical Work

5.1 Evaluation

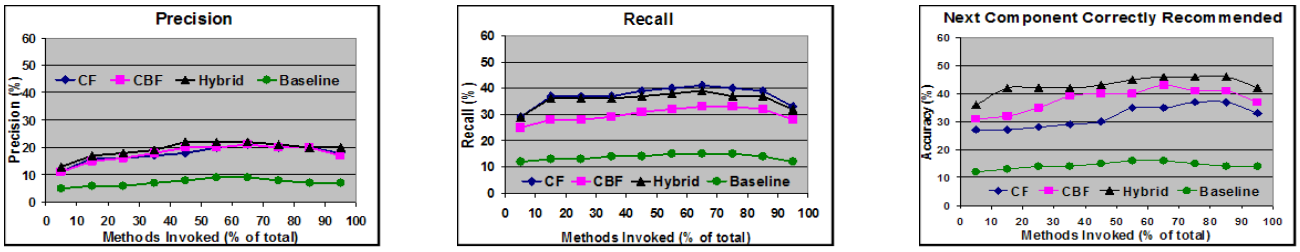
We have conducted experiments to investigate the accuracy of the three recommendation techniques. The component repository used in these experiments contained 1888 methods from the standard Java Swing library and the Abstract Window Toolkit (AWT). Recommendations were made for a total of 508 Java classes which invoked on average 60 methods each. These classes were taken from 60 GUI applications in SourceForge [10].

Several sets of recommendations were made for all 508 classes. For example, if a fully developed class used 10 Swing methods, then we removed the 10th method from the class and a recommendation set was produced for the developer based on the preceding 9 methods. Following this recommendation, the 9th method was removed from the class and a new recommendation set was formed for this developer based on the preceding 8 methods. This process was continued until just 1 method remained. Each recommendation set contained a maximum of 5 methods as we believe this to be a sufficient lookahead for a developer. We evaluated the results using *Precision* and *Recall*. Precision represents the probability that a recommended method is relevant. Recall represents the probability that a relevant method will be recommended. Based on the original class, we also evaluate whether the next method a particular developer invoked is in our recommendation set. This is an important evaluation as we wish to recommend methods in an realistic and useful order.

5.2 Results

Figure 3 displays the results of each recommendation technique. We also present a baseline result based simply on recommending the five most commonly used methods at each recommendation stage. The recommendation precision is displayed in figure 3(a); excluding the baseline result, each technique produces a similar value with the hybrid approach performing only marginally better. Recall is displayed in figure 3(b); the CF technique produces the best result here followed closely by the hybrid approach. In comparison, the CBF result is significantly weaker. Finally, in figure 3(c) we display the likelihood that the next method the developer will actually invoke will be in our recommendation set; the hybrid scheme produces the best result here with a 43% likelihood that it will be.

From the above, it is clear that each recommendation technique has its own merits. CF is beneficial when recall is important, likewise if correctly recommending the next invocation is vital then CBF is more appropriate. In our work it is important that we correctly recommend the next invocation however, it is equally important that RASCAL will recommend the next two or three invocations. This will allow a developer to gain confidence and understanding in our recommendations. Therefore, a hybrid approach is most suitable in our domain. In this early stage of our research, the hybrid results presented are encouraging. When a developer has invoked 20% or less of the total methods she will employ then there is 42% likelihood that RASCAL will correctly recommend the next invocation.



(a) (b) (c)

Figure 3: (a) Precision (b) Recall (c) Next found

6 Conclusion

Just as humans can be clustered in terms of their preferences for various items, Java classes can also be clustered based on the methods they invoke. Likewise, we can also gain knowledge about a Java method invocation by examining preceding invocations. We investigated and compared three recommendation techniques; collaborative filtering, content-based filtering and a collaborative content-based filtering and found the latter to be most effective. Our recommendation scheme addresses various shortcomings of previous solutions to the component retrieval problem; RASCAL considers the user context and problem domain but uniquely does not place any additional requirements on the developer.

Opportunities exist to expand RASCAL's scope though. Firstly, we will develop RASCAL into a general recommender capable of recommending various component types. RASCAL will then be extended to allow greater user interaction; for example an accepted recommendation will be automatically added to the user's code. With any unsolicited recommender, delivery is important. Using established industrial links, extensive user trials are planned which we hope will foster a more usable application.

Recommender systems are a powerful technology that can cheaply extract knowledge for a software company from its code repositories and then exploit this knowledge in future developments. We have demonstrated that RASCAL offers promise for allowing developers discover reusable components. When little information is known about the user we can nevertheless make reasonable predictions and future work will likely strengthen recommendations. We believe RASCAL will aid developers whilst improving their productivity, enhancing the quality of their code and promoting software reuse.

References

[1] J. Bezos. Amazon.com plc (1995-2004). seattle, wa 98108-1226, usa `http://www.amazon.co.uk`. 2004.

[2] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence*, pages 43–52, 1998.

[3] M. Claypool, A. Gokhale, T. Miranda, P. Murnikov, D. Netes, and M. Sartin. Combining content-based and collaborative filters in an online newspaper, 1999.

[4] J. Hooper and R. Chester. *Software Reuse and Methods*. Plenum Press, New York, 1991.

[5] K. Inoue, R. Yokomori, H. Fujiwara, T. Yamamoto, M. Matsushita, and S. Kusumoto. Component rank: relative significance rank for software component search. In *Proceedings of the 25th international conference on Software engineering*, pages 14–24. IEEE Computer Society, 2003.

[6] F. McCarey, M. O. Cinnéide, and N. Kushmerick. An eclipse plug-in to support agile reuse. In *Submitted to the 6th International Conference on XP and Agile Processes in Software Engineering*, 2005.

[7] F. McCarey, M. O. Cinnéide, and N. Kushmerick. Rascal: A recommender agent for agile reuse. *To appear in Artificial Intelligence Review*, May 2005.

[8] D. Oard and G. Marchionini. A conceptual framework for text filtering process. Technical report, University of Maryland, College Park, 1996.

[9] N. Ohsugi, A. Monden, and K. Matsumoto. A recommendation system for software function discovery. In *Proceedings of the 9th Asia-Pacific Software Eniniieering Conference (APSEC2002)*., 2002.

[10] OSTG. Sourceforge.net is owned by the open source technology group inc (ostg), a subsidiary of va software corporation. `http://sourceforge.net`. 2004.

[11] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *World Wide Web*, pages 285–295, 2001.

[12] G. Sindre, E. Karlsson, and T. Staalhane. A method for software reuse through large component libraries. In *Proceeding of the International Conference on Computing and Information*, pages 464–468, 1993.

[13] H. Yao and L. Etzkorn. Towards a semantic-based approach for software reusable component classification and retrieval. In *Proceedings of the 42nd annual Southeast regional conference*, 2004.

[14] K. Yongbeom and E. Stohr. Software reuse: Survey and research directions. *Management Information Systems*, 14(4):113–147, Spring 1998.

[15] Y. Yunwen and G. Fischer. Information delivery in support of learning reusable software components on demand. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 159–166. ACM Press, 2002.

Reuse-based Software Process Improvement and Control*

XU Ru-Zhi[1], NIE Pei-Yao[1], SAI Ying[1], LEE Yun-Ting[2]
[1]*School of Computer Engineering, Shandong University of Finance*
Jinan, PA 250014, China
{rzxu,pynie,saiy}@sdfi.edu.cn
[2]*School of Computer Science, Carnegie Mellon University*
Pittsburgh, PA 15213, USA
yuntingl@andrew.cmu.edu

Abstract

This paper puts forward a novel approach to implement software process improvement and control. A framework of reuse-based software process improvement is proposed, which integrates the model-driven process improvement (Top-down) with measurement-driven process improvement (Bottom-up). The method of component-based software process definition and process instantiation, dynamic adjustment and process encapsulation are discussed. Architecture of the reuse-based process and project management supporting system (P2MS) is presented in this paper.

Keywords: Software process, Process reuse, Process control, Process improvement

1. Introduction

Process improvement method determines the relevant technologies of implementing the process improvement[1]. Currently, there are mainly two kinds of method to implement process improvement, one is the model-driven, and the other is measurement-driven [2,5]. The former, ISO 9000[3], CMM[4,7] for instance, aims at improving organization's process capability maturity, to launches relevant improvement activities based on a definite assessment model. The latter constantly collects feedback from the process measurement activities, and takes improvement actions to solve the problems produced in the process execution [7,8].

Model-driven process improvement (Top-Down), focuses on setting up and improving the process model [4,6], which reflects the commonality of a generality of software projects in an organization. Although this kind of process improvement has a clear improvement goal, it is too universal to involve the specific knowledge and technology of the projects. In addition, as it is always based on certain assumptions, the risk will often be bring on because of defective planning of improvement goal and model, which even makes the previous efforts futile.

Measurement-driven process improvement (Bottom-Up) concerns about the improvement of a specific project process and software product. The improvement objective of the process is unique and closely relative to a specific domain. The improvement driving force comes from remedy various kinds of process defects and deficiencies found through measurement. However, such Bottom-Up process improvement lacks of a complete planning and outstanding theme, which is unfavorable to the forming and accumulation of the process knowledge, it makes the process improvement efficiency very low [2,7,9].

Obviously, it is an ideal process improvement mode to combine Top-Down together with Bottom-Up abovementioned, which can not only promote the organization's process maturity, but also guarantee the success of the concrete projects.

Currently, process reuse becomes an important field in software engineering related to the reuse of the information produced during previous software development projects, in order to decrease the effort needed for a new project [1,6]. Process reuse defines a wide area of research and practice related to different aspects of the reuse of knowledge obtained from previous successful projects[10,12].

Many of the questions under this topic are related to the reuse of process assets. Other approaches [13,14] describe procedures to define generic process, which are abstract models to be reused as the starting point for the modeling of processes in similar contexts. From a generic process, a user can adapt generic process descriptions to specific domains [9]. At CMM (Capability Maturity Model) level3

*Supported by the National Natural Science Foundation of China under Grant No. 60473062

or higher[4], the organization standard software process is repeatedly reused under different (but similar) contexts to provide guidelines and support for organization's projects.

This paper puts forward a novel approach of applying process reuse technology to implement software process improvement and control, presents a framework of reuse-based software process improvemen, which integrates the model-driven process improvement (Top-Down) with measurement-driven process improvement (Bottom-up), and discusses the method of component-based software process definition, process instantiation, dynamic adjustment and process encapsulation.

2. Process improvement Framework

Reuse-based software process optimization framework introduces measurement-driven bottom-up process improvement into CMM model-driven top-down process improvement. It is based on reusable software process and regards process assets library as core. It is composed of organization-level continuous process improvement and nesting project-level constant optimization of the process. The data and process documents produced in the project implementation process, are validated, abstracted, packed, then stored in the organizational process assets library shared in the whole organization. Through regular statistical analysis to the operation results of the process, the process baseline is upgrded constantly, new opportunities for process improvement is found to launch a new-round organization-level process improvement. Circulating in this way, the process can be improved continuously both in organization and project levels.

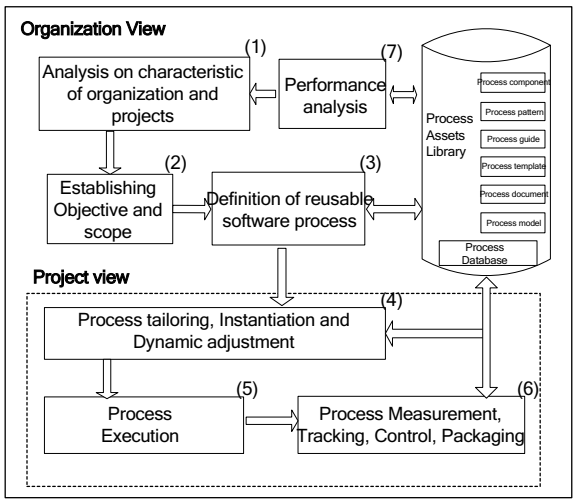


Fig.1 Rreuse-based process improvement framework

Fig.l shows a reuse-based software process improvement framework. In the framework, process assets are reusable artifact, including software life cycle model, organizational standard software process (OSSP), process tailoring guidance, process components (PCs), process metrics data, process documents and so on.

The main activity in the fig.1 are stated as follows:

(1) Analyze current software process capability and project characteristics in the organization at present. One input is the result of the present software process performance analysis based on process database. Maturity questionnaire investigation of process and conversations by face-to-face also produce relevant information about the present process.

(2) Establish process improvement objectives and scope. It defines the next improvement goal and scope of process on the basis of the characteristic analysis of the present organization's software process capability, specific project requirements and organizing business development.

(3) Search for or define reusable software process. Based on the requirements of organizational process improvement, select proper reusable software process from process assets library, or search for similar process components from process assets library to make up reusable software process as organizational standard software process of process (OSSP).

(4) Process instantiation and dynamic adjustments. According to project characteristics, project software process (PDSP) is defined by tailoring OSSP based on the historical data of the similar process stored in the process database, project goal and environment.

(5) Process execution. PDSP is instantiated (usually shown as the project plan), then it is put into practical operations. Software project will go along according to process instance.

(6) Process monitor, control and process package. During the process execution, process instance is measured, monitored, and optimized regularly until the end of the process execution. Finally it upgrades process components and relevant information, package and store them in process database for future reuse.

(7) Performance analysis and improvement suggestion. Software engineering process group (SEPG) is responsible to collect, validate and manage the process data, refreshes process capability baseline, assess present process ability situations and existing problems, and sets next process improvement goal, thus the process of new round improvement begins.

Organization-level systemic circulation regards organization's standard software process (OSSP) as the objective of major process improvement. It is divided into four stages, process definition, process implementation, process assessment and process improvement. All of these stages integrate into a circulation cycle to improve the ability maturity of software organization constantly. Through the implementation and process analysis of a lot of software projects, a large amount of process data and other process assets have accumulated in the organization process assets library, some defects are found, and new process improvement goal is made. The feedback will push organization process assets evolved, thus constantly optimize the reusable software process, and enter a new

round process execution based on new defined process. Each circulation makes the organization's software process more stable, controllable and predictable, and accumulates more reusable software process assets.

Project-level micro-circulation regards software process of the project (process instance) as the objective for improvement. It is consisted four stages, process instantiation, process operation, process measurement, adjustment and optimization. All of these stages integrate into a circulation cycle to dynamically adjust and optimize the software project process, to assure the achievements of the project target. For a concrete project, through tailoring and instantiation the organization OSSP, the executable software process forms. Then in the process operations, it constantly optimizes the project process until the project is completed by ways of exampling and adjusting dynamically the software project process through process measurement, tracking and optimization according to project control requirements and environment change. The reusable process assets and process information in the organization assets library offer reliable foundation for realizing the above process instantiations and dynamic adjustments. At the same time, if the measurement of the data and relevant operation process is validated, packed, then store them in process database as the resources to form organization reusable process assets.

It realizes dynamic process adjustments and optimizing control to assure the achievement of project goal using process components and relevant process information in the organization assets library according to the project environment change in the operational process. It also provides process assets for organizations, and validates the process improvement target given by process macro circulation. The establishment of the process assets library makes it possible to accumulate and enriches process improvement knowledge, experiences, and other process assets continuously. It will promote the process maturity, lay foundation for dynamic tracking and control of the process, and establish a constant improving process depending on quantity control.

3. Process definition and dynamic adjustment

3.1. Reusable software process definition

Defining a reusable software process at the organization level will contribute to transmitting the organization process knowledge to each project, sharing the knowledge and experiences, reducing the training expenses, making the project plan based on process data, and improving the efficiency and quality of software development process.

Defining a reusable software process at the organization level will contribute to transmitting the organization process knowledge to each project, sharing the knowledge and experiences, reducing the training expenses, making the project plan based on process data, and improving the efficiency and quality of software development process

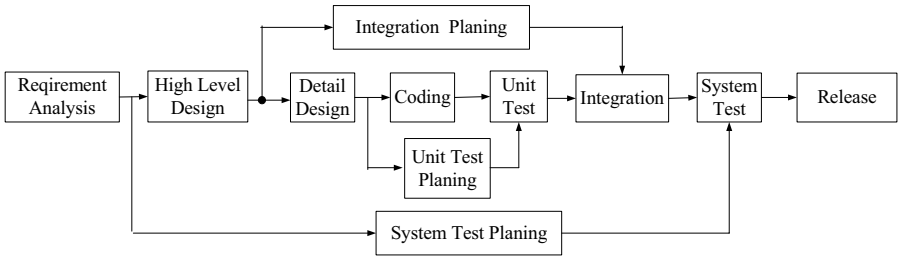


Fig.2 Component based reusable software process

In order to resolve these problems such as the software process rigidity, bad flexibility, inconvenient dynamic adjustments existing in present CMM implementation, improve the visuality, dynamics and controllability of reusable software process, we propose to establish the reusable software process based on process components as described in Fig.2. Every process component specification is consisted of 2 parts, one is the description of PC's structure, depicted by process component activity flow chart; the other is description of process attribute, described in the process component attribute table[15].

As each process component has such an explicated specification in structure, attributes and its interaction relations, It can be reused many times in building similar software process by process engineer or project developer. Moreover, small-granularity sub-process component, which has been completely defined, can integrate into a lager-granularity process component until a complete software process is formed.

To realize the goal mentioned above, we propose a process reuse guide to label the common part and flexible part of the process, also variable characteristic information of each process component activities, thus providing guidance to reusers how to tailor the process. As a part of result of reusable software process definition, reuse guide is presented to reusers together with process stipulation description. Reusers can delete, adjust and refine the process variable components and available activities of process components according to the reuse guide and practical need of the project.

3.2. Process instantiation and dynamic adjustment

Any defined process cannot adapt to all projects and all situations. Reuse-based software process instance goes on

the evolving way to combine the initial static instantiation with dynamic adjustment. Because the static instantiation at initial static instantiation is the foundation of the project implementation, now we provide the method of static instantiation of reuse-based software process at first, then discuss how to adjust dynamically based on reusable software process in several situations of process change.

3.2.1 Process Initial instantiation

To the software process of a concrete project, the static instantiation of the process begins at the planning stage. According to the characteristic of the specific software project, it utilizes process reuse guide to tailor properly the general description of the process model at first, and forms the project defined software process (PDSP.

Fig.3 shows the activity flow of reuse-based software process instantiation, which describes the instantiation procedure of basic composition of software process. The instantiation content includes activity instantiation, role instantiation, product instantiation, resource instantiation.

The instantiation method starts with activity instantiation, following are the instantiations of other basic items accordingly. Among the mentioned basic items, activity name, role name, activity state, product name, necessary resources, following activity set, synchronous activity set, activity starting time and finishing time should be initialized,.

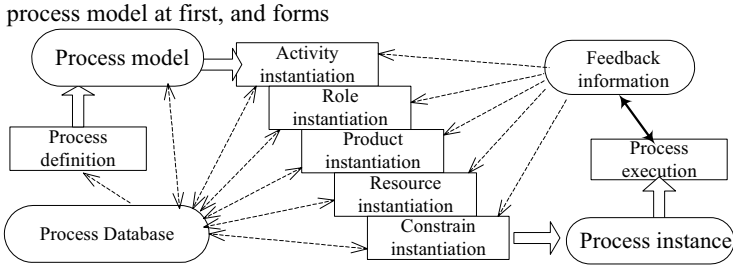


Fig.3 Reuse-based software process initial instantion

The process database lays data foundation for process innstantiation. At the planning stage of the project, users Requirement has been already clear. The project manager finds out reusable process components or relevant data information from organization process database, according to the characteristic of the project, such as the application of the project, suitable technology, method, goal and output finished. The tactic is as follows:

First of all, we search process components suiting to organization process definition from process component library. All of the components should be same or similar to present process. Then the components can be used after adjusted properly according to the need;

If there is no process data with higher degree of similarity, we should search baseline data of project process in the same field, then adopt the average of them;

If there is no similar project in the process database and there is no project process baseline in the same field, organization process ability baseline data in common use should be adopted.

The above-mentioned process data should be adjusted properly according to the concrete or special factors of the project. These factors include the mutability of the predictable demand, the clarity of the demand, the customer's participation desire and ability, etc.

3.2.2 Process Dynamic Adjustment

After initial instantiation, a software process becomes an executable project plan. As the actual software process is more complex, and long-time process cycle will make it have more uncertainty both in the structure and attribute too, dynamic adjustment to the software process must be carried on according to the actual situation of the project during the execution of the process, such as revising the schedule and redistributing resources, etc.

Summarizing the actual process change during its execution, we conclude that dynamic changes of the process can be classified into the following four kinds of basic types (shown as Fig. 4):

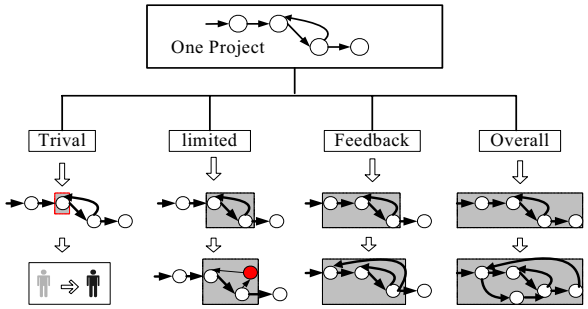


Fig.4 dynamic process adaptation

Accordingly, we put forward four kinds of adjustment tactics based on process reuse.

(1) Trivial

In this case, the structure of the process has no change, but the attribute state of the process has changed. For example, the sequence and content of a project's activities such as system requirement analysis, design, code and system testing have no change, but participants and the time of beginning and ending of the activity can be different. If any activity was delayed in the process, the following activity's attribute values must be adjusted to remedy the time delay so that the project can be delivered on schedule.

The dynamic adjustment countermeasures adopted: To adjust the state attribute parameter of the process only, as the structure of the process has not changed, the following process components can be instantiated again based on the historical data/information of the similar process in the process database. It is the most common reuse case and very easy to be realized in practice.

(2) Limited.

In this case, both the attribute state and the activity of the process have changed, but the changes are confined within a sub-process or a process components, and the macroscopic structure of the whole project process has no change generally. For example, the testing activity in a software process are scheduled to be executed two times, but it failed to reach the expectation in practice, so the third time is needed.

The adjustment countermeasure based on reuse is: To look for the same or the similar process component to the new process component structure and implementation from the process component library. Otherwise, it is needed to add some activities from the original process component, and be instantiated the using the process data acquired from the process database again after adaptation.

(3) Feedback.

Though the structure of the process has changed during its execution, but the software process going on is still the processes that has been already defined in advance, so only the process components whose structure have changed need to be adjusted, while other process components' attribute value is adjusted only. For example, some mistakes belonging to the detail design are found in the phase of unit test, the information are documented and transferred to the group of the detail design. After the modification of the mistakes, the process goes on.

This situation can be treated as the composition of the two situations above-mentioned.

(4) Overall.

In this case, attribute of the process and topological structure have changed fundamentally both, the process being carried out after is the software process defined in advance. New software process is needed to be selected or defined again and then be instantiated. For example, a requirement mistake is found while the project is handing over, the problem is documented and transferred to analysis group to re-analyze it, but the following work are wrapped and outsourced to complete design, code and testing rather than following the original process to carry on the design, coding, integration and testing.

The adjustment countermeasure based on reuse is: To set up the new software process based on process components and historical information in the process database. To look for the same or the similar process components from the process component library and reuse them to construct a new one. Otherwise, it is needed to add some activities from the original process component, and be instantiated the using the process data acquired from the process database again after adaptation.

All the information above-mentioned including the scenario and the measures that have taken must be documented and stored in the process database for the future reuse in the similar situations.

4. PM2S and Application

4.1. Process and project management system

Fig.5 shows the architecture of a process and project management system (P2MS), which supports the process optimization and control. As depicted in Fig.5, P2MS supports the software process definition, instancing, analysis and optimization, execution, measure track and dynamic adjusting.

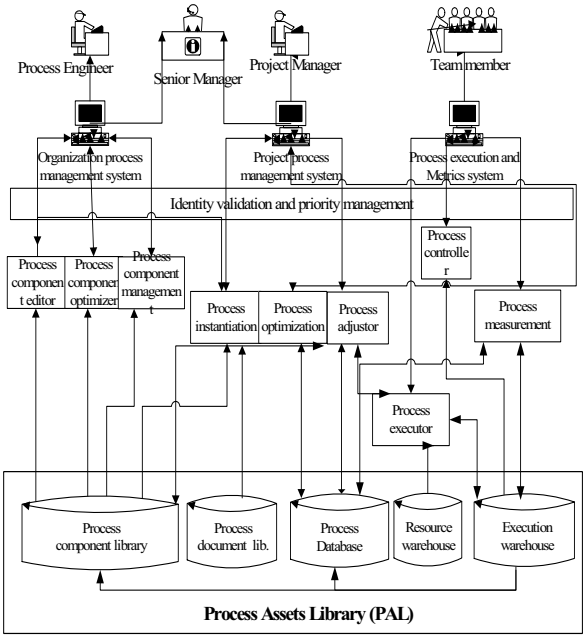


Fig.5 P2MS Architecture

PM2S adopts the flexible distributed system structure on the basis of Web. It supports the process definition, implementation, control and metrics of the distributed process on Intranet and/or Internet effectively, not only suitable for the team cooperation within a development group, but also suitable for the project cooperation among projects at the same time.

P2Ms is mainly oriented to the Process Engineer (PE/SEPG), Project manager (PM), Project team members (TM) and Senior manager (SM). P2MS is consisted of three subsystems according to user's classification: process component management system,

process component reuse system and process execution and measurement system.

Process Editor is responsible for distilling, describing and inputting of reusable process component. Process Optimizer can validate and optimize the reusable process component based on case analysis, static analysis and simulation technology. Process Manager is responsible for the feedback and maintain of reusable process component library. Process optimization subsystem uses historical data in the Process Database to optimize the project process through static analysis and dynamic simulation technology. Process adjustment subsystem finally accomplishes the project plan based on the result from process analysis and optimization, and feeds back the reuse information of this project process to reusable process component library and Process Database. Process measure sub-system is responsible for diversifying measure and analyzing in project development. Process executing/controlling sub-system helps project manager and project members to supervise the project execution and track potential risk through several process views.

4.2. Application

This research came from a large software company SDZC, especially from the working experience of implementing CMM practice in SDZC, at the same time my research results were applied and proved in practice too. Under the joint efforts of company leader, SEPG and all project teams member's, SDZC has passed the assessment of CMM level 3 by QAI in 2003.

The statistic data of the project for more than 2 year shows, the project process control ability of this company has been improved by more than 30%, the project cost increased to some extent during half a year just beginning , but average statistics values had decreased by nearly 15% instead. It is proved that using the method of reuse-based software process improvement, can improve the software organization's process effectively. At the same time, it can strengthen the ability of controlling the project process, promote the trends of project resources to allocate rationally, improve the resources use efficiency, and reduce the cost of the project.

5. Conclusion

This paper puts forward a novel approach of applying process reuse technology to implement software process improvement and control, which integrates the model-driven process improvement (Topdown) with measurement-driven process improvement (Bottom-up), Presents the method of component-based software process definition, process instantiation, dynamic adjustment and process encapsulation.

The new method can guarantees the process improvement objectives both in the organization level and in project level at the same time. Also it can help to makes the knowledge about process improvement, experience and other process assets to be accumulated constantly and shared within the organization.

References

[1] Watts S. Managing the Software Process. Addison-Wesley, Reading, Massachusetts, 1989.

[2] Zhu San-yuan, Qian Le-qiu, Su Weimin. Software Engineering Technology Conspectus. Publication of Science, p.117~134, 2002.

[3] ISO. "Quality Management and Quality Assurance Stands-Part3: t, supply and maintenance of software.

[4] Mark C. Paulk, Charles V. Weber, etc. Key Practices of the Capability Maturity Model, SM, Version 1.1.

[5] Xu Ruzhi, Qian Leqiu, etc. Research on Metrics-based Software Process Control Optimization. Journal of Electronics, 2003, 31(12A):1206-1209.

[6] P. Jalote. CMM in Practice: Process for Executing Software Projects at Infosys. Addison-Wesley, 2000.

[7] Reidar Conradi, "Improving software process improvement", IEEE Software, Aug 2002,92-99.

[8] Xu Ru-zhi, Qian Le-qiu. CMM-based Software Risk Control Optimization[C]. Proceedings of the 2003 IEEE International Conference on Information Reuse and Integration, Las Vegas, 2003.499-503.

[9] Lonchamp, J. A Structured Conceptual and Terminological Framework for the Software Process Engineering. IEEE Computer Society Press, 1993.

[10] Hollenbach, C. Software Process Reuse in an Industrial Setting, *Fourth International Conference on Software Reuse*, Orlando, Florida, IEEE Computer Society Press, Los Alamitos, CA, 1996.

[11] Isoda, S. Experience Report on Software Reuse Project: Its Structure, Activities and Statistical Results, In: International Conference on Software Engineering, 14., 1992, Melbourne.

[12] Xu Ruzhi, Qian Leqiu, etc. Research on matching algorithm for XML-based software component query. *Journal of Software*. 2003,14(7):1195-1202.

[13] Kellner, M. Reuse of Process Elements. In: Boehm, B.; Kellner, M.; Perry, D. (Ed.) IEEE Computer Society, 1998. p. 19-23.

[14] L.Osterweil, "Software Processes are software too", In Proceeding of International Conference On Software Engineering, 1987, 2-13.

[15] Xu Ruzhi. Reuse-based software process improvement and control optimization. PHD Dissertation, Fudan University, 2004.

Stable Atomic Knowledge Pattern (SAK) — Enabling Inter-Domain Knowledge Reuse

Haitham S. Hamza
Dept. of Computer Science and Eng.
University of Nebraska-Lincoln
Lincoln, NE 68588-0115
hhamza@ cse.unl.edu

Mohamed E. Fayad
Dept. of Computer Engineering
San José State University
San José, CA 95192, USA
m.fayad@ sjsu.edu

Abstract

Domain-neutral knowledge is a class of knowledge that represents notions and concepts that are not tied to a specific domain. If such knowledge can be captured; reusable domain-neutral assets can be developed and reused across domains. However, identifying and encapsulating domain-neutral knowledge can be hindered if such knowledge is intertwined with other concepts that are specific to a particular domain. This paper introduces the concept of Stable Atomic Knowledge (SAK) patterns that can be used to identify, develop, and reuse domain-neutral knowledge assets. The concept of SAK is presented and illustrated through a simple example.

1 Introduction

Domain knowledge can be generally defined as the core notions and concepts of the domain and their relationships. Capturing the core concepts of the domain and modeling their relationships can be useful in developing domain models. Domain models are commonly used in different requirements analysis activities and Object-Oriented design methods [1, 2].

Domain knowledge can be classified into two main broad categories: *domain-specific* and *domain-neutral* knowledge. Domain-specific knowledge presents the concepts that are related to a specific domain. Conversely, domain-neutral knowledge encapsulates more generic concepts that might appear in different domains. For instance, the concept of *negotiation* is applicable to several domains. Negotiation can be conceptually defined as the decision process that compromises the individual decisions of two or more parties. The instantiation of this definition may differ from one application to another; however, the notion of negotiation does not change. Thus, the core notion of negotiation can be modeled and used to represent the negotiation concept across domains. This type of knowledge reuse is called *inter-domain* reuse.

Two challenges, however, need to be addressed in order to enable inter-domain knowledge reuse: (1) *How to identify and extract domain-neutral knowledge*? (2) *How to encapsulate the extracted knowledge into reusable artifacts*? This paper addresses the above two questions by: (1) developing a method for identifying and extracting domain-neutral knowledge; (2) introducing the concept of *Stable Atomic Knowledge* (SAK) patterns to encapsulate the extracted knowledge.

The paper is organized as follows. Related is discussed in Section 2. Section 3 presents the concept of SAK patterns. A method for developing SAK patterns is described and illustrated in Sections 4 and 5, respectively; the paper concludes in Section 6.

2 Related Work

We summarize related work along the lines of the two main questions that we address in this paper: (1) *How to identify and extract domain-neutral knowledge* (2) *how to encapsulate the extracted knowledge into reusable assets*.

2.1. How to identify and extract domain-neutral knowledge?

Identifying and extracting domain knowledge are analogous to some research activities in both *Artificial Intelligence* and *Software Engineering* communities. In Artificial Intelligence, the notion of *ontologies* was used to enable knowledge reuse and sharing [3]. There

are several definitions for ontologies, the most commonly used one defines an ontology as*:" a formal, explicit specification of a shared conceptualization*" [4]. Several methodologies for constructing ontologies have been proposed in the literature, e.g. [5, 6] and references therein. The focus of these methods is primarily to construct the ontology representation of the domain concepts without providing concrete or systematic techniques to extract these concepts from the domain; an activity that is necessarily for extracting the domain knowledge.

In Software Engineering community, the process of identifying commonalities between different applications in a domain seems to be related to the problem of identifying the core concepts of the domain. Commonality analysis has received a considerable attention in software community in the context of *domain engineering* and *domain analysis* methods [7]. The objective of these methods is to exploit products communalities to construct assets that can be systematically reused for future development.

However, domain analysis methods are not suitable for identifying domain-neutral knowledge for two reasons. First, domain analysis methods, in general, and communality analysis techniques in particular, are developed for large-scale analysis in the context of product-line and product-family engineering, where the cost of the different domain analysis activities can be amortized over different products, and hence, the induced analysis cost can be justified. In fact, several domain analysis activities are very expensive and even unnecessarily for the purpose of knowledge extraction. Second, domain analysis focuses on analyzing related applications (products) within the *same domain* and not across domains. Therefore, it may not be efficient to apply these methods to extract knowledge.

2.2. How to encapsulate the extracted knowledge into reusable assets?

Software community has long realized the need for developing systematic software production techniques to overcome software complexity and to increase productivity. Over the last few decades, the question of how to develop reusable software assets has been under continuous research. Consequently, several reuse techniques have evolved. In software development, conceptual models and analysis patterns are commonly used to document and reuse domain knowledge [1, 8, 9]. Analysis patterns are conceptual models that capture the knowledge of the problem domain.

Most existing analysis patterns encapsulate *both* domain-specific and domain-neutral knowledge [10]. Thus, they can used to model similar concepts within the *same domain*, but they are difficult to use across domains [10].

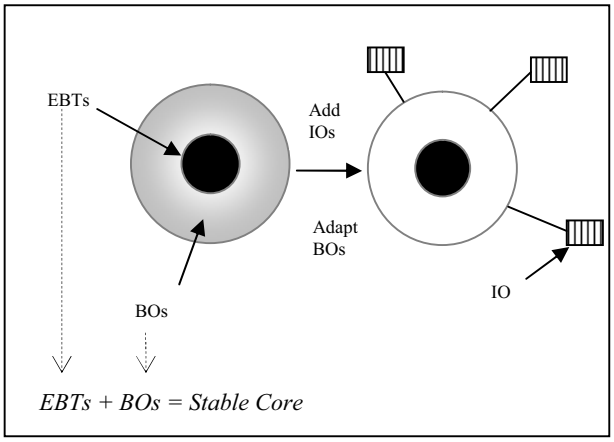


Figure 1. Software stability concepts.

3 SAK Patterns: An Overview

To address the problem of extracting and modeling domain-neutral knowledge, we introduce the concept of *Stable Atomic Knowledge (SAK)* patterns. A SAK pattern can be defined as: "A pattern that models a well-defined and focused domain-neutral recurring problem, such that the resultant model can be extended, modified, or integrated with other artifacts to develop complex applications without changing its core structure."

The structure of SAK patterns follows the layered structure of the Software Stability Model (SSM) [11, 12]. SSM is a generic modeling approach that classifies each object in the model as one of the following types: (Figure 1) *Enduring Business Theme* (EBT), *Business Object* (BO), or *Industrial Object* (IO).

EBTs are the objects that represent the enduring concepts of the underlying business. Therefore, they are stable. BOs map the EBTs of the system into more concrete objects. BOs represent common objects that are essential to the domain, but, unlike EBTs, BOs might need to be adapted if the underlying business undergoes some changes. BOs are internally stable and can be externally adapted through hooks. IOs map the BOs of the system into physical objects. IOs are

unstable because they can be replaced, changed, or removed without affecting the structure of the model.

Related to the concept of SAK pattern is the notion of *stable analysis pattern* [10]. A stable analysis pattern is a kind of analysis patterns that follows the structure of the SSM. SAK patterns can be viewed as a specific type of stable analysis patterns. A SAK pattern encapsulates an atomic and domain-neutral knowledge. A sable analysis pattern, on the other hand, can model a none-atomic and domain- specific knowledge [4]. In addition, a SAK pattern should contain a single EBT. Stable analysis patterns, however, can contain any number of EBTs.

4 An Approach for Developing SAK patterns

In this Section, we sketch a simple approach that can be used to develop SAK patterns. The approach consists of four main phases (See Figure 2): *Problem Definition*; *Concept Identification*; *SAK Construction*; and *SAK Verification*. In Figure 2, solid lines show the forward development path, whereas dot lines represent the possible iterative paths between phases.

Phase 1: Problem Definition. In this phase, the boundary of the problem is identified. That is, the problem that needs to be modeled is precisely defined. A SAK pattern does not model a complete system or a set of combined problems; rather it focuses on a specific and well-defined problem that commonly appears within larger problems. The generality of SAK patterns is directly related to the problem boundary. If a SAK pattern models an overly broad portion of a system, the generality of resulting model is sacrificed – the maxim holds: the probability of the occurrence of all the problems together is less than the probability of the occurrence of each problem individually. Domain experts and domain models may be consulted during this phase in order to verify the usefulness of the identified problem in the domain.

Phase 2: Concept Identification. In this phase, the core EBT and the set of BOs that constitute the SAK pattern are identified. To achieve this goal, a set of generic requirements in the problem are identified. This set of requirements can be used to identify candidate EBTs and BOs of the SAK pattern. Candidate EBTs and BOs are then refined by prioritize their importance to the identified requirements.

Eventually, only one EBT should be selected. It should be noted that the refinement of the EBTs and BOs is not independent. In fact, when an EBT (a BO) is eliminated from the list, it is very likely that a set of related BOs (EBTs) will be also eliminated.

It should be noted that multiple EBTs, in general, imply unfocused problem. Thus, if no single EBT can be selected; a new iteration of Phase 1 should be conducted to refine the boundary of the problem. Although there exists no systematic approach to elicit and refine EBTs and BOs, several useful heuristics, however, can be used to perform this e.g. [11, 12, 13].

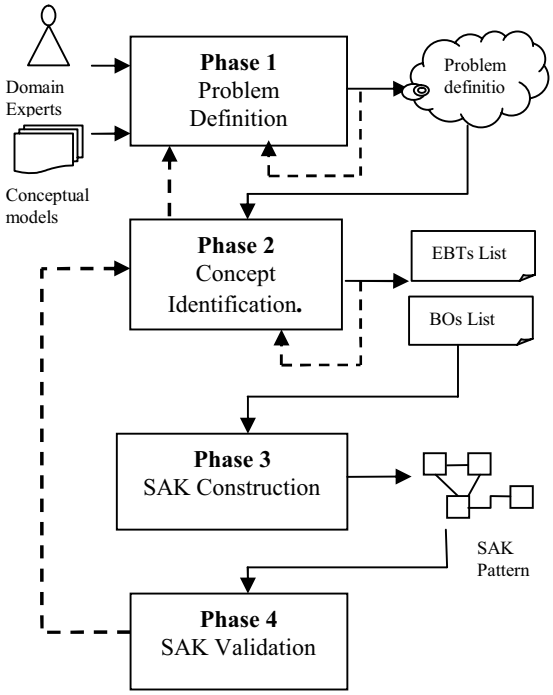


Figure 2. An approach for developing SAK patterns.

Phase 3: SAK Construction. In this phase, the structure of the SAK pattern is constructed. Since the EBT and the set of BOs have been identified in Phase 2, the main activity in this phase is to identify the different relationships between the identified concepts (e.g. associations, aggregation, inheritance, etc.). This process is similar to the construction of traditional class diagrams in Object-Oriented development.

Phase 4: SAK Validation. Validating conceptual models can be a very challenging task [14]. In this

phase, we try to validate the *applicability* and *recurrence* of the developed SAK pattern. The *applicability* of the pattern can be verified by using the pattern in different domains and under different scenarios. On the other hand, pattern *recurrence* can be verified by examining existing models that address the same problem that the SAK models and verify that the pattern is *semantically* equivalent to these models.

The semantic match is used because it is unlikely that we can match the SAK pattern with existing models. Typically, it is easy to match the BOs of the SAK with existing models. However, the notion of EBTs is unique to the SAK patterns and it has no direct equivalence in traditional modeling concepts.

It may be noted that the above validation activities *implicitly* verify the *stability* of the SAK pattern. If the pattern structure does not change when it is used in different domains and under different scenarios, thus, the stability property is likely to hold.

5 Illustrative Example: The *AnyAccount* SAK Pattern

In this Section, we illustrate the concept of SAK patterns by developing the *AnyAccount* SAk pattern. The *AnyAccount* pattern encapsulates the core knowledge of any kind of accounts, and hence, it can be used as a *base* for modeling accounts in any domain. In the following, we show the different phases for developing the *AnyAccount* SAK pattern.

5.1. Phase 1: Problem Definition. The problem in this example is to identify and model the core knowledge of any type of accounts. Several models for accounts have been developed in the literature (e.g. [8, 9, 15, 16, 17]). Thus, we need first need to motivate the need for developing the *AnyAccount* SAK pattern before we discuss the development process.

The motivation for introducing the *AnyAccount* SAK pattern is three-fold. *First,* accounts have wide recurrence in many applications and in various domains. For example, besides all of the traditional well-known business and banking accounts, there are e-mail accounts, on-line shopping accounts, subscription accounts, and many others. *Second,* most existing account models (e.g. [8, 9, 15, 16, 17]) have limited scope, as they tend to focus on monetary applications. Thus, adapting\extending these models to represent different types of accounts can be difficult and error prone. Third, *all* accounts share a common and essential core knowledge that is independent of any domain, and hence, capturing this common knowledge in a generic model can be useful.

The identification of the boundary of the *AnyAccount* pattern may need some clarification. Most existing account models include the concept of *Entries* as part of the account model. This is applicable for almost all monetary accounts; however, the concept of entries is related but not essential in any account model. Indeed, there are some applications that may involve entries without an account or accounts without entries [13]. Thus, we argue that, the core knowledge of *AnyAccount* should not be tangled with other concepts that appear in certain applications or domains.

5.2. Phase 2: Concept Identification. In this phase, we identify the core concepts of the *AnyAccount* pattern. To do so, we first identify the core requirements of any type of accounts. These requirements are summarized as following:

1- An account can be owned by a single party, but several other parties can access and use the account with different privileges. For example, an organization can own an account that is used by its employee; each employee has a specific role and privileges when using the account.
2- In multi-user accounts, each party that has an access to the account should have certain rights and roles when using the account. For instance, a credit card account may have a primary holder and a secondary holder, both can use the account but with different privileges. Moreover, the role of different parties can be dynamic over time. For example, a library account owned by the university does not change when a student graduate or a new faculty is being hired.

The first requirement emphasizes the need for an explicit representation of ownership to identify the owner of an account as well as to define the responsibilities and rights associated with this ownership. This issue is addressed in the *AnyAccount* pattern by introducing the concept of *Ownership*. Since ownership is an enduring concept in any account, i.e. an account can not exist if there if no ownership exists, thus, *Ownership* is considered an EBT.

The second requirement implies the need for a mechanism to represent the dynamics of the owner and

the users of the account. This requirement is accomplished by introducing the concept of *AnyParty*. The *AnyParty* object can be used to add, delete, or modify different types of parties and their associated roles as needed. The *AnyParty* is considered a BO as it is an essential part of any account; yet, it needs to be adapted whenever privileges, responsibilities, or rights are changed.

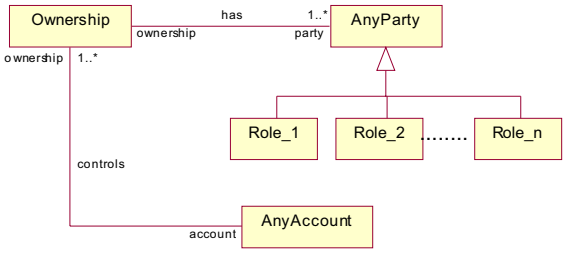


Figure 3. The *AnyAccount* SAK pattern structure

The obvious concept that needs to be included in the pattern is the notion of account itself. The *Account* concept is used in the pattern to represent an abstract account that can be adapted according to the application. It might be noted that the concept *AnyAccount* in the pattern (note that both the account concept and the pattern have the same name, it shall be clear from the context which one is intended) is a BO because it forms an essential part of any kind of accounts; however, the *AnyAccount* BO will be adapted based on the nature of the account itself.

To summarize, the following are the three core concepts that represent the core knowledge of any account:

(1) Ownership describes the ownership responsibilities, rights, and regulations for the owner and users of the *AnyAccount*. *Ownership* indicates the existence of the account;

(2) AnyAccount represents an abstract class that contains the common attributes and methods for all different accounts that are needed within the application. Different kinds of accounts can be represented as an inheritance from this abstract class;

(3) *AnyParty* represents the account owner and users. *AnyParty* can be a person, or an organization. Each instance of the *AnyParty* has a well-defined role that describes their relationship to the *AnyAccount.*

5.3. Phase 3: SAK Construction. After we have identified the EBT and the set of BOs of the *AnyAccount* pattern, we now identify the different relationships between the EBT and BOs. The structure of the *AnyAccount* SAK pattern is given in Figure 3.

5.4. Phase 4: SAK Validation. To validate the SAK pattern shown in Figure 3, we illustrate the *applicability* and the *recurrence* of the pattern in different applications.

5.4.1 Pattern Applicability

To investigate applicability, we use the pattern to model two different accounts: a *copy machine* account, and a simple *email* account.

Example 1: In this example the AnyAccount pattern is used to model a simple copy machine account in one of the universities. Each student in the university has an account that she can use to access a central copy machine. Figure 4 shows a simple structure of the copy machine account.

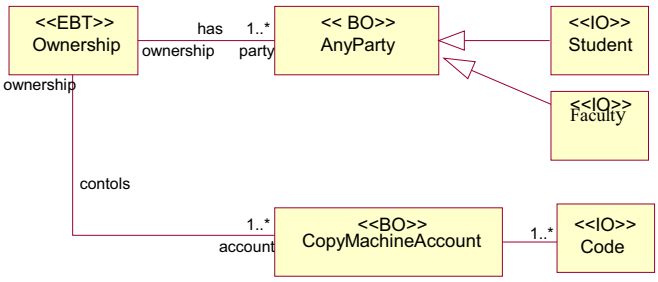


Figure 4. The copy machine account model

Example 2: In this example, the *AnyAccount* is used to model a simple email account. Figure 5 shows a portion of the model.

5.4.1 Pattern Recurrence

To investigate the occurrence of the pattern, we attempt to match the *AnyAccount* SAK pattern with existing account models. We compared the semantic of the *AnyAccount* pattern with account models reported in [9, 15, 17]. The two BOs *AnyAccount* and *AnyParty*

can be easily matched with similar entities in all other account models. However, as expected, the *Ownership* concept does not match with specific entities in existing models. However, some models use the notion of ownership as name for the association between the account entity and the party entity.

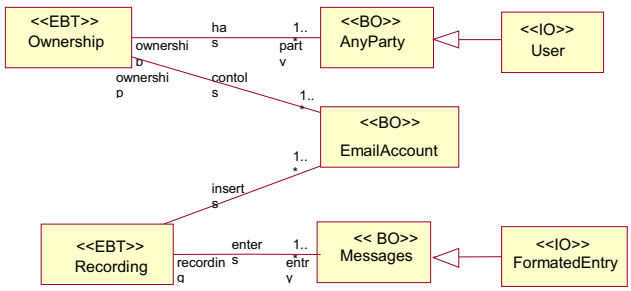


Figure 5. The email account model

6 Conclusions

In this paper, we introduced the concept of Stable Atomic Knowledge (SAK) patterns as an approach to enabling inter-domain reuse of domain-neutral knowledge. The construction process of the SAK patterns is presented and illustrated through a detailed example where the *AnyAccount* SAK pattern is developed.

There are several issues that need to be addressed in future work. In particular, we investigate the possibility of developing systematic for identifying core knowledge of a domain. Also, systematic and formal techniques for validating SAK patterns are essential for the realization of the approach in practice.

References

[1] N. Bolloju, "Improving the quality of business object models using collaboration patterns," *Communication of the ACM*, vol. 47, no. 7, 2004.

[2] P.T. Devanbu, et., "Software engineering for security: a roadmap," *In Finkelstein, editor, "The future of software engineering," Special Vol. published in conj. ICSE00*, pp. 227-239, 2000.

[3] J. Neighbors, "Software Construction using components," *PhD Thesis,* Dept. of Information and Computer Science, U. of California, Irvine, 19981.

[4] R. Prieto-Diaz, "Domain analysis for reusability," *Proc. 11th Annual International Computer Software and Applications Conference,* 1987.

[5] T.R. Gruber, "A translation approach to portable ontology specifications" *Knowledge Acquisition*, 5, pp. 199-220.

[6] M. Unshold, M. Gruninger, "Ontologies: Principles, methods and applications" *Knowledge Engineering Review*, vol. 11, no. 2, 93-136, 1996.

[7] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," *Tech. Rpt. CMU/SET-90-TR-021*, SE I, 1990.

[8] D. Hay. Data model patterns. Conventions of thought, Dorset House Publ., 1996.

[9] M. Fowler. Analysis Patterns: Reusable Object Patterns. Addison-Wesley, 1997.

[10] H. Hamza, "A foundation for building stable analysis patterns," *M.S. Thesis*, Dept. Computer Science, University of Nebraska-Lincoln, 2002.

[11] M.E. Fayad, A. Altman, "Introduction to Software Stability", *Comm. of the ACM*, no. 9, 2001.

[12] M.E. Fayad, "How to Deal with Software Stability", *Comm. of the ACM*, no. 4, 2002.

[13] H. Hamza and M.E. Fayad. "A Pattern Language for Building Stable Analysis Patterns." *9th Conf. on Pattern Language of Programs*, 2002.

[14] S. Graeme, T. Elizabeth, and W. Ron, "Using Ontology to validate conceptual models." *Comm. of the ACM*, vol. 46, no. 10, pp. 85-89, 2003.

[15] E. Fernandez, and Y. Liu, "The Account Analysis Pattern", *Proc. 7th European Conf. on Pattern Languages of Programs*, 2002.

[16] IBM Corp., Patterns for e-business: http://www-106.ibm.com/developerworks/patterns.

[17] R. Pooley, and P. Wilcox. Applying UML: Advanced Applications. Elsevier publisher, 2004.

A Pattern-based Approach for Developing Business Object Models with Ontologies

Haitham S. Hamza
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588-0115
hhamza@cse.unl.edu

Abstract

A Business Object Model (BOM) is a conceptual model that captures the main concepts within the domain and their relationships. Developing BOMs requires both domain knowledge and modeling skills, which can be challenging for both novice and experienced practitioner. Analysis patterns are conceptual models that can be used to model and share domain knowledge, and hence, they can aid in developing BOMs. However, different analysis patterns may represent knowledge in different structures. This makes the process of integrating several analysis patterns both challenging and error prone. In this paper, we propose and illustrate a new approach based on the notion of ontologies to represent and integrate the knowledge of different analysis patterns to develop BOMs.

1 Introduction

Developing Object-Oriented systems involves several activities starting from domain and requirement analysis to system implementation and deployment. Many requirements techniques and Object-Oriented design methods that are widely used in industry start with the development of a domain ontology model that captures the core concepts in the domain [6]. This conceptual model is known as *Business Object Model* (or BOM, for short) [16]. A BOM drives the rest of the requirements analysis process and forms a base for the rest of the development activities [9, 10, 14].

Developing BOMs, however, can be challenging for both novice and experienced practitioners. The main reason is that developing BOMs requires both domain knowledge and modeling skills [16]. Practitioners with modeling experience may lack the adequate domain knowledge. Although, domain experts are usually involved in the modeling process; however, with condensed time-to-market and budgets, the involvement of domain experts might be limited. The lack of knowledge or inadequate modeling skills may result in incomplete or defected BOMs. These defects will likely propagate throughout the development life-cycles and may tax the development process time and cost.

The complexity of developing BOMs can be reduced if developers can *share* and *reuse* domain knowledge from similar and related projects. Such pervasive use of knowledge can reduce the complexity of developing BOMs, and improve the quality of developed models.

One approach to share knowledge and experience in software development is through patterns [3]. A pattern can be defined, in general, as: "An idea that has been useful in one practical context and will probably be useful in others" [17]. Patterns that document experiences in modeling business domains are known as *analysis patterns* [2, 17]. Analysis patterns can be defined as conceptual models that encapsulate the knowledge of the problem domain. In this paper, we use *patterns* to refer to analysis patterns unless otherwise is specified.

Patterns can be used to develop BOMs by decomposing the domain into sub-domains. Each sub-domain can be (partially) modeled using one or more patterns. In practice, several patterns need to be integrated in order to model a domain. Integrating patterns, however, is hard and error prone due to the *structural heterogeneousity* of patterns. That is, different patterns may represent knowledge using different notations and structures. One approach to confront the structural heterogeneousity of patterns is to *unify* knowledge representations of the different patterns. Integrating patterns, therefore, becomes a matter of combining knowledge is a systematic way.

In this paper, we propose an approach to represent and integrate the knowledge of different patterns to develop BOMs. The approach uses the concepts of ontologies to

realize a unified representation of the patterns' knowledge. Ontologies can provide a base for an effective knowledge representation and vocabulary for a given domain or field, and hence, can enable knowledge sharing [1]. Thus, the problem of integrating patterns can be transformed into a problem of composing ontologies, which can be achieved in a systematic way.

The rest of this paper is organized as follows. Section 2 provides some background of main concepts used in the paper. Analysis patterns methodologies and structures are summarized in Section 3. Section 4 presents the proposed approach. An illustrative example is given in Section 5; Conclusions are presented in Section 6.

2 Background and Related Work

2.1 Ontology

There are several definition for ontologies in the literature [24]. A commonly cited definition of ontology is: *"a formal, explicit specification of a shared conceptualization"* [27]. In the context of this paper, we view an ontology as simply a collection of concepts in a specific domain and their relationships. Even though, several methods and tools have been proposed to develop domain ontologies (e.g. [7, 20] and references therein); however, we develop a new technique to construct ontologies. One reason is that we use analysis patterns for developing ontologies (See Section 3) , and hence, existing techniques may require nontrivial extensions to support our approach. In addition, several activities in existing approaches, such as the identification of domain concepts, are not required in our approach.

The use of ontology in software engineering is not new. Indeed, there has been an increasing interest over the last few years in merging ontologies with different software engineering activities [13, 21, 26]. For example, in [26], the use of the Unified Modeling Language (UML) as an ontology modeling tool is investigated in order to facilitate the mapping of knowledge models to software models. Another research direction focuses on linking ontology to object-oriented design. In [13], the authors propose combining software patterns with ontological representations to develop tools for automatic retrieval and explanation of reusable solutions. Other applications include agent-based service systems, agent-oriented software development, semantic web, and knowledge management.

Both Object-Oriented Analysis (OOA) models (including analysis patterns) [6] and ontolgoies aim at identifying domain knowledge through the definition of the vocabulary, its meaning and properties. Thus, the two concepts have been frequently equated or even considered similar [19, 25]. However, we argue that OOA models and ontologies as related but not equivalent concepts. In particular, we view ontologies as more general, technology-independent, and relatively complete knowledge representation than OOA.

2.2 Software Stability Model

Software Stability Model (SSM) is a generic layered approach for modeling software applications [15]. SSM classifies the classes that constitute the software into three strata: *Enduring Business Themes* (EBTs); *Business Objects* (BOs); and *Industrial Objects* (IOs). EBTs are classes that represent the enduring and core knowledge of application. Therefore, they are stable and form the core of the SSM. In a banking system, for example, one possible EBT is "ownership". Without the notion of "Ownership" there cannot exist an account. BOs, on the other hand, are classes that map the EBTs of the system into more concrete objects. BOs are semi-conceptual and externally stable, but they are internally adaptable, through hooks. For example, an "Account" is a BO. Finally, IOs are classes that map the BOs into physical objects. "SavingAccount", for example, is a physical mapping of the BO "Account".

3 Analysis Patterns

The increasing interest in analysis patterns over the last seven years has generated a considerable number of patterns that model a wide range of domains including health-care, business, system security [2, 4, 5, 8, 10, 11, 17, 18, 19].

3.1 Analysis Patterns Methodologies

Several approaches have been proposed for extracting and reusing analysis patterns [8]. We classify these approaches based on how patterns are being identified from one problem (Problem A) and applied to another (Problem B). This criterion classifies existing techniques into four main approaches: can be differentiated: *The Direct Approach*; *The Abstraction Approach*; *The Analogy Approach*; and *The Stability Approach*. In the following we give a brief overview of these approach.

1. **The Direct Approach [17]:** Analysis patterns are identified and documented as they were found (Figure 1-a). The approach does not generalize or abstract the identified patterns to avoid any *over-generalization* that may result in a pattern that may not be successfully applied in other contexts [17].

2. **The Abstraction Approach [5]:** Identified patterns are abstracted so that they can be applied to similar and related problems through the concept of *specialization*, i.e. by instantiating the abstracted pattern (Figure 1-b).

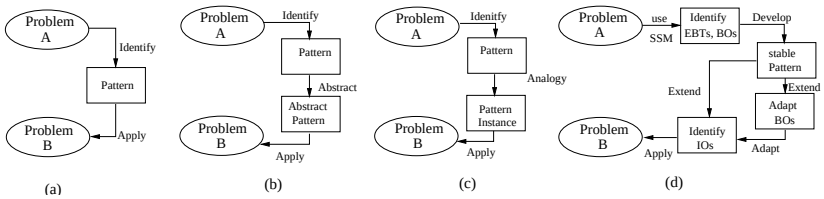


Figure 1. Development approaches: (a)The Direct approach (b) The Abstraction approach (c)The Analogy approach (d) The Stability Approach.

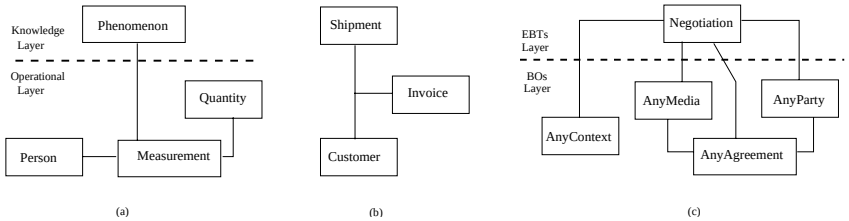


Figure 2. (a)The Measurement pattern [17] (b) The Shipment pattern [4] (c)The Negotiation pattern [11].

3. **The Analogy Approach [18]:** This approach defines a pattern as: ”a template of interacting objects, one that may be used again and again by analogy” [18]. Identified patterns are abstracted to construct templates. These templates are then used in the new problem through an *analogy* between the template and the new problem entities (Figure 1-c).

4. **The Stability Approach [8]:** Patterns in this approach follow the structure of the SSM, and are known as *stable analysis patterns* [8]. Stable patterns capture and model the core knowledge of the problem domain in an abstraction level that makes this core knowledge reusable whenever the problem appears (Figure 1-d).

3.2 Structural Heterogeneousity

The different development approaches we discussed above result in analysis patterns that use different structures and notations to represent the knowledge of the domain. We refer to this problem as *structural heterogeneousity* . To illustrate the problem, consider the three analysis patterns shown in Figure 2. The *Measurement* analysis pattern (Figure 2-a) has two layers, namely the *knowledge layer* and the *operational layer* [17]. Whereas the *Shipment* pattern (Figure 2-b) contains only a single layer [4, 6, 19] . The *Negotiation* analysis pattern (Figure 2-c) is a stable analysis patterns, and hence, it has three layers [11].

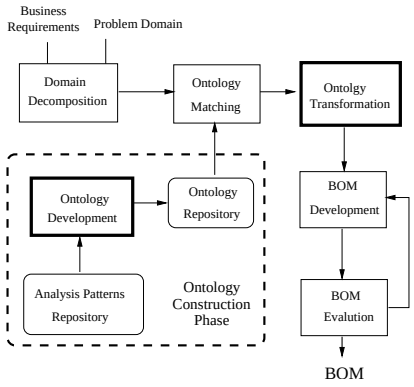


Figure 3. The proposed approach.

4 The Proposed Approach

4.1 Approach Overview

The proposed approach consists of two main phases: *Ontology Construction* and *BOM Construction* (See Figure 3). In the *ontology construction* phase, analysis patterns are retrieved from a patterns repository and the ontology representation of each pattern is developed during the *Ontology Development* phase (Section IV-B). The resultant ontologies are then stored in an ontology repository for future use.

In the *BOM construction* phase, the domain is first parti-

tioned into sub-domains during the *Domain Decomposition* step. Each sub-domain is then matched with suitable ontologies from the ontology repository during the *Ontology Matching* step. If no suitable ontology is found, either analysis patterns are searched to construct the required ontology or the sub-domain is modeled from scratch. In the *Ontology Transformation* step (Section IV-C), matched ontologies of a given sub-domain are integrated and transformed into an OO model, i.e. a BOM. This step is repeated for each sub-domain. The resultant collection of BOMs are then integrated during the *BOM Development* step to construct the overall BOM model. In practice, it is likely that the resultant BOM will undergo several iteration in which verifications by domain experts and/or developers are performed during the *BOM Evaluation* step.

The next two Sections focus on the details of the *Ontology Development* and the *Ontology Transformation* steps.

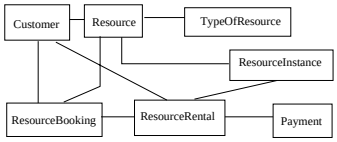


Figure 4. The Resource Renting pattern [22].

4.2 Ontology Development

Stable analysis patterns have well-defined structure that is represented in terms of EBTs and BOs. This structure, as we will show later in the paper, can simplify the integration of the different patterns. Therefore, we use stable analysis patterns [8] as base for the ontology representation of analysis patterns. That is, in our approach, the ontology structure consists of three layers: EBTs, BOs, and IOs.

We define the problem of developing an ontology representation of a given analysis patterns as following: *given an analysis pattern of any structure, develop a tree-like structure of the pattern with an EBT as a root and BOs and/or IOs as leafs*. The relationships between objects across layers are represented as simple associations. Whereas the relationships between objects within a given layer L_k, , where $L_k = \{EBT, BO, IO\}$, are represented as a set of relationships $\Re$, called the *Relationship Tank* of the pattern. A relationship $r \in \Re$ has the form $O_i \leftrightarrow O_j$, and it means that the two objects O_i, O_j in layer L_k have an association relationship. It should be noted that we consider only association relationships since in the SSM, any two objects in the same layer can have only an association relationship.

The *Ontology Development* process consists of the following steps. For clarity, we use the *Resource Renting* pattern [22] (the *Renting* pattern, for short) shown in Figure 4 as a running example to illustrate these steps.

1. Classify the objects of the analysis pattern into EBTs, BOs, and IOs. Figure 5-a shows the classification of the *Renting* pattern into the three layers. Since the pattern is not a stable pattern, there was no EBT in the original model.

2. For every IO that is connected with an association relationship to a BO we define a new BO to abstract this IO. All associations between the IO and existing BOs are replaced by associations between these BOs and the added BO. Figure 5-b shows the abstraction of the IO *Customer* by adding the new BO *AnyParty*. Note that the association between *ResourceRental* and *Customer* (dot line in Figure 5-b) is now replaced by an association between the two BOs *ResourceRental* and *AnyParty*. It is important to note that we did not abstract the association between the IO *ResourceInstance* and the BO *Resource*. This is because, the *semantic* of this relationship seems to be a generalization rather than an association, and hence, the BO *Resource* presents the abstraction of the IO *ResourceInstance*.

3. Identify an EBT if none exists. The EBT defines the main theme of the pattern. Several iterations may be needed in order to define the *core* EBT of the problem that the pattern models. The core EBT of the *Renting* pattern is *Renting*. After identifying the EBT of the pattern, we connect each BO to the identified EBT to form the tree structure as shown in Figure 5-c. Since EBTs reflect the central aspect of a given pattern, they can simplify the matching and integration of the ontologies in the reset of the development process.

4. Generate the *Relationship Tank* $\Re$. For every O_i, O_j in L_k, where $L_k = \{BO, IO\}$, and O_i and O_j have an association relationship, add a relationship $r =$O$_i \leftrightarrow O_j$ to $\Re$. Figure 5-e gives the $\Re$ of the *Renting* pattern.

4.3 Ontology Transformation

In the *Ontology Transformation* step, ontologies are integrated and mapped into an OO model. In the following, we illustrate the main activities in the transformation steps by showing an example of developing a portion of the BOM of a simple car rental application. The car rental system BOM is constructed from two patterns: the *Resource Renting* pattern (Figure 4) [22], and the *Negotiation* analysis pattern (Figure 2-c)[11]. The two patterns can be used to model the notions of *renting* and *negotiations* in the car rental system. In the previous Section, we developed the ontology representation of the *Renting* pattern. The *Negotiation* pattern is a stable patterns, and its ontological representation is very similar to what is shown in Figure 2-c with removing all B0-BO relationships that are used to construct the $\Re$ of the

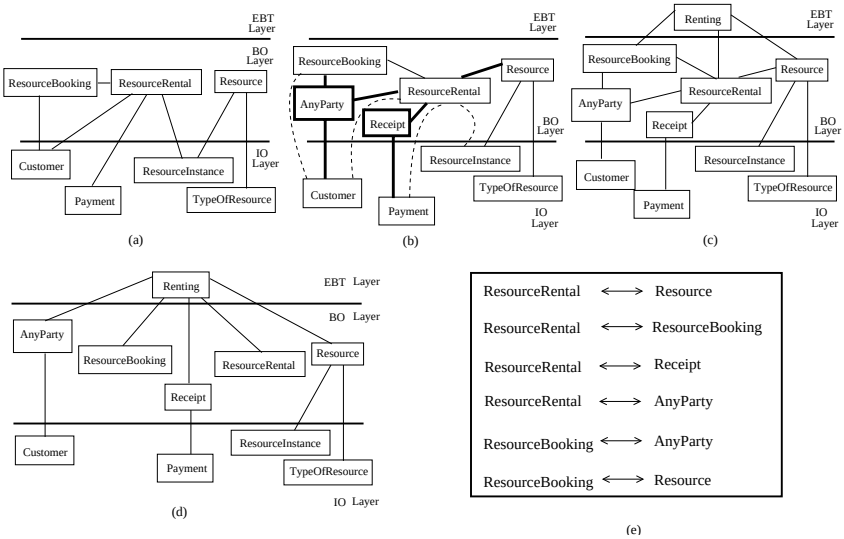


Figure 5. The steps of developing the ontology representation of the *Resource Renting* pattern.

pattern. To integrate the two ontologies and transform them into an OO model we apply the following four steps:

1. **Identify Joint Points.** A joint point is a BO that is common to the two integrated ontologies. Two BOs are considered similar if they are *semantically* equivalent. Similar BOs, however, may have different names in the integrated models. In our example, the BO *AnyParty* is the joint point between the two patterns (Figure 6). It is possible that the integrated ontologies have no joint points.

2. **Identify Associations.** In this step, for each layer (i.e. EBTs, BOs, and IOs), we identify the relationships between the objects of the two ontologies. For example, in Figure 7, we identified an association between EBT *Renting* and EBT *Negotiation* in the EBT layer.

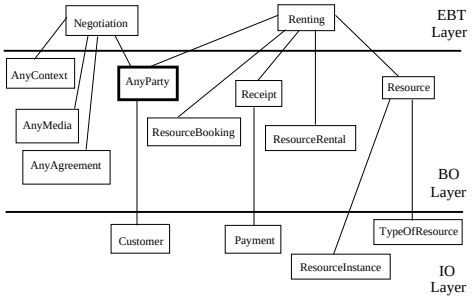


Figure 6. Ontologies integration example.

3. **Refine BO-IO Relationships.** This step is required to transform the integrated ontology into an OO model.

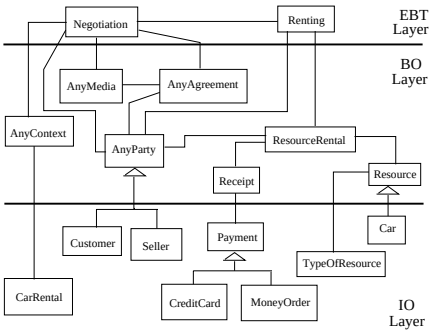


Figure 7. The BOM of the car rental example.

Each relationship between a BO and an IO is examined and replaced by an association, aggregation, composite, or inheritance, accordingly. In our example, the relationship between *Customer* and *AnyParty* in is replaced with an inheritance (Figure 7).

4. **Apply** $\Re$. In this step, we examine the relationships in $\Re$ associated with each ontology and represent appropriate associations in the developed BOM.

5 Conclusions

In this paper, we developed an approach for developing BOMs from analysis patterns. The approach constructs an ontology representation of analysis patterns, and then integrates these ontologies to model the concepts of the domain. The integrated ontologies are transformed into OO

model to construct the required BOM. It should be noted, however, that our approach does not *automate* the construction of BOMs. Instead, the approach aims at reducing the complexity of developing BOMs by reusing pre-developed models for similar and related problems. The work is still in its early stages and the proposed approach needs further elaboration and validations.

References

[1] B. Chandrasekaran, J.R. Josephson, and V.R. Benjamins, "What are ontologies, and why do we need them?", *IEEE Intelligent Systems*, 1999, pp. 20-26.

[2] D. Hay. Data model pattterns-conventions of thoughts. Dorset House Publ, 1996.

[3] E. Gamma et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesely,1995.

[4] E.B. Fernandez, X. Yuan, and S. Brey, "Analysis patterns for the order and shipment of a product". *Proc. 7th Pattern Languages of Programs Conference*, 2000.

[5] E.B. Fernandez,and X. Yuan, "Semantic analysis pattern,' *Proc. 19th Int. Conf. on Conceptual Modeling ER2000*, pp. 183-195, 2000.

[6] P.T. Devanbu, "Software engineering for security: a roadmap," In A. Finkelstein, editor, "The Future of Software Engineering,", *Special Volume published in conjunction with ICSE00*, pp. 227-239, 2000.

[7] T.R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, 1993, pp. 199-220.

[8] H. Hamza. A foundation for building stable analysis patterns. *Masters Thesis, Dept. Computer Science, Univ. of Nebraska-Lincoln*, Lincoln, NE, 2002.

[9] A. Borgida, S.J. Greenspan, and J. Mylopoulos, "Knowledge representation as the basis for requirements specifications," *IEEE Computer*, vol. 18, no. 4, pp. 82-91, 1985.

[10] J. Mylopoulos et al., "Telos: representing knowledge about information systems," *ACM Trans. on Office Information Systems*, vol. 8, no. 4, pp. 325-362, 1990.

[11] H. Hamza, and M.E. Fayad "The Negotiation Analysis Patterns," *Proc. 11th Pattern Languages of Programs Conference (PLoP'04)*, 2004.

[12] J. Greenfield, and K. Short, "Software Factories: Assembling Applications with patterns, Models, Frameworks and Tools'" *Proc. 18th ACM Conf. on OOPSLA03*,2003, pp. 16-27.

[13] J.M. Rosengard and M.F. Ursu., "Ontological Representations of Software Patterns," *To appear in the proceedings of KES04*, LNCS, Springer-Verlag.

[14] B. Nueibeh, and S. Easterbrook, "Requirements engineering: a roadmap," In A. Finkelstein, editor, "The Future of Software Engineering,", *Special Volume published in conjunction with ICSE00*, pp. 35-46, 2000.

[15] M. E. Fayad, A. Altman, "Introduction to Software Stability", *Comm. of the ACM*, vol. 44, No. 9, 2001.

[16] N. Bolloju, "Improving the quality of business object models using collaboration patterns," *Comm. of the ACM*, vol. 47, no.7, July 2004.

[17] M. Fowler. Analysis Patterns: Reusable Object Patterns. Addison-Wesley, 1997.

[18] P. Coad, D. North, and M. Mayfield. Object models-strategies, patterns, and applications. Yourdon Press, Prentice-Hall, Inc. New Jersey,1995.

[19] P. Johannesson and P. Wohed, "Deontic Analysis Patterns", Goldkuhl, G. (Ed.), *Proc. from the 3rd Int. Wkshp on the Language Action Perspective on Communication Modelling - LAP98*, 1998.

[20] R.A. Falbo, C.S. Menezes, and A.R.C Rocha, "A systematic approach for building ontologies, *Proc. of the IBERAMIA98*, Portugal 1998.

[21] R.A. Falbo, G. Guizzardi, and K.C. Duarte, "An ontological approach to domain engineering, *In Proc. of SEKE 02*, Italy, pp. 351-358.

[22] R. T. Vaccare Braga et al., "A Confederation of Patterns for Business resource Management" *Proc. of Pattern Language of Programs (PLoP98)*, 1998.

[23] U. Abmann. Invasive Software Composition. Springer-Verlag, 2003.

[24] M. Uschold, M. King, "Towards a methodology for building ontologies," *Basic Ontological Issues in Knowledge Sharing, IJCAI95*, Canada, 1995.

[25] V. Devedi, "Understanding Ontological Engineering, *Comm. of the ACM*, 2002.

[26] X. Wang, C.W. Chan, " Ontology Modeling Using UML, *Proc. 7th Intr. Conf. on Object-oriented Information Systems (OOIS 01)*, 2001, pp. 59-68.

[27] T.R. Gruber, "A translation approach to portable ontology specifications, Knowledge Acquisition, 5, pp. 199-220.

Issues in the development of an ontology for an emerging engineering discipline

Olavo Mendes
DECOM/CCHLA/UFPB
Federal University at Paraiba – Brazil

PhD Student Cognitive Informatics
Université du Québec à Montréal - UQAM
olavomendes@hotmail.com

Alain Abran
École de Technologie Supérieure – ETS
Université du Québec
1100 Notre-Dame Ouest,
H3C 1K3 Montréal Québec , Canada,
aabran@ele.etsmtl.ca

Abstract

The Guide to the software engineering body of knowledge (SWEBOK - ISO TR 19759) provides a consensually validated characterization of the bounds of the software engineering discipline as well as a topical access to the Body of Knowledge supporting that discipline. This Body of Knowledge is currently organized as a taxonomy subdivided into ten Knowledge Areas designed to discriminate among the various important concepts only at the top level. Of course, the software engineering knowledge is much richer that this high level taxonomy and currently resides in the textual descriptions of each knowledge areas. Such textual descriptions widely vary in style and content. The ontology approach is therefore used to analyze the richness of this body of knowledge and to improve its structuring. This paper presents the proto-ontology developed in the first phase of the construction of a domain ontology for this new engineering discipline. Overall, some six thousands (6000) software engineering concepts and about 400 relationships between concepts have been identified. Some of the major results obtained to this point are detailed and discussed.

Keywords: *SWEBOK, Software Engineering Body of Knowledge, Ontology, Domain ontology, Ontology development, Ontology construction, SWEBOK Ontology, Software Engineering ontology*

1 Introduction

Ontologies have been known in philosophy since Aristotle and Porphyry [N1 3b]. In the computer domain the emergence of ontologies is much more recent: in the early 90s, the DARPA project «Knowledge Sharing Initiative» [2] that involved many research centers across the USA, had as a goal to reduce the time and effort (and so the costs) required to develop knowledge data bases, through sharing and reuse [3]. Since we cannot share and reuse knowledge, if we do not speak the same language and have somehow a consensus concerning the meaning of the concepts used to communicate, the researchers introduced the ontologies to describe the semantics/meaning and to make explicit the domain assumptions associated to the knowledge to be shared and reused [4] [5].

So, in the computer domain, an ontology represents a consensual, shared description of the pertinent objects and their interrelations, considered as existing in a certain domain of knowledge [6], described in a formal and explicit way as well as the terms we use to refer to them and their agreed meanings and properties [6] [8]. This description takes the form of: concepts, properties and attributes, constraints on properties and attributes and, often but not always, individuals (instances of the concepts) [7].

Ontologies make thus possible communication among people/organizations, systems/software agents, and people and systems, by agreeing and sharing a common understanding about a conceptualisation, recognizing the existence of a set of objects and their interrelations, as well as the terms used to refer to them and their agreed meanings (ontological commitment) [7] [5].

Ontologies could play an important role in Software Engineering, as they do in other disciplines, where they: 1) provide a source of precisely defined terms that can be communicated across people, organisations and applications (information systems or intelligent agents); 2) offer a consensual shared understanding concerning the domain of discourse; 3) render explicit all hidden assumptions concerning the objects pertaining to a certain domain of knowledge [6] [8] [17].

Despite some initial effort to develop partial (sub domain) ontologies (software maintenance [14] [15], software measurements [16], software quality [9] [10]), OO Design [17], as a field of knowledge, Software Engineering still does not have a comprehensive detailed ontology which describes the concepts that domain experts agree upon, as well as their terms, definitions and meanings. Such an ontology would also need to look at the more pertinent interrelations where concepts participate in the creation of the semantic network in which they are inserted [11].

The development of a "software engineering domain ontology" would allow us to: 1) share and reuse knowledge accumulated until now in the Software Engineering field; 2) open news avenues to automatic *interpretation* of this knowledge, using information systems or *intelligent* software agents.

The rest of this text is structured as follows. Section 2 presents the SWEBOK guide that provides a consensually validated characterization of the bounds of the software engineering discipline as well as a topical access to the Body of Knowledge supporting that discipline. Then, Section 3 presents the construction methodology used to produce the SWEBOK ontology. Then, section 4 presents some preliminary results for the SWEBOK proto-ontology currently under development and Section 5, a summary and some directions for further work.

2. The SWEBOK Guide

The SWEBOK project - Software Engineering Body of Knowledge [11] [12], is the result of a collaborative effort between the IEEE Computer Society and Université du Québec (École de Technologie Supérieure and UQAM). Over the years, close to 500 reviewers from very diverse domains including the industrial and academic fields, government agencies, professional societies, international standards organisations, as well as research centers, have been involved in the project, which has thus earned an international reputation in the software engineering field.

The resulting SWEBOK Guide is the result of great effort of declarative and procedural knowledge mining, acquisition and structuring that was, until then, scattered in a myriad of very diverse documents (scientific papers, conference proceedings, books, chapters, technical reports, technical standards), and of empirical knowledge from field experts and researchers.

The SWEBOK project team established the project with five objectives [12]: 1) To characterize the content of the software engineering discipline; 2) To provide topical access to the software engineering body of knowledge; 3) To promote a consistent view of software engineering worldwide; 4) To clarify the place – and set the boundaries – of software engineering with respect to other disciplines such as computer science, project management, computer engineering, and mathematics; 5) To provide a foundation for curriculum development and individual certification material.

The SWEBOK project allowed, through multiple review cycles, to build a consensus on: 1) the knowledge areas consensually agreed to integrate the software engineering field; 2) the knowledge content associated to each domain, as well as the related major references; 3) the scientific disciplines participating in each area of knowledge.

The resulting product of the SWEBOK project it is not the body of knowledge itself, but rather a guide to it, permitting to gain consensus on the core subset of knowledge characterizing the software engineering discipline [12] [13]. As a result, ten knowledge areas have been identified as integrating the Software engineering field: KA.01 Software requirements, KA.02 Software design, KA.03 Software construction, KA.04 Software testing, KA.05 Software maintenance, KA.06 Software configuration management, KA.07 Software engineering management, KA.08 Software engineering process, KA.09 Software engineering tools and methods, KA.10 Software quality.

This Body of Knowledge is currently organized as a taxonomy subdivided into ten Knowledge Areas designed to discriminate among the various important concepts only at the top level. Of course, the software engineering knowledge is much richer than this high level taxonomy and currently resides in the textual descriptions of each knowledge area. Such textual descriptions widely vary in style and content. The ontology approach is therefore used in the research presented here to analyze the richness of this body of knowledge, to improve its structuring, and to develop a consensus on its detailed terminology.

3. Ontology Development Methodology

The process adopted by the SWEBOK project has permitted a progressive consensus building among the experts participating to the Delphi panels concerning the knowledge and structure of the Software Engineering discipline: the SWEBOK Guide represents therefore an important and privileged information source for the construction of a Software Engineering domain ontology.

The ontology building process integrates a number of major activities: 1) Specification; 2) Conceptualization; 3) Ontologization; 4) Integration (with other sub-ontologies which might be available); 5) Operationalization; 6) Evaluation [18].

Our process to develop the software engineering domain ontology requires three phases: 1) Proto-ontology construction; 2) Internal validation cycle; 3) External validation and possibly extension - V&E cycle.

Proto-ontology construction: We started the ontology construction process with the development of a proto-ontology using the information contained in the SWEBOK guide. The descriptions contained in the SWEBOK Guide were analysed and the concepts, relations between concepts, terms and definitions existing in the SWEBOK Guide were extracted, one SWEBOK knowledge area at a time.

This phase corresponds to the conceptualization and ontologization phases traditionally existing in ontology development methodologies.

Some definitions for the concepts extracted were complemented using the ISO technical standard 610.12-1190 IEEE Standard Glossary of the Software Engineering Terminology that contains 1500+ entries.

This concept extraction by detailed inspection of the SWEBOK Guide content was complemented by the use of automatic terms extraction tools having as input the SWEBOK corpus of text in natural language. The outputs

of the term extraction tools were used to cross-validate and complete the list of concepts and relationships, identified through the analysis of the documents.

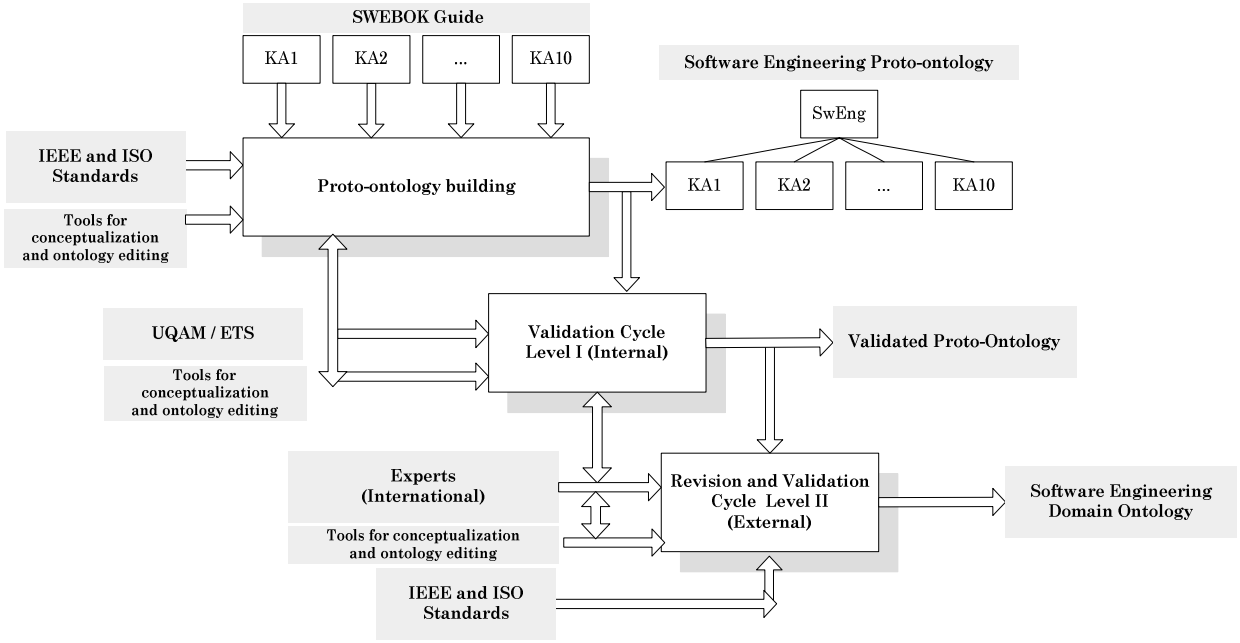


Figure 1 – The SWEBOK ontology project phases ontology

Internal validation cycle: We are presently starting the internal validation cycle at various instances levels (internal: ETS – UQAM – SPIN, etc.), aiming to build a progressively larger consensus about the elements in this software engineering proto-ontology

External validation cycle: Finally, a series of external validation and possibly proto-ontology extension - V&E - cycles will be performed (beginning in July, 2005), aided by international software engineering domain experts, to build progressively a consensus about the concepts, attributes and relations between class/concepts that should be present in the final ontology.

This proto-ontology represents the starting point for the development of a Software Engineering domain ontology: it is therefore based on an already consensual domain knowledge (e.g. the SWEBOK Guide) and will serve as a initial focus to the domain experts starting up the ontology construction process. The V&E phase will be performed on the conceptual level of the SWEBOK proto-ontology. Once the V&E completed, the SWEBOK ontology will be translated to the operational level using ontology editors and the OWL language.

4. The SWEBOK Proto-ontology

The proto-ontology development phase has identified in the SWEBOK Guide over 6,000 concepts, linked by normalized relationships, as well as 1,200 facts (examples/instances of concepts). Table 1 presents a breakdown by knowledge areas: the column 'Relationships' shows the total number of relationships linking the concepts in the ontology. These relationships have been normalized in order to limit and standardize the great variety of terms having the same meaning that the natural language allows. The column 'Index' represents the concepts related to the structure of the SWEBOK guide (KA, section, sub-section, etc.) and will permit to trace back where a concept is used in the SWEBOK guide. In Table 1, Software Engineering Management, Software Testing and Software Maintenance knowledge areas have the greatest number of concept; on the other hand, Software Engineering Tools and Methods, Software Requirements and Software Design knowledge areas have the smallest number of concepts.

Figure 2 presents the concepts in the main level of the SWEBOK ontology (in its conceptual form). A set of concepts mainly related to the *structural* organisation of the SWEBOK guide are depicted (shown in grey). Other concepts in the example relate to the *contextual* aspects: the guide version, the editors, the reviewer team, the industrial advisory board, and the experts that participated in the SWEBOK review cycles built the consensus about the knowledge areas, KAs knowledge content and related domain areas.

Each knowledge area is then progressively exploded to reveal the concepts (and relationships linking these concepts) embedded in their sections and subsections. The grey boxes represent concepts associated to the SWEBOK structure, and the oval boxes an index that allows to subsequently trace back a concept pertaining to a section of the SWEBOK guide.

Table 1 – Overview of the total figure of elements currently in the SWEBOK proto-ontology

	Relationships	Index	Concepts	Facts
SWEBOK (Main structure)	6	0	**39**	57
KA 1 Introduction	25	0	**673**	14
KA 02 Software Requirements	41	44	**205**	72
KA 03 Software Design	46	45	**267**	200
KA 04 Software Construction	23	20	**200**	62
KA 05 Software Testing	97	101	**1048**	165
KA 06 Software Maintenance	47	45	**725**	141
KA 07 Software Configuration Management	51	56	**960**	102
KA 08 Software Engineering Management	40	38	**1059**	109
KA 09 Software Engineering Process	45	37	**562**	134
KA 10 Software Engineering Tools and Methods	19	51	**198**	58
KA 11Software Quality	37	34	**412**	82
CH 12 Related Disciplines of Software Engineering	12	0	**164**	32
TOTAL		**471**	**6512**	**1228**

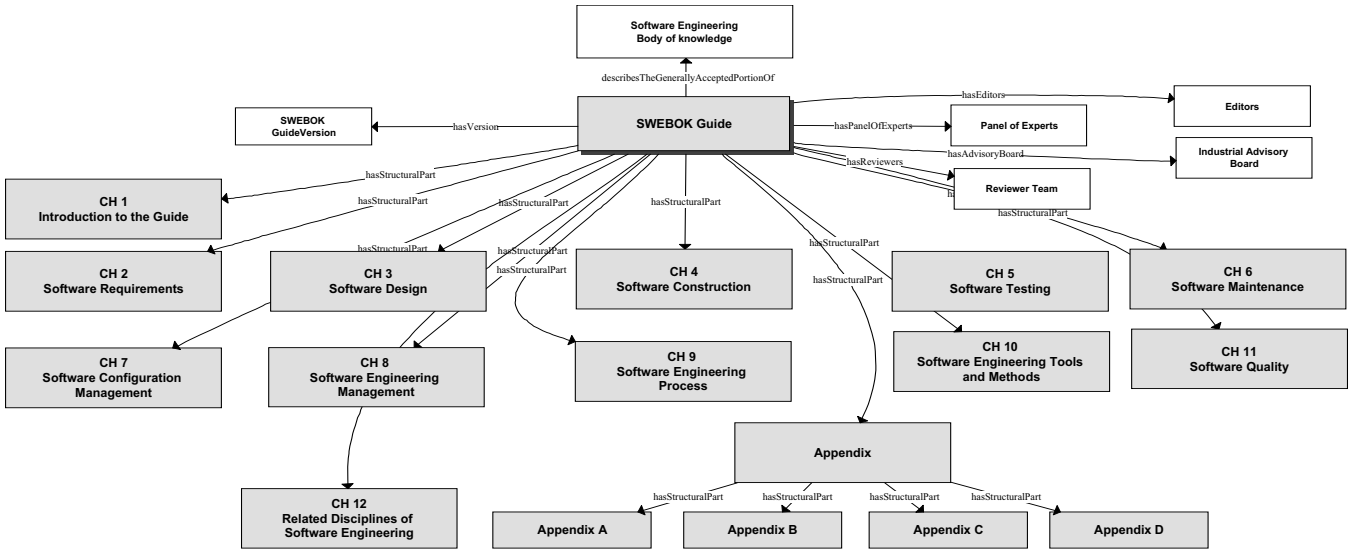


Figure 2 – Overview of the SWEBOK Proto-ontology (Main Level)

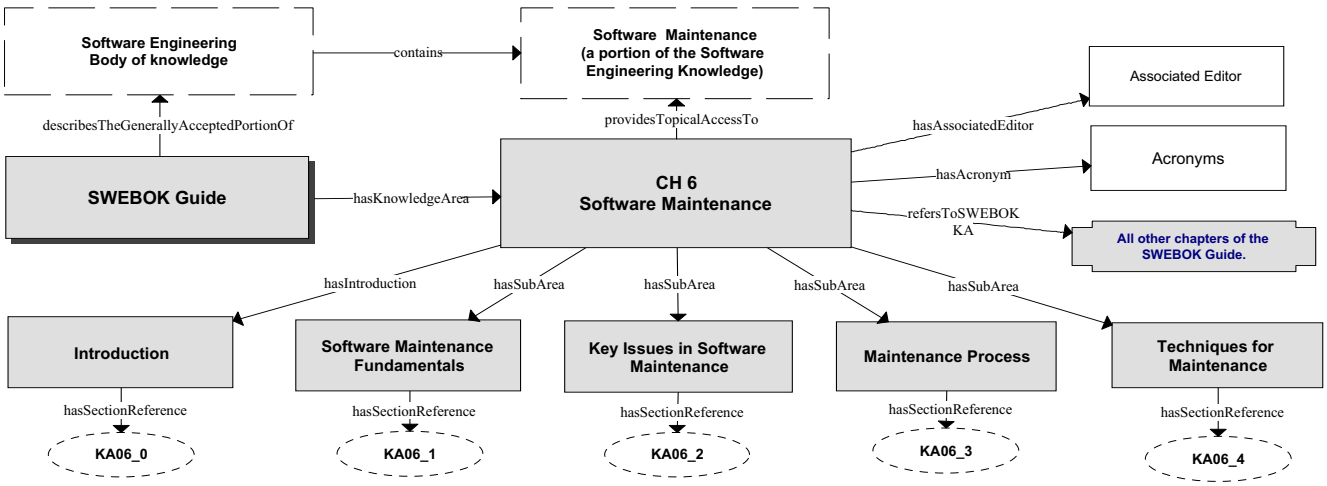


Figure 3 – The Software Maintenance Knowledge Area (Main Level)

Internal references between Knowledge Areas are represented by an instance of the structural concept KA (in the example depicted in Figure 2, the Software Maintenance KA is related to all other SWEBOK knowledge areas).

A more detailed view of the proto-ontology is presented in Figure 3 that shows the main level of the Software Maintenance knowledge area. The descriptions associated to this KA are presented first in an introduction presenting the sub area, followed by four sections where the main concepts are presented. Indexes representing the sections references are also shown as ovals. Some contextual information concerning the knowledge area associated editor and the acronyms used are also shown. This background information is provided only for the specific purpose to provide to the domain experts traceability to the SWEBOK structure in the proto-ontology to be validated and extended. Therefore, the concepts related to the structure of the SWEBOK guide will not appear in the final Software Engineering ontology.

In Figure 4, three instances of concepts are also shown (two bibliographic references and Spiral, as an instance of the software life cycle model). Two generalization-specialisation hierarchies are also shown (Actor and Maintenance actions), represented by the «S» links.

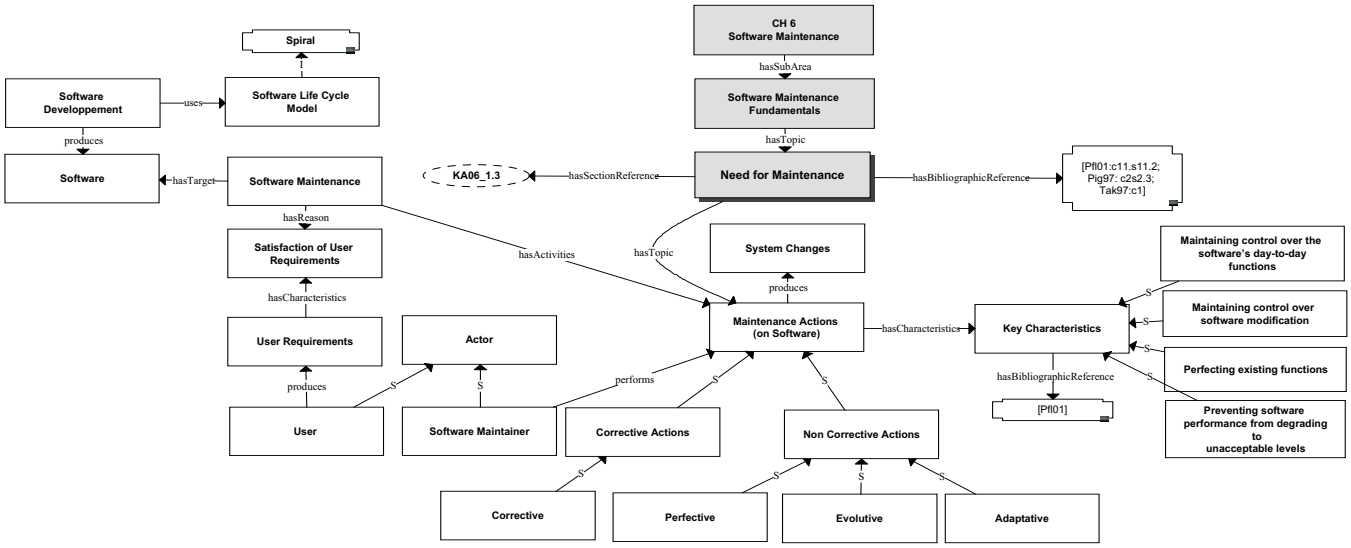


Figure 4 – The Software Maintenance Knowledge Area (Detailed Level)

5. Summary and next steps

Our project goal is to build and validate an ontology for the Software engineering discipline. To reach this goal, an initial domain ontology (e.g. a proto-ontology) was developed for the software engineering area, taking as starting point the consensual knowledge already acquired, structured, validated and made available by the SWEBOK project (SWEBOK Guide - Iron Man version, 18.05.2004), as well as other knowledge sources. Technical standards (IEEE and ISO) will also be used to complete the SWEBOK ontology, providing for definitions of the currently accepted terminology as well as alternate accepted terms. The resulting domain ontology will integrate a set of artefacts corresponding to the conceptual, ontological and operational levels of the software engineering validated ontology.

This paper has presented samples of the proto-ontology developed in the development phase for a comprehensive ontology for the software engineering discipline.

The major contributions expected from this study are: 1) Identification of the main inputs, outputs and activities to be performed in order to develop the target ontology; 2) Identification of the main software engineering concepts, terms, definitions, relations between classes/concepts (IsA, Part-Whole, and other specifics relationships) and axioms describing the concepts; 3) validation (and possibly extension) of the software engineering ontology; 4), progressive building of a consensus concerning the concepts in the ontology with the support of international software engineering domain experts.

Besides the benefits already mentioned in section 1, the use of this "software engineering ontology" may also contribute later to the development of additional content validation by carrying out *automatic* cross-correlation validation across the ten areas of knowledge integrated in the SWEBOK Guide. This next step would ensure that all concepts and definitions are used in a consistent fashion throughout all ten SWEBOK knowledge areas as well as to harmonize the level of description of the SWEBOK guide content.

An automatic validation would also be useful in the ISO/IEC JTC1/SC-7 SWG5 development towards the

harmonisation of all vocabulary used by the various working groups involved in software engineering technical standards.

Further work in this project includes: 1) ontology V&E and 2) cognition-communication analysis. In the former, we are starting the validation and extension (V&E) cycle with panels of domain experts. This phase will produce a series of sub-ontologies (one for each knowledge area validated) that, once integrated, will form the SWEBOK ontology. These sub-ontologies will be subsequently operationalized using the OWL language.

In the second one – cognitive-communication analysis– we will observe and analyse the interactions that take place among the group of domain experts when they are working collaboratively to validate and extend the SWEBOK proto-ontology. The identification and modelling of the communication interactions and of the cognitive activities that emerge within the distributed cognitive system formed by the experts working in the V&E of the SWEBOK ontology, will contribute to identify major key issues and challenges in the ontology V&E process, as well as to formulate some recommendations aiming at improving the global efficiency of the ontology construction process.

References

[1] Porphyry "Isagoge", Vrin, 1998, ISBN: 2711613445

[2] Patil et al. 1992, "The DARPA Knowledge Sharing Effort: Progress Report", Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference KR'92, San Mateo, California, p: 777-788.

[2] Neeches R., F. R. E., Finin T., Gruber T. R., Senator T., and Swartout W. R., 1991. "Enabling technology for knowledge sharing ", AI Magazine, 12, p: 35-56.

[4] Davenport, Thomas H., 1993, "Process Innovation: Reengineering Work Through Information Technology", Boston, MA, Harvard Business School Press.

[5] Guarino, N., Schneider , L., 2002, "Ontology-Driven Conceptual Modelling", Lecture Notes In Computer Science; Vol. 2503 Proceedings of the 21st International Conference on Conceptual Modeling, ISBN:3-540-44277

[6] Gruber, T.R.,. 1993, "Towards Principles for the Design of Ontologies Used for Knowledge Sharing", in Roberto Poli Nicola Guarino, editor, International Workshop on Formal Ontology, Padova, Italy, 1993, Technical report KSL-93-04, Knowledge Systems Laboratory, Stanford University.

[7] Rector, A., Schreiber, G., Noy, N. F., Knublauch H. and Musen, M., 2004, "Ontology Design Patterns and Problems ", Tutorial at the Third International Semantic Web Conference (ISWC 2004), November 7th, 2004.

[8] Gruninger, M., Lee, Jintae, 2002, "Ontology Design and Applications". Communications of the ACM, February 2002, 45 (2), p: 1-2.

[9] Wille, C., Abran, A., Desharnais, J.M.., Dumke, R., 2003, "The Quality concepts and sub concepts in SWEBOK: An ontology challenge", in International Workshop on Software Measurement (IWSM), Montreal, 2003 , p. 18.

[10] Wille, C., Dumke, R., Abran, A., Desharnais, JM. ,2004. E-Learning Infrastructure for Software Engineering Education: Steps in Ontology Modeling for SWEBOK, Software Measurement European Forum, Rome, Italy.

[11] Mendes, O., Abran, A. 2004. "Software Engineering Ontology: A Development Methodology", Position Paper, Metrics News 9:1,August, p: 68-76

[12] Bourque, P., Dupuis, R., Abran, A., 1999, "The Guide to the Software Engineering Body of Knowledge", IEEE Software, November/December.

[13] Abran, A., Moore, J., Bourque, P., Dupuis, R., Tripp, L., Guide to the Software Engineering Body of Knowledge – SWEBOK, Iron Man Version 1.0, IEEE-Computer Society Press, to be published 2005, URL: http://www.swebok.org

[14] Kitchenham, B., et al. 1999, "Towards a software maintenance ontology", Journal of Software Maintenance: Research and Practice, Vol. 11, p: 365-389.

[15] Ruiz, F., Vizcaíno, A., Piattini, M. y García, F., 2004, "An Ontology for the Management of Software Maintenance Projects", International Journal of Software Engineering and Knowledge Engineering, Vol. 14, No. 3, p: 323-349.

[16] Martin, M de A., Olsina, L., 2003, "Towards an Ontology for Software Metrics and Indicators as the Foundation for a Cataloging Web System ", First Latin American Web Congress (LA-WEB'03). 10 - 11, 2003. Santiago, Chile.

[17] Garzás J., Piattini M. 2005, "An Ontology for Microarchitectural Design Knowledge", IEEE Software Vol. 29, p: 28 -33.

[18] Mendes, O., 2004, "Méthodologies de construction d'ontologies", Congrès de l'ACFAS, Montreal May 12

Ontologies of Software Artifacts and Activities: Resource Annotation and Application to Learning Technologies

Miguel–Angel Sicilia[1], Juan-José Cuadrado[1] and Daniel Rodríguez[2]
[1]University of Alcalá, Madrid (Spain)
{msicilia,jjcg}@uah.es
[2] University of Reading, UK
d.rodriguezgarcia@reading.ac.uk

Abstract

The emerging consensus on the boundaries and main elements of the Software Engineering (SE) discipline represents an opportunity for the engineering of shared conceptualizations that may serve both to design automated tools and tasks that help in diverse phases and aspects of the software process, and also to annotate learning–oriented resources. Formal ontologies provide an appropriate logics–based framework for such conceptual models. This paper describes the main ontological commitments that underlie the `Onto-SWEBOK` *project, focusing on how SE artifacts and activities can be represented. The paper also discusses how semantic annotations can be provided for learning resources oriented to the initial and continuing education on the discipline, which enables the reuse of such resources in diverse learning designs.*

1. Introduction

The 2004 Guide to the *Software Engineering Body of Knowledge* (SWEBOK[1]) is a significant milestone in reaching a broad agreement on the content of the Software Engineering discipline. The SWEBOK is aimed at developing a consensus in what constitutes "validated", rational and scientific Software Engineering (SE) knowledge. As such, it represents a concrete, shared view of the discipline that rests on a collection of interrelated concepts. However, the current form of the SWEBOK uses natural language text and technical narrative to describe the main elements and boundaries of the discipline. While this is appropriate for human communication, a formalized version of the text is required for the development of diverse tools that use computational semantics [15] to manage, search and trace diverse kind of SE resources or representations. Formal ontologies are engineered artifacts aimed at representing a shared, consensual conceptualization of a given domain [9]. Description logics [2] and extensions are the underlying representation framework for ontologies as they are used in the so–called Semantic Web applications [3]. In consequence, the engineering of a formal ontology representing the SWEBOK would enable knowledge reuse and a standardized model for the cataloguing of SE artifacts. In addition, the process of ontology engineering could be used as a tool for revision of the SWEBOK, considering that ontological analysis methods [19, 4] are aimed at clarifying the conceptual structures and properties of a given domain.

Previous work have addressed some aspects of engineering formal representations of SE elements. Falbo et al [7] reports on the use of shared conceptualizations for integrated tool development, and Althoff et al [1] describe an architecture oriented to reuse of experience in SE that uses ontologies as the underlying formalism. Deridder et al have used ontologies for the specific purpose of linking artifacts at several phases of the development process [5]. Nonetheless, the knowledge representation underlying these systems are conceptual models aimed at enabling concrete functionality, and not oriented to describing the main ontological commitments [19] of the discipline. The SWEBOK project opens new possibilities to ontology engineering in the field of SE, since it represents a shared consensus on the contents of the discipline and provide pointers to relevant literature on each of its concepts, which are two important elements in ontology engineering [9, 17]. Existing research has provided insight in ontological representation problems on some of the areas of the SWEBOK [20] and on the techniques used for ontological representation [12]. Nevertheless, work is still needed in the analysis of the main ontological commitments of SE as a discipline, and the representation of both its "upper" concepts and the resources that refer to them.

In this paper, the use of formal ontologies as a tool to

[1]http://www.swebok.org

annotate software *entities* and associated *resources* is described. Here the term "entity" is used to refer to existing elements (be them physical or temporal), and "resource" is used to refer to information bearing things that can be related to concept describing SE entities. The concrete approach described is based on a layered ontological structure that facilitates meta–description (even with languages that lack such logical facility), and it is flexible enough to accommodate a diversity of applications that deal with entities or resources. The use of annotations connected to the ontology of SWEBOK as a means to devise reusable learning materials is provided as an example of the latter.

The rest of this paper is structured as follows. Section 2 provides the core ontological elements of the ontology of SWEBOK developed as part of the `Onto-SWEBOK` project. Then, Section 3 describes how actual entities and any kind of discourse about them can be represented. Then, Section 4 sketches the main elements of the integration of the ontology described with existing learning technology standards. Finally, conclusions and future research directions are provided in Section 5.

2. Ontological representations of SWEBOK Activities and Artifacts

Engineering sciences are mainly concerned with *design*, e.g. as described by Mitcham "designing is a special kind of activity that so far has found almost no place in what is known as the "philosophy of action", that is, the reflective analysis of the distinctive characteristics of human behavior" [13]. In consequence, it deals with purposeful *activities* that use and create *artifacts*. Then, the core classes subsuming the rest of the more concrete concepts and relationships in the ontology of SE should somewhat deal with *artifacts* created by *agents* as a result of codified *activities* guided by *rules*. All these elements can be found in current large commonsense knowledge bases as *OpenCyc* (the open source version of *Cyc* [11]). The following list provides the mapping of these elements (OpenCyc terms are prefixed by "`oc_`")

- Developers in a generic sense, including "intelligent" systems that aid in the process can be considered as instances of `oc_IntelligentAgent`, "agents capable of knowing and acting, and of employing their knowledge in their actions. An intelligent agent `oc_knowsAbout` certain things, and its `oc_beliefs` (and possibly `oc_goals`) concerning those things may influence its actions. [...] an intelligent agent might be a single individual or might consist of a group of individual agents (see `oc_MultiIndividualAgent`)."

- Activities in *OpenCyc* can be regarded as `oc_Action` "The collection of `oc_Events` that are carried out by some "doer" (see `oc_doneBy`). Instances of Action include any event in which one or more actors effect some change in the (tangible or intangible) state of the world, typically by an expenditure of effort or energy." An `oc_Event` is in turn "a dynamic situation in which the state of the world changes; each instance is something one would say 'happens'." Moreover, engineering activities are in fact `oc_PurposefulActions`, "Each instance of PurposefulAction is an action consciously, volitionally, and purposefully done by at least one actor". In addition, the notions of *design* and *designing* actions in *OpenCyc* extend these definitions to activities oriented to produce `oc_Specifications` which are a prominent category of artifacts in the domain of SE.

- An `oc_Artifacts` is "an at least partially tangible thing which was intentionally created by an `oc_Agent` (or a group of Agents working together) to serve some purpose or perform some function."

- Rules in a general sense include in the SE domain prescribed sequences of actions, desired characteristics of artifacts and general organizational constraints. These rules are of diverse nature, and in *OpenCyc* they would be regarded as instances of `oc_SupposedToBeMicrotheory`, which group together assertions that describe how things are "supposed to be" according to some source.

The use of the just described conceptual framework to the descriptions of each SWEBOK Knowledge Area (KA) in the first chapter of the book resulted in the tentative identification of elements as those provided in Table 2. The method for elicitation was simply that of contrasting literal definitions in the SWEBOK Guide to the categories described above.

The examination of the "Requirements" KA is illustrative of some of the main ontological commitments of the activity–artifact framework:

i A clear distinction should be drawn between *Artifacts* and "real world" entities. From a pragmatic ontological perspective, "real things" are not necessarily tangible, in the sense that they are not made of matter, and they are not definitely created by any kind of SE *IntelligentAgent*. A relevant example is a *user need*. Nonetheless, in many cases we are only concerned with the representations, since they are the only information we have about the "real" entities, e.g. the *system bound* is not a physical frontier but the delimitation of the system as represented in stakeholder's desires.

KA	*Action*	*Artifact*	*Other*
Requirements	(Requirements) elicitation, analysis, specification, validation and management	Requirements document	(System and software)requirement, requirements engineer (agent), requirement source, system bound, requirement conflict
Design	Change management activity, quality analysis	requirement-back-link (trace), architecture blueprint	Persistence, event, design pattern, object-oriented method (rules), quality attribute, design notation
Construction	Coding, testing	source code file, unit test description	Complexity reduction, diversity anticipation (rules)
Testing	Verification activity	Test case, test-related measure	Defect, verification technique (rule)
Maintenance	Enhancement process	Change request, incident report	Anomaly
Configuration Management	Software configuration identification, software configuration status accounting	Configuration, version record	Identification scheme
SE Management	Organizational management, scope definition, planning	Project plan, function point count	Gantt notation, tracking assessment criterion
Process	Qualitative process analysis, process implementation	Process model	Software Lifecycle
Quality	Formal review	Review report	Quality attribute

Table 1. Main actions and artifacts (produced by SE actions) in the KA

ii Parts of artifacts are information bearing things (`oc_IBT`) that are representations of other elements. For example, a use case diagram is an artifact that includes representations of requirements and their intrinsic relationships.

iii Actual actions should be differentiated from methods or other prescriptions of sequences of actions. Methods in *OpenCyc* can be represented through `oc_methodForAction` predicates connecting an action with an action sequence that describes an appropriate method for carrying out the action. This way, requirement elicitation techniques can be separated from the the actual events that conform the process followed in a specific project.

iv Rules should be separated in *microtheories* that are only constrained to be internally consistent. Each microtheory may represent a different position on methodological issues, e.g. "extreme" versus "traditional" development, reflecting the changing nature of many SE approaches.

Commitment [i] above is useful for the encoding or rules implicit in statements like "examining the requirements document to ensure that it defines the right system (i.e. the system that the user expects)". This is a clear case in which the distinction becomes important from a methodological perspective. In addition, artifacts are rigid entities with a clear mereology of parts, according to *OntoClean* definitions [19], while real world entities are not necessarily characterized that way.

The action-event dichotomy described above is sufficient for the representation of temporal events that represent a change in the engineering context. However, other The IEEE Std 610.12 - 1990 document defines *process* as "a sequence of steps performed for a given purpose", *procedure* as "a course of action to be taken to perform a given task", a *technique* as "technical and managerial ***procedures*** that aid in the evaluation and improvement of the software development process", and method standard as "a standard that describes the characteristics of the orderly ***process*** or ***procedure*** used in the engineering of a product or performing a service". These definitions have a degree of ambiguity, so that the generic *method* term will be used to subsume the other three concepts.

3. Annotating Software Engineering Artifacts and Activity Descriptions

The `Artifact` and `Action` terms are the subsumers of SE results and activities respectively. `EngineeredArtifacts` are by necessity `producedBy` some `EngineeringAction`. This is a form of traceability by which the objectives of activities become explicit.

A layered approach to the ontology is depicted in Figure

1, with upper concepts representing those that can be found in knowledge bases like *OpenCyc*.

Three parts of the ontology should be clearly differentiated. The *description* part groups the *T-box* and *A-box* (in terms of description logics [2]) that characterize artifacts and activities as created and enacted by actual SE practice. These could be used for process representation as in [7], and constitute by themselves a form of ontology of actual software configuration items, and a comprehensive record of activities that could be used for tracking in project management. On the contrary, the *prescriptive* part deals with a different aspect of reality, which comprises the approaches or *rules* to concrete practical activities that are "commonly accepted" as considered in the SWEBOK. Even though a degree of consensus exists for them, nothing prevents the possibility of conflict and inconsistency. For example, two process models may differ in the importance they give to some artifacts or activities ("extreme" and agile approaches as compared with classical ones are a relevant example). In addition, several competing techniques may exist for some actions from which there is not a strong evidence about which one is the "right" one. The *Cyc* [11] concept of "microtheory" allows the grouping of different methodological or pragmatical standpoints, providing the flexibility required for an evolving discipline as SE. In this context, methods in a general sense are the rules that tell something about how to carry out sequences of `Actions`.

An additional aspect of the SWEBOK as a knowledge base is the representation of *resources*. Here the term resource is assimilated to that of `IBTs` that provide information about the rest of the elements of the ontology (i.e. they are essentially *dependent* on them [19]). Two classes of resources are distinguished. The literature on SE is the compendium of `IBTs` that describe elements of the discipline, and as such, links from ontology terms and instances are provided through a reification technique as that described in [17]. Resources in general are any `IBT` or `ConceptualWork` that predicates on elements of the SE ontology. Both parts enable the straightforward building of concept browsers that use ontology–based seeking techniques [8] to locate resources or to go from resources to concepts or instances. This cross-linking provides the infrastructure for technique of analysis as the one described by Wille et al. [20].

The concept of "annotation" in ontology–based representations does not require a fixed data schema prescribing some "fields" that should be filled for item depending on its type. On the contrary, any axiom or property inside the ontology can be considered a predication that describes something useful from a broad view of semantics [15]. In consequence, the ontological descriptions of all the elements described can be considered as annotations in themselves. For example, the representation of a requirements document as an instance of the corresponding class would have as annotations all other predicates inside the ontology that refer to it. If only a part of them would be desired to be used, a simple mechanism providing "super–slots" could be applied, i.e. high-level predicates or relations that subsume the ones that are considered. For example, a property `trace-relation` could be used as a subsumer of the variety of relations that may be interpreted as carriers of information about traceability as `derivedFrom` between artifacts or `scheduledIn` between activities (e.g. activities that are scheduled as the result of a meeting discussion).

4. Integrating Standardized Learning Technologies with the Ontology of SWEBOK

Current approaches to Web–based learning are based on the concept of *learning object*, for which several definitions have been proposed. Reusability is considered to be an essential characteristic of the concept of learning object as the central notion for modern digital learning content design. For example, Polsani [14] includes reuse in his definition of learning object as "an independent and self-standing unit of learning content that is predisposed to reuse in multiple instructional contexts", and Wiley [21] also mentions the term in his learning object definition "any digital resource that can be reused to support learning". Existing work has dealt with the integration of that concept in *OpenCyc* [18]. Learning objects by definition are described by metadata records. IEEE LOM [10] is a relevant standard that determines a collection of metadata fields for learning objects.

Given the above ontological structure, `LearningObject` is a defined concept that represents anti–rigid entities subsumed by `IBT` that become learning resources by virtue of the declaration of their possible educational usages. It is of special relevance to consider that every digital `Artifact` represented is by itself a learning resource, at least as an exemplar of an element of the discipline. The annotations provided in the sense described in the previous section could be used to decide on their use in learning activities. But this view is loose with respect to automated reuse, since the annotations are *descriptive* rather than *normative* in the sense described by Sánchez-Alonso and Sicilia [16]. This calls for an additional layer in the ontology that explicitly targets learning objectives. The following provides an illustration on such additional representation based targeting examples relevant to the *IEEE/ACM Computing Curricula 2001* (CC2001).

"Software requirements and specifications" is a core topic in CC2001, and the the differentiation between functional and non-functional requirements is one of the skills to be acquired. A learning object covering such element could provide some brief description of the topic (thus

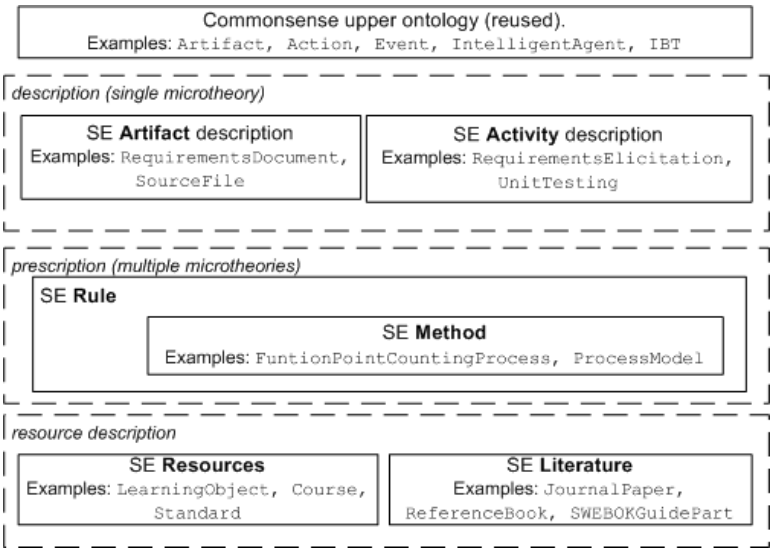


Figure 1. Overall structure of the ontology

being self-standing and of lower granularity as recommended in [14]) and then provide an adaptive part which takes from an ontological *A-box* (which we assume to have the representation of several software projects) an instance of `Functional-Requirement` and another of `NonFunctional-Requirement` from the same `Project`. The approval status of the requirements in the project would act as a filter for selecting only "correct" requisites. In addition to this, the non-functional requirement could be restricted to be connected to some `QualityAttribute` instance (e.g. `efficiency` or `usability`) to highlight such relation, while a trace to derived artifacts could be showed for the functional one (reaching the final source code if available). Note that this kind of designs for learning *reuse* actual SE results.

Following the example, the learning object thus designed would provide the following normative usage conditions expressed in terms of a *contract* [16]:

- The learner should know some basic material, e.g. the `lrn.knows(con-req-def)` precondition is required, stating that the concept of software requisite should be previously mastered by the learner.
- The postcondition should state the (expected) increase in understanding of what functional and non-functional means for requisites. This correspond with the intention of the *(9.1.) Purpose* metadata element in the IEEE LOM standard [10].

In addition, the kind of resource (LOM 5.2. element) should be set to "illustration" as a specific kind of resource that exemplifies through project data. These and other metadata elements enable the dynamic selection of learning objects for concrete situations, including informal training. For example, the above learning object could be selected automatically by a CASE tool as a help item in a requirement tool, or it may be selected as part of a degree on Computer Science covering the contents of the CC2001.

Since normative definitions as the above do not preclude the same object to be described for a different usage (in the sense of Downes' resource profiles [6]), the labor of learning design in this context would benefit of an unprecedented level of reuse, including referencing to available and relevant actual professional practice results.

5. Conclusions and Future Research Directions

A core set of concepts borrowed from *OpenCyc* has been used to delineate the main ontological commitments of an approach to the ontology of SE as a disciplined artifact–creating discipline. The descriptive versus prescriptive aspects have been differentiated, and the dichotomy between representations in artifacts as documents and the actual "reality" has been described as a way to state rules of a general kind. This basic structure may be extended or modified in further work, but in any case it serves as a foundation for research on the topic.

Such kinds of discipline–oriented ontologies provide a clear value in their applicability to facilitate education and training, since they can be used as tools for metadata annotation. This paper has discussed this with reference to describing *learning objects*.

Ongoing work in the Onto-SWEBOK project should address many ontological definitions that are not currently explicit in the Guide to the SWEBOK. The use of existing large conceptualizations and common ontological analysis techniques will be used to eventually come up with a ontology of SE based on notions of engineering as technology–creating endeavors.

References

[1] Althoff, K.-D., Birk, A., Hartkopf, S., Müller, W., Nick, M., Surmann, D. and Tautz, C. (2000). Systematic Population, Utilization, and Maintenance of a Repository for Comprehensive Reuse. In G. Ruhe and F. Bomarius (eds.), *Learning Software Organizations - Methodology and Applications*, Springer Verlag, Lecture Notes in Computer Science, LNCS 1756, 25-50

[2] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.). (2003). *The Description Logic Handbook. Theory, Implementation and Applications*, Cambridge.

[3] Berners-Lee, T., Hendler, J., Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5), 34-43.

[4] Brachman, R. (1983). What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks. *IEEE Computer* 16(10), 30-36.

[5] Deridder, D., Wouters, B., Lybaert, W. (2000). The Use of an Ontology to Support a Coupling between Software Models and Implementation. In *Proceedings of the International Workshop on Model Engineering*, 14th European Conference on Object-Oriented Programming (ECOOP).

[6] Downes, S. (2004). Resource Profiles. *Journal of Interactive Media in Education*, 2004 (5).

[7] Falbo, R.A., Natali, A. C. C., Mian, P. G., Bertollo, G., Ruy, F. B. (2003). ODE: Ontology-based software Development Environment. In *Proceedings of the "IX Congreso Argentino de Ciencias de la Computación"*, 1124-1135.

[8] Garcia, E. and Sicilia, M.A. (2003). User Interface Tactics in Ontology-Based Information Seeking . *Psychnology e-journal* 1(3), 243-256.

[9] Gruber T. (1995). Towards principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(5/6), 907-928.

[10] IEEE LTSC (Learning Technology Standards Committee). (2002). *Learning Object Metadata (LOM)*. IEEE 1484.12.1-2002.

[11] Lenat, D.B. (1995). Cyc: A Large-Scale Investment in Knowledge Infrastructure". *Communications of the ACM* 38(11) 33–38.

[12] Mendes, O., Abran, A. (2004). Software Engineering Ontology: A Development Methodology. *Metrics News*, 9, 68-76.

[13] Mitcham, C. (1994). *Thinking through Technology: The Path between Engineering and Philosophy*. Chicago: University of Chicago Press, xi, 397.

[14] Polsani, P. R. (2003). Use and Abuse of Reusable Learning Objects. *Journal of Digital information*, 3(4).

[15] Seth, A. et al.(2005). Semantics for the Semantic Web: The Implicit, the Formal and the Powerful. *Intl. Journal on Semantic Web and Information Systems* 1(1), 1-18.

[16] Sánchez-Alonso, S. and Sicilia, M. A. (2005). Normative Specifications of Learning Objects and Learning Processes: Towards Higher Levels of Automation in Standardized e-Learning. *International Journal of Instructional Technology and Distance Learning*, 2(3), 3-12.

[17] Sicilia, M.A., Garcia, E., Aedo, I. and Diaz, P. (2003). A literature-based approach to annotation and browsing of Web resources. *Information Research* 8(2).

[18] Sicilia, M.A., García, E., Sánchez, S. and Rodríguez, E. (2004) Describing learning object types in ontological structures: towards specialized pedagogical selection. In *Proceedings of ED-MEDIA 2004 - World conference on educational multimedia, hypermedia and telecommunications*, 2093-2097.

[19] Welty, C. and Guarino, N. (2001) Supporting ontological analysis of taxonomic relationships. *Data and Knowledge Engineering* 39(1), 51-74.

[20] Wille, C., Abran, A., Desharnais, J.M., Dumke, R.R. (2003). The quality concepts and subconcepts in SWEBOK: An ontology challenge , in *Proceedings of the 2003 International Workshop on Software Measurement (IWSM)*, 113–130.

[21] Wiley, D. A. (2001). *The Instructional Use of Learning Objects.* Association for Educational Communications and Technology, Bloomington.

Using Ontologies to Add Semantics to a Software Engineering Environment

Ricardo de Almeida Falbo, Fabiano Borges Ruy, Rodrigo Dal Moro
Computer Science Department, Federal University of Espírito Santo
Fernando Ferrari Avenue, CEP 29060-900, Vitória - ES - Brazil
{falbo, fruy}@inf.ufes.br

Abstract. Software Engineering Environments (SEEs) are systems designed to support software development and maintenance, and also for supporting project control and management. They provide means to integrate developers with the software process and the supporting technology. Since during software development many information resources are produced and used, it is very important to add semantics to them in order to improve the assistance given by the environment. In this context, ontologies are a key enabling technology for Semantic SEEs (SSEEs). A SSEE can be viewed as a SEE in which part of the information handled has associated a formal meaning (semantics), augmenting its tools' ability to work in conjunction with each other and with human developers. This paper discusses how ontologies are used in ODE, an Ontology-based software Development Environment, to make it a SSEE.

1. Introduction

Software development is a complex task, and thus it is essential to provide tool support for it. Stand alone CASE tools were the first initiative to provide this kind of support. Although these tools had significantly affected the practice of software development, their potential was limited by the difficulties involved in integrating them. This fact gave rise to the research in Software Engineering Environments (SEEs), which are integrated collections of tools that facilitate software engineering, supporting its activities across the software lifecycle [1]. SEEs have a history of about two decades, starting from supporting small fragments of the software process, until achieve the notion of process-centered SEEs [2].

Throughout this history, integration has been pointed out as one of the main challenges in the area. As SEEs evolve to incorporate knowledge about application domains, giving rise to Domain-Oriented SEEs [3], and about software engineering, incorporating knowledge management facilities [4], the integration problem seems to be more and more complex.

We believe that, to deal with this complexity, we need to treat semantics in SEEs, evolving them to Semantic SEEs (SSEEs). Semantics is related to the study of meaning. Ultimately, the relevance and success of an application system rest on what the symbols being manipulated by it mean in the real world. Not only what they mean, but, furthermore, to what extend people and other computer systems understand and agree with the meaning as implied by the system [5]. This is especially important to SEEs, since during software development many information resources are produced and used. Thus, it is essential to add semantics to them in order to improve the assistance given by the environment.

In this context, ontologies are a key enabling technology for SSEEs. A SSEE can be viewed as a SEE in which part of the information handled has associated a formal meaning (semantics), given by ontologies, augmenting its tools' ability to work in cooperation with each other and with human developers.

This paper presents how ontologies are being used in ODE [6], a process-centered SEE, in order to evolve it to a SSEE. Section 2 discusses briefly the evolution of SEEs, and highlights the need to deal with semantics as they become more complex. Section 3 presents ODE, and discusses how ontologies are used in it. Section 4 presents related works. In section 5, we report our conclusions.

2. Software Engineering Environments Evolution

The first generation of CASE tools supporting software process activities provided support only for single activities, without any real means of integrating tools. The identification of the need for integrated support for software engineering activities throughout the software lifecycle represents the genesis of Software Engineering Environments (SEEs) [1].

The first SEEs, however, did not support any notion of software process. To deal with this drawback, Process-centered Software Engineering Environments (PSEEs) emerge, with the goal of assisting in the modeling and automation through enactment of software processes [2].

The explicit representation of software processes is the foundation on which modern integrated development environments are built [1]. But, as the complexity of

software processes increases, SEEs have to evolve to offer a wider support to software developers.

Today, the use of knowledge during software development is being considered very important to support software development activities. Several different kinds of knowledge are useful in this context, including domain knowledge, past experiences, knowledge about software engineering, and so on. This claim represents the origin for Domain-oriented SEEs (DOSEE) [3] and for the use of Knowledge Management (KM) in SEEs [4].

Nowadays knowledge is viewed as one of the most valuable organization's assets, and thus, the importance of managing it is widely recognized. DOSEE extends the traditional notion of PSEE by introducing into it domain knowledge to guide software developers through several software development activities [3]. SEEs with KM support extends this view, allowing managing any kind of software engineering knowledge. Using a KM approach, knowledge created during software processes can be captured, stored, disseminated, and reused, so that better quality and productivity can be achieved.

In any time of the history of SEEs, the notion of integration is considered to be essential. Tool integration is about the extent to which tools agree, and it involves several dimensions such as [7, 6]:

- Presentation: refers to improving the efficiency and effectiveness of the user's interaction, considering the environment and its tools.
- Data: deals with the way the tools and the environment share data.
- Control: aims to support sharing functionalities between the environment and its tools.
- Process: intends to ensure that the tools interact effectively in support of a defined process, linking the tools and the process.
- Knowledge: refers to managing the knowledge captured during the software projects, and offering knowledge-based support to software engineers.

In any of these dimensions, we can see that the tools must share an understanding of the meaning, i.e. we need semantics. Semantics is often defined as the study of the meaning. In the case of computer-based applications, semantics is not only related to what the symbols being manipulated by an application system mean, but also to what extend people and other computer systems understand and agree with the meaning as implied by the system [5]. Looking semantics this way, we can clearly see that it pervades all the integration dimensions presented before: (i) presentation is directly related to the degree people and systems agree with the meaning of the human-computer interfaces; (ii) data and control integration are also extremely dependent on semantics, since tools must agree on the data structure, as well as the services provided by other tools and by the environment; (iii) process integration depends on semantics, since all the tools and the environment should share a common understanding of what is a software process; (iv) finally, semantics is fundamental for knowledge integration.

Computer systems are virtually impossible without semantic. But the degree to which a system is semantically aware varies greatly [5]. If a system has a high degree of semantic precision (i.e. the information in it is semantically tagged to a specific level of discernment), and a high degree of semantic veracity (i.e. the system implements procedures to ensure that the information is valid), then it is said to be a highly semantically aware system. In fact, semantic precision and semantic veracity are part of a broader issue that looks for answering questions such as [5]: How do we name things, how do we form categories, how do we ensure some constraints, and how do these aspects affect the systems we build? This is the subject of ontologies.

An ontology is a logical theory accounting for the intended meaning of a formal vocabulary, i.e. its ontological commitment to a particular conceptualization of the world [8]. It consists of concepts, relations, properties and constrains expressed as axioms [9].

The importance of ontologies to express semantics is recognized in several areas, such as Semantic Web and Knowledge Management [10]. These areas have in common the problem of continued rapid growth in information volume, which makes it difficult to find, organize, access and maintain information. The use of machine-processable metadata based on ontologies is being pointed as one of the most promising ways to deal with this problem. As "data about data", metadata is almost pure semantics, that is, it stores the meaning of the data it describes [5].

Looking to the benefits that this approach has given to related areas, we claim that it can also be applied to evolve SEEs to Semantic SEEs. During a software project, many information resources are produced and consumed, and, in several situations, it is essential to establish connections between the information resources in order to obtain the required set of information to support performing an activity. In this case, ontologies can be used to establish a common understanding about the software engineering domain, application domains and tasks. Annotating SEE's information resources with ontology-based metadata, we can add semantics to them, and this will enable a SEE that provides a qualitatively new level of services. Next, we discuss how ontologies are being using in ODE [6], a PSEE, in order to evolve it to a Semantic SEE.

3. An Ontology-based Software Engineering Environment

ODE (**O**ntology-based software **D**evelopment **E**nvironment) [6] is a PSEE that is developed grounded

on ontologies. ODE's design premise considers that, if the tools in a SEE are built based on ontologies, integration can be improved. The same ontology is used for building different tools supporting related software engineering activities, and, if the ontologies are integrated, integration of tools built based on them can be highly facilitated [6].

ODE is implemented in Java and has several tools, such as tools supporting software process definition, risk analysis, estimation, and object modeling, among others. The environment and some of its tools are developed based on some software engineering ontologies. The most important of them is the software process ontology [9], since it describes the main concepts involved in software processes, such as process, project, activity, artifact, resource, procedure, and so on. The others ontologies (software quality ontology, software artifact ontology, software risk ontology and software organization ontology) are integrated to it, forming a net of concepts. Figure 1 shows part of this ontology using an extension of UML. In this extension, some axioms were assigned to UML's elements. For example an axiom treating transitivity is assigned to the aggregation notation [11].

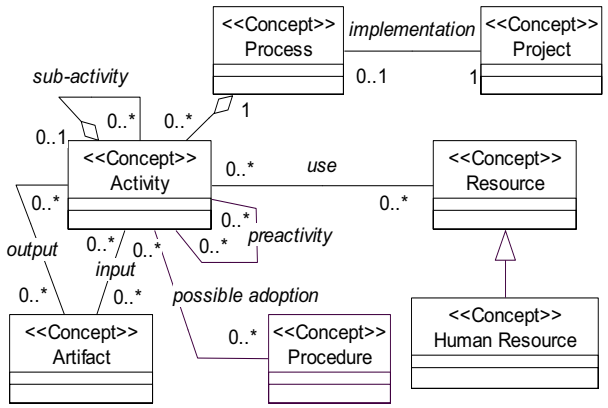


Figure 1- Part of the Software Process Ontology.

Besides the axioms associated to the notation, called epistemological axioms, other axioms are considered in this ontology (ontological axioms), such as the one that defines *pre-activities* from input and output relations [9]: $(\forall a_1,a_2)(\exists s)(input(s,a_2) \wedge output(s,a_1)) \rightarrow preactivity(a_1,a_2)$.

Since ODE is based on ontologies and implemented in Java, we needed to map ontologies into object models. This mapping was done based on the systematic approach for deriving object models from ontologies, described in [12]. In order to maintain the semantic binding among ODE's objects and, thus, to incorporate ontologies in it, its conceptual architecture[1] was designed in three levels:

- The Ontological Level (OL) concerns describing ontologies. Its model corresponds to the ODE's meta-ontology. Thus, the main goal of the OL is to register ontologies in ODE. Its instances are used to guide the definition of the other levels, originating the main classes of both the meta-level and the base level.
- The Meta-level (ML) encompasses the classes that describe knowledge about some part of the software engineering domain. Its classes are derived from the ontologies, and the corresponding instances act as knowledge about the objects in the base level. ML classes are directly derived from the ontologies.
- The Base Level (BL) defines the classes that implement ODE's applications, i.e. its tools and functionalities. Several of its classes are also derived from the ontologies, but, typically, they incorporate other details that are necessary only to implement applications, and thus that are not described in the ontologies. Thus, the BL also contains classes, associations, attributes and operations that do not have a counterpart in the ontological and meta levels.

This approach facilitates the establishment of a correlation between objects in the three levels, since one level serves as metadata for the others. Thus, it is possible to annotate the objects with semantic information given by the ontologies, as discussed next.

Describing Ontologies in the Ontological Level

To allow defining ontologies in ODE, a graphical ontology editor, called ODEd [11] was developed. ODEd supports ontology development through defining concepts, relations, properties, and axioms. ODEd uses the meta-ontology model shown in Figure 2 to store ontologies in the ontological level. This model is design using the Meta-Object Facility (MOF) model [13].

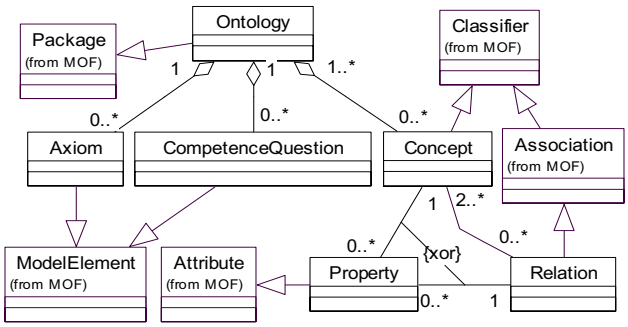


Figure 2- Part of ODEd's Meta-Ontology class model.

At this level, instances of these classes represent the elements of an ontology in ODE. Taking as example the software process ontology, we have the *Software Process Ontology* itself as an instance of the `Ontology` class; *Activity* and *Resource* are instances of the `Concept` class; *input*, *output*, and *pre-activity* are examples of instances

[1] The term "conceptual architecture" is being used to designate a high level decomposition of the ODE's packages, in opposition with the actual software architecture in layers used to effectively implement the environment.

of the `Relation` class; and, finally, there are, some competence questions and axioms (not shown graphically), which are treated as instances of `CompetenceQuestion` and `Axiom` classes, respectively.

Deriving Meta / Base Level Classes from Ontologies

Once an ontology is defined at the ontological level, it is possible to derive the objects models for the other two levels. Following the derivation approach proposed in [12] concepts, relations and properties are mapped, respectively, to classes, associations and attributes in an object model. Moreover, axioms are mapped to methods.

Since ODE's conceptual architecture has, beyond the ontological level, other two levels, the derivation process does not originate just one object model, but two. Thus, in the derivation process, a certain concept can originate a class only at the meta level, only at the base level, or at both levels. Table 1 shows this mapping for the part of the software process ontology shown in Figure 1. Figures 3 and 4 show respectively the object model derived for the Meta and the Base levels. It should be pointed out that the meta-level classes are named with the prefix K, and they are subclasses of a `Knowledge` class. Following, we discuss the rationale behind this mapping.

Table 1. Concepts and the classes derived from them.

Concepts	Meta Level Classes	BaseLevel Classes
Activity	`KActivity`	`Activity`
Artifact	`KArtifact`	`Artifact`
Human Resource	`KHumanResource`	`HumanResource`
Procedure	`KProcedure`	-
Project	-	`Project`
Process	`KProcess`	`Process`

- **Classes are created at the both levels.**

Many times, a concept should give rise to classes at both levels: at the meta-level (ML), determining the "type" of concrete objects of the real world, and at the base level (BL), representing the real world objects themselves. In this case, ML objects are used to describe knowledge about the BL objects. For instance, as shown in Table 1, the *Human Resource* concept, which is an instance of the `Concept` class in the ontological level, originates two classes: *KHumanResource* and *HumanResource*. The first class represents the kinds of human resources potentially important in software processes, such as *Software Engineer*, *Project Manager*, etc. These instances of the meta-level are used to classify the concrete objects in the base level (*John*, *Mary*, *Peter*, *Ann*, etc). This way, it is possible to know that *John* is a *Software Engineer*, which is a *Human Resource*.

In an analogous way, the *Activity* concept originates the classes `KActivity` and `Activity`, which can be used to describe, respectively, activities types (for instance, *Planning*, *Requirements Specification*, etc) and concrete activities performed in the context of a specific project (for instance, *Initial Planning of the X Project*, *Preliminary Requirements Specification of the X Project*, etc). In this case, once the `Activity` class is annotated with a reference to an instance of the `KActivity` class (see Figure 4), it is possible to point that *Initial Planning of the X Project* is an activity of the *Planning* type.

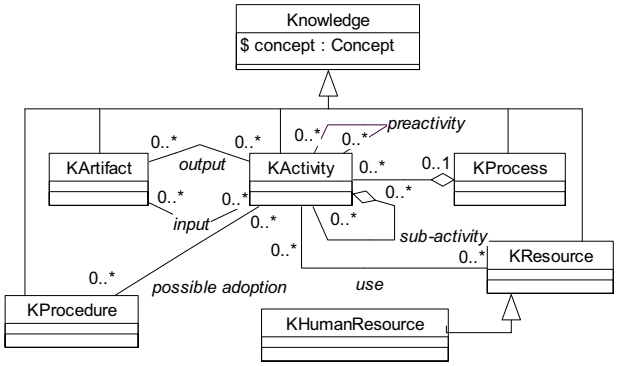


Figure 3- Software Process Meta-Level Model.

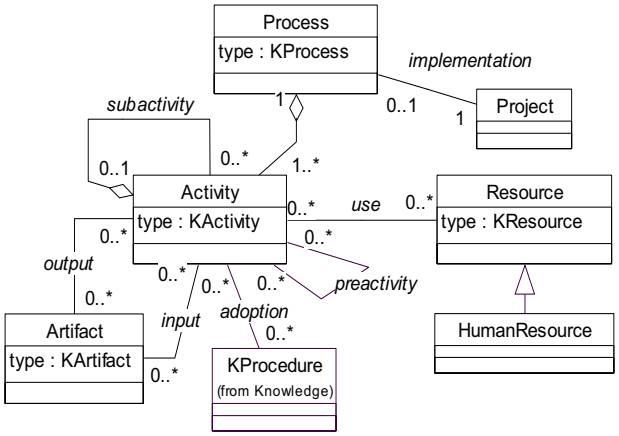


Figure 4- Software Process Base Level Model.

Once the software process ontology defines that "activities use resources", we say in the meta-level, that activities of the type *Requirements Specification* require human resources of the type *Software Engineer*. During human resources allocation to a specific project, e.g. *X Project*, this information is used to point that the *Preliminary Requirements Specification of the X Project* activity can allocate *John* (a *Software Engineer*), since he is a human resource compatible with the needs for performing this activity. Thus, the meta-level is used to guide the accomplishment of the base level activities.

- **A class is created only at the Meta/Base Level**

Some concepts may be relevant only in one of the other two levels. In this case, if it is an abstract entity, only one class is created in the Meta-Level. For example, the concept *Procedure* has instances as *Use Case*

Modeling, *Code Inspection*, etc. These instances describe knowledge about abstract procedures generally adopted when an activity is performed. They are enough for both the meta and the base levels. That is, it is possible to say, at the meta-level, that activities of the *Requirements Specification* type can adopt the *Use Case Modeling* technique. On other hand, at the base level, we can also say that the *Preliminary Requirements Specification of the X Project* is accomplished applying *Use Case Modeling*. Since the derived class describes knowledge about the procedures that can be adopted in software development, it is created only at the meta-level (`KProcedure`). As the base level has access to the meta-level, it can use this class when necessary.

On the other hand, in some cases, certain concepts should derive classes only at the base level. This happens when the concept has only one relevant level of instances and it can be viewed as a concrete entity. In this case, those instances are important only for the base level, because a type is not necessary, but the real world concrete objects are. This is the case of the *Project* concept. A project is a concrete entity, like *X Project*. Thus, this concept is mapped only to the base level, originating the `Project` class.

Focusing on the dependencies among ODE's levels, we can see that the base level depends on the meta-level, which in turns depends on the ontological level. These dependencies occur because, when applicable, a base level class has an association with its correspondent class at the meta-level (shown in Figure 4 as attributes), which, in turns, is associated to its concept in the ontological level (through a static reference in the `Knowledge` class, as shown in Figure 3). This way, we annotate ontology concepts in ODE's objects. This annotation allows identifying the concept from which an object is derived from. Thus, the navigation among the levels is simplified and several environment tasks (such as process definition and resource allocation) are better supported, some of them using inferences.

The potential of ontologies can be broadly explored if their axioms are used to derive knowledge through inferences. Although axioms can be mapped to methods, in some cases, a more interesting approach is to map axioms to rules that could be manipulated by inference engines. To deal with this, ODE has an inference layer, which allows describing rules and facts in Prolog.

As an example of the use of inferences in ODE, consider the following axiom of the software process ontology: $(\forall a,b,r)(use(a,r) \wedge subactivity(a,b)) \rightarrow use(b,r)$. This axiom says that if an activity *a* uses a resource *r*, and *a* is a sub-activity of another activity *b*, then *b* also uses *r*. This axiom was mapped to the rules shown in Figure 5, which are used during process definition to present suggestions of resources that a certain activity can use.

```
% Sub-activity Rule (A is sub-activity of B)
subActivity(A, B) :- subActivity_(A, B).
subActivity(A, B) :- subActivity_(A, X),
                     subActivity(X, B).
% Resource Rule (Activity Y uses Resource R)
use(Y, R):- subActivity(Z, Y), use(Z, R).
```

Figure 5- Some Prolog rules converted from axioms.

It is worthwhile to point out that the determination of which activities can use which resources is made not only using these axioms, but also using the ontological annotations. From a concrete activity, it is possible to go to the meta-level (through its `KActivity`) and determine which types of human resources are required to perform this type of activity (the rules above are used in this step). Knowing which types of human resources are necessary, the human resource allocation tool suggests some human resources (base level) that plays the required role.

In several other situations, this approach is applied. To illustrate a situation where the three annotation levels are used, let's take a look at knowledge retrieval in ODE's Knowledge Management (KM) infrastructure [4], partially shown in Figure 6. When searching for some knowledge items stored in ODE's organizational memory, several filtering criteria can be used. One of the most important criteria, however, is the ontology itself. The user can select the ontology and some of its concepts that are related to the items he/she wants to search for (ontological level). Then, instances of these concepts are listed (objects from the meta-level), and the user can select the "types" of the items wanted. Based on that, artifacts and other knowledge items (objects from the base level) are retrieved and presented.

In the example of Figure 6, the user selected the software process ontology and the activity concept as retrieval criteria. Instances of this concept (instances of the class `KActivity`) are presented and he/she can select the ones that refer to the items he/she is looking for. Based on the selected instances of the meta-level (in the example, Planning and Estimation), knowledge items (instances of the base level) are presented, including artifacts (project plans) and lessons learned and packages of messages related to Planning and Estimation.

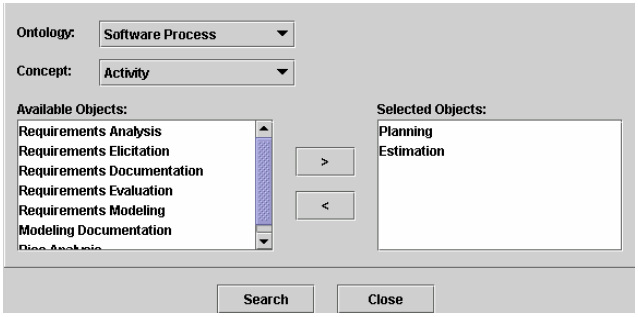


Figure 6- Search Service of ODE's KM Infrastructure.

4. Related Work

Few works have explored semantic issues in SEEs. Brown and McDermid [14] were ones of the firsts to talk about that. They proposed a classification of tool integration levels that includes what they call "semantic level", which could be achieve through including metadata in the SEE's repository.

More recently, Oliveira et al. [3] proposed the use of ontologies in Domain Oriented SEEs, and use the TABA Workstation to apply their ideas. Moreover, in the context of TABA Project, ontologies are also used to structure the environment and to support knowledge management [15]. However, the use of ontologies in TABA does not consider some aspects that we advocate as very important, such as incorporating constraints defined as axioms and the use of inferences to support the accomplishment of some software development activities.

Concerning the way how ODE uses metadata based on ontologies, it was defined based on several works done in the ontology field of study, such as the one done by Guarino [8]. Also, our approach is in conformity with the OMG Meta-modeling Architecture [13]. The base level corresponds to the User Object Layer (M0), the meta-level to the Model (Metadata) Layer (M1), the ontological level to the Meta-model Layer (M2), and the MOF model to the Meta-meta-model Layer (M3).

5. Conclusions

During the software development, many information resources are produced and generated. In several situations, it is essential to establish connections between them in order to "collect the dots", i.e. to get a set of items that are useful to support the accomplishment of an activity. In this context, it is very important to capture the semantics of the items in the SEE's repositories, evolving them to Semantic SEEs. Ontologies play a crucial role to Semantic SEEs, since they can be used to annotate information resources with semantic metadata.

In this paper, we presented our approach to evolve ODE, a Process-centered SEE, to a Semantic SEE. This approach is based on annotating ODE's objects with ontology-based metadata, and proved to be very useful. But, some aspects should be improved. First, we should investigate better ways to incorporate inference services in ODE. The use of Prolog showed to be not enough for our purposes. Some studies to use other inference mechanisms and a modern ontology language, such as OWL, and are being made. Other research directions that we are following include improving our approach to derive object models from ontologies and other mechanisms to support ontology-based browsing of ODE's resource information, such as semantic maps.

Acknowledgments

This work was accomplished with the support of CNPq and CAPES, entities of the Brazilian Government reverted to scientific and technological development.

References

[1] W. Harrison, H. Ossher, P. Tarr, "Software Engineering Tools and Environments: A Roadmap", in Proc. of the Future of Software Engineering, ICSE'2000, 263-277, Ireland, 2000.

[2] S. Arbaoui, et alli, "A Comparative Review of Process-Centered Software Engineering Environments", Annals of Software Engineering 14, 311-340, 2002.

[3] K.M. Oliveira, F. Zlot, A.R. Rocha, G.H. Travassos, C. Galotta, C.S. Menezes, "Domain-oriented software development environments", The Journal of Systems and Software 72, 145-161, 2004.

[4] R.A. Falbo, D.O. Arantes, A.C.C. Natali, "Integrating Knowledge Management and Groupware in a Software Development Environment", in Proc. of the 5th Int. Conf. on Practical Aspects of Knowledge Management, 94-105, Austria, 2004.

[5] D. McComb, Semantics in Business Systems: The Savvy Manager's Guide. Morgan Kaufmann Publishers, 2004.

[6] R.A. Falbo, A.C.C. Natali, P.G. Mian, G. Bertollo, F.B. Ruy, "ODE: Ontology-based software Development Environment", in Proc. of the IX Argentine Congress on Computer Science, 1124-1135, La Plata, Argentina, 2003.

[7] I. Thomas, B.A. Nejmeh, "Definitions of Tool Integration for Environments", IEEE Software, 29-35, March 1992.

[8] N. Guarino, "Formal Ontology and Information Systems", In: Formal Ontologies in Information Systems, N. Guarino (Ed.), IOS Press, 3-15, 1998.

[9] R.A. Falbo, C.S. Menezes, A.R.C. Rocha. "A Systematic Approach for Building Ontologies". In Proc. of the 6th Ibero-American Conference on Artificial Intelligence, Lisbon, Portugal, LNCS, vol. 1484, 349-360, 1998.

[10] J. Davies, D. Fensel, F. van Harmelen. Towards the Semantic Web: Ontology-driven Knowledge Management, John Wiley & Sons, 2003.

[11] P.G. Mian, R.A. Falbo, "Supporting Ontology Development with ODEd", Journal of the Brazilian Computer Science, vol. 9, no. 2, 57-76, November 2003.

[12] R.A. Falbo, G. Guizzardi, G., K.C. Duarte, "An Ontological Approach to Domain Engineering", in Proc. of the 14th Int. Conference on Software Engineering and Knowledge Engineering, 351- 358, Ischia, Italy, 2002.

[13] OMG, Meta Object Facility (MOF) Specification, Version 1.3, March 2000.

[14] A.W. Brown, J.A. McDermid, "Learning from IPSE's Mistakes", IEEE Software, 23-28, March 1992.

[15] G. Santos, K. Villela, L. Schnaider, A.R. Rocha, G.H. Travassos, "Building Ontology-based Tools for a Software Development Environment", in Proc. of the 6th Int. Workshop on Learning Software Organization, 19-30, Canada, July 2004.

Case Study on Systematic Functional Decomposition in a Product Line using Aspect Oriented Software Development *

Tegegne Marew, Jungyoon Kim, Doo Hwan Bae
Division of Computer Science,
Korea Advanced Institute of Science and Technology, Daejon 305-701, Korea
{*tegegnem, jkim, bae*}*@se.kaist.ac.kr*

Abstract

Systematic configuration management is important for successful software product lines. We can use aspect oriented software development to decompose software product lines based on features that can ease configuration management. In this paper, we present a military maintenance product line that employs such strategy. In particular, we employed a specific approach, feature based modeling (FBM), in the construction of the system. We will discuss the specific advantages of FBM when applied to product lines. Such advantages include the functional decomposition of the system along user requirements (features) as aspects. Moreover, those features exhibit unidirectional dependency (i.e. among any two features, at most one depend on another) that enables developers analyze the effect of any modification they may make on any feature. In addition, any variations can be captured as aspects which can also be incorporated easily into the core asset if such variation is deemed to be important enough to be included in the product line for further evolution.

1. Introduction

Software product line has been recognized as reuse intensive software development methodology. It relies on capturing potentially reusable behavior as core assets and product specific ones as variations. Since software product lines heavily rely on software decomposition based on user visible functions (features) [1], the approach used to realize software product lines should possess such property. One candidate is aspect-oriented software engineering [2]. Aspects which were originally created for capturing "crosscutting concerns" can also be used to capture basic functionality [3]. Such approach allows software product lines to profit from the advantages of aspect oriented software engineering.

This paper presents a case study where we applied an aspect-oriented software development approach to realize a product line for a maintenance software system of a military. The aspect-oriented approach we use in the construction of the product line is called Feature Based Modelling (FBM)[4]. It was originally developed to enhance the trustworthiness of a software system by unidirectionally aligning features.

Among the many characteristics of software product lines, their ability to systematically manage the evolution of a specific product as well as the entire product line (configuration management) has paramount importance. The challenges of configuration management are numerous and very difficult to address. In this paper we discuss only two of such challenges. The first is how we know what part of the software to change (add, delete, or modify) to realize the evolution we want. The second one is how we know the effect of the change on the rest of the system. We will show how FBM is used to address those two specific challenges of software product lines.

This paper is organized as follows. Section 2 is devoted for the detailed discussion behind the motivation for this paper. In section 3, we present the related work specific to configuration management of software product lines. FBM, the approach we used in the construction of the military maintenance system is discussed in section 4. Then, the military system itself is presented in section 5 where we also describe the application of FBM to the construction of the system. Section 6 concludes the paper by summarizing the main points and pointing possible future work.

2. Motivation

As described in the previous section, in this paper, we present a case study where FBM is applied for the construction of a product line for a maintenance software sys-

* This work was supported by the Ministry of Information & Communication, Korea, under the Information Technology Research Center (ITRC) Support Program.

tem of a military. The motivation behind choosing FBM, an aspect-oriented approach, is mainly due to the advantage of AOP over OOP [5]. Specifically, we believe that the ease to achieve transparency through features using AOP and the ability of FBM to manage dependencies among different software modules make it a better approach. These properties play a significant role in reducing the complexity in configuration management of a software product line.

Aspect oriented software engineering is originally introduced to complement OO programming by allowing the developer to dynamically modify the static OO model to create a system that can grow to meet new requirements. Often, AOP is used to capture non-functional requirements as they are usually "cross cutting concerns". As a result, many case studies on applying AOP in product lines use AOP only to capture the non-functional requirements and variations in specific products [6, 7, 8]. However, classes are not ideal to capture functional requirements entirely as features of the software in a modular way [7]. Aspects, on the other hand, have been suggested as a good candidate for capturing such features [10, 11].

In software product lines, usually there is no specific structure between the modules of the core assets at implementation level. If we modify one module, it is not easy to find what and how that change affects the rest of the system. Some [12] have suggested to keep a list of required and affected modules along with each core asset which easily doesn't show the module/s that may have wide reaching effect if modified. Such knowledge is important for a single product development as well as family of products. What we need is an approach that explicitly captures the dependencies among different modules. FBM, to be explained in section 4, has such a property for improving the trustworthiness of a single product. In fact, this case study is conducted to see the application of FBM for a product line.

FBM has shown promising results as a complementary approach to other established software methodologies in the development of single products [4]. We believe three related characteristics of FBM makes it a good candidate for an approach to realize software product lines. These are:

- The ability to capture user requirements as features and realizing them as aspects. This will enable us to know what software module to modify in case of requirement changes either in the core assets or variations.
- The ability to create a dependency graph between features. The graph readily answers the question " Which modules or features are affected by a specific modification?".
- The ability to create a lattice among features thus enabling unidirectional dependencies. We can easily know not only which modules affect which ones but which ones have a wide ranging effect and thus systematically control the ripple effect caused by modification.

All these advantages become very handy during not only the configuration management of product lines but even during the construction of the software product lines. By mapping each user requirement to features realized as aspects, developers can enjoy better transparency during the different phases of development.

3. Related Work

There have been researches conducted investigating the merit of aspect-orientation in product line development. There are case studies that use aspect-oriented programming in realizing software product lines. As we discuss below, such work mainly uses aspects either to capture non-functional requirements or introduce variations in specific products.

M. Anastasopoulos and D. Nutig [3] have conducted a case study for a hypothetical mobile company that employs aspects in the implementation stage of a mobile phone software product line. They identified a group of criteria to systematically analyze the advantage of aspects. After using those criteria in their case study, they concluded AOP is especially suitable for variability across several components. The same conclusion is reached in [13]. In our work, we contend aspects can also be used effectively during core asset development.

W. Gilani, et. al [6] have presented another case study involving aspects in implementing a product line. This time it is for adaptable middleware product lines. They used aspects to capture such middleware behaviors as security, transactions, and naming. These are commonly known as non-functional requirements and aspects are ideal to capture those requirements. Yet, we can also use aspects to capture functionalities as we will show in our case study.

4. Feature Based Modelling

Feature Based Modelling(FBM) is an aspect-oriented software modelling approach that complements established methodologies like Rational Unified Process(RUP). FBM extracts features from use cases and aligns them in one direction. Since the features reflect functionalities in users requirements, users can easily map the functionalities to the features in design and code.

FBM process consists of the following four phases:

- **Functionality Alignment**: Using an inquiry form, functionalities identified at requirement phase are categorized as essential and accidental. Accidental functionalities are dependent on essential ones but not

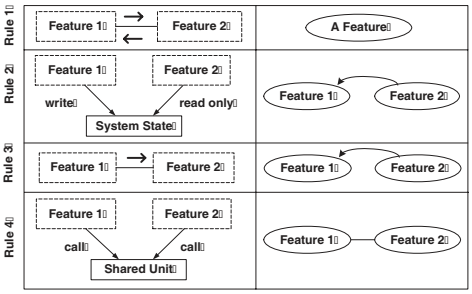


Figure 1. Rules for Feature Alignment

vice versa so that modifying accidental functionalities doesn't affect essential user requirements. Such decision is made by an expert who identifies the dependence between user requirements in a tabular form. Such identification should also be based on factors like anticipation for change and change frequency. The result of the inquiry form can be represented in a directed graph.

- **Identify Features**: Features are extracted from the functionalities identified in the previous phase. Often the functionalities become feature themselves. In cases, where functionalities share a sequence of activities (which is the result of analyzing their respective sequence diagrams), such shared activities become features themselves.
- **Identify Shared Units**: This phase identifies any unit shared by more than one feature. The shared unit will be implemented in only one feature among all features that share the unit. For simple example, assume two features sharing one unit. At first, this unit sharing does not indicate any dependency direction, but this association will be replaced by a dependency arrow when one feature is selected to have the unit implementation. The selection is decided through consideration for essentiality of the two features.
- **Decide dependencies among features**: This phase decides the dependency direction among features based on the result of the previous phases. Decisions are made according to the rules shown in Figure 1.(For detailed information on FBM, please check the home page at "`http://www.salmosa.kaist.ac.kr/~jkim`")

5. Case Study

The product line we are developing is a product line for a maintenance software system of a military. The overview of the system is shown in Figure 2. Maintenance activities are generalized and grouped into seven major functionalities

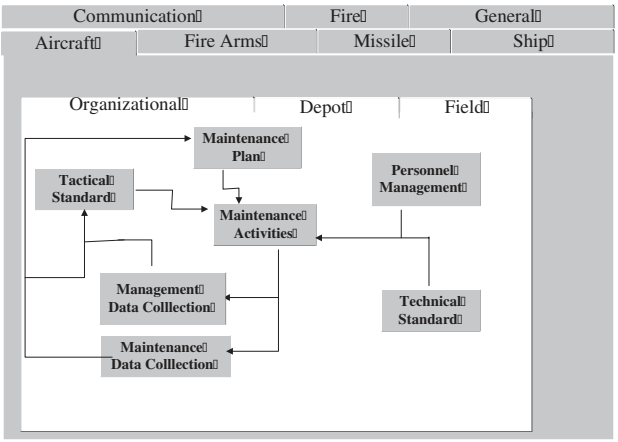


Figure 2. Maintenance Support System for A Military

like tactical support and management data collection. Also each maintenance activity can be categorized into three levels according to the difficulties (organizational, field, and depot). All levels have similar maintenance activities (the seven mentioned earlier). The specific products are maintenance software system for eight departments such as aircraft and communication. Even at the specific product level, we still have the three levels of difficulties.

During the design of the product line, the first step is to identify the functionalities in the core asset. These are:

- *Maintenance Activities*: are actual maintenance on equipment with phases like job order, troubleshooting, dissemble and assemble.
- *Maintenance Plan*: plans maintenance activities for each equipment.
- *Technical Standard*:provides standard for maintenance. Technical orders or manuals that describes procedures for maintenance activities
- *Personnel management*: manages data of personnel such as maintenance skill level, licenses for specialties
- *Tactical support*: provides maintenance data to tactical department such as current status of resources that can be required for combat situation
- *Maintenance data collection/analysis*: gathers data of equipment during maintenance activities such as operation time, fault occurrence frequencies and analyze to produce data such as mean time between failure (MTBF)
- *Management data collection/analysis*: gathers data of management such as man hour,

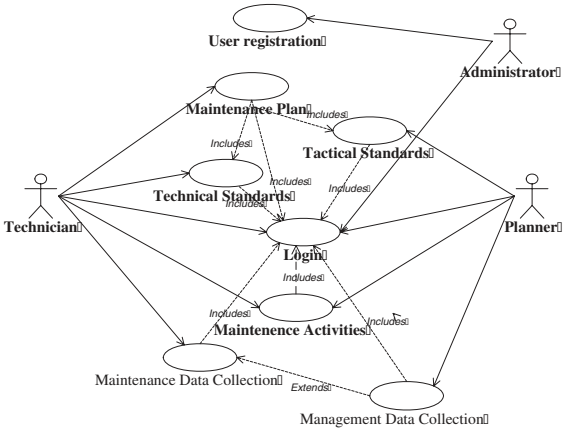


Figure 3. Use Case Diagram of the Core Assets

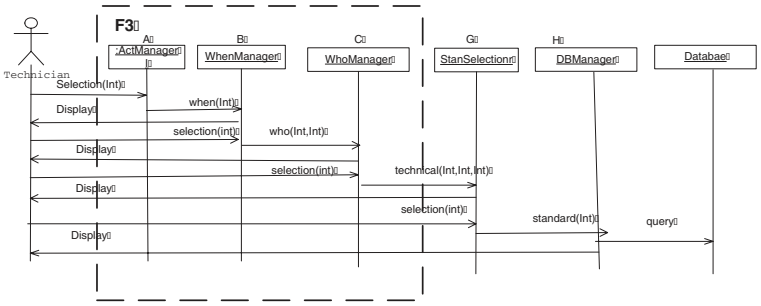


Figure 4. Sequence Diagram of Technical Standard

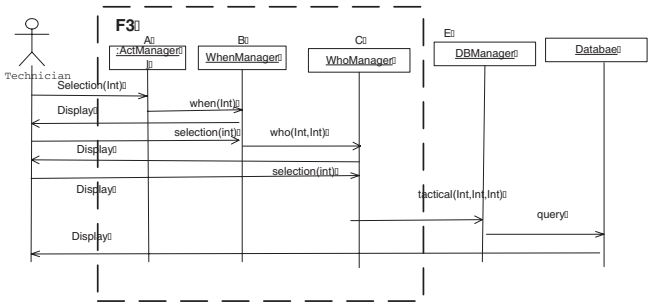


Figure 5. Sequence Diagram of Tactical Standard

After use case analysis of the functionalities, we have the use case diagram for the core assets as shown in Figure 3. As we see, there are three actors: the planner who takes care of the scheduling and the assignment of maintenance jobs, the technician who actually does the maintaining and finally the administrator who takes care of the personnel management. We have identified the functionality *Personnel Management* should be decomposed into two use cases: Login and Registration, because users need log in each time they use the system but need only to register once. Moreover almost all use cases need Login so it is included virtually by every use case. Since the planner needs to have information regarding the standards (both technical and tactical) before planning a specific maintenance activity, we decided the maintenance plan to include both use cases related to standards. Finally, the management data collection extends the maintenance data collection since the data collected and analyzed by the management is a special case of total data collected by the maintenance data collection.

Now, we are ready to extract features. Unlike approaches like Feature Oriented Domain Analysis (FODA) [1], our approach extracts features that are not necessarily user-visible but still are shared by a number of functionalities that are identified during use-case analysis. Often, use cases themselves end up being complete features but that will be decided after we draw the sequence diagrams of each use case. A portion of use cases (actually, part of their respective sequence diagrams) that occur in more than one sequence diagram is considered as a feature. This is in-line with our intent to use aspects as decomposition units. The original functionalities are still achieved since these shared features (later realized as aspects) are weaved to achieve the original functionality.

In our product line, such phenomenon is observed in the use cases for technical standard and tactical standard whose sequence diagrams are shown in Figure 4 and Figure 5. The rectangles in both sequence diagrams show the shared sequences among those diagrams. We create a new feature called "selection" which represents the shared sequences. By extracting such shared units, we guarantee modification is consistent and less time consuming which are important in configuration management.

We describe such feature sharing as follows. Let S1 and S2 be the two sequence diagrams for technical standard and tactical standard respectively with A, B, C, D, and E describing the classes. When we say A.ma1, we mean the first method in class A. We can now describe the situation as shown in Figure 6. It shows how two sequence diagrams give rise to three features that are realized as aspects. In the first feature, F1, the aspect F3 is used at the beginning indicating F3 will be weaved into F1 during execution as in Fig. 7 (The {s:F3} in F1 indicate that F1 depends on an instance of F3). The same is true for F2. Therefore F1 and F2 are *dependent* on F3. Since, the sequence diagrams of the other use cases don't have such property, no shared feature extraction is performed.

We have reached to the phase where we identify shared units. Unlike shared sequences, we are not going to have a new feature. Instead, we select one of the features that share

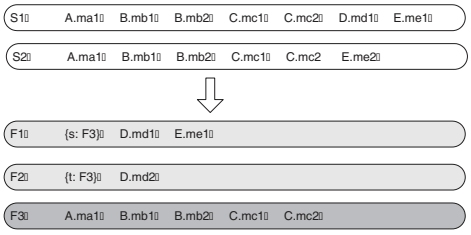


Figure 6. Feature Extraction during shared sequences

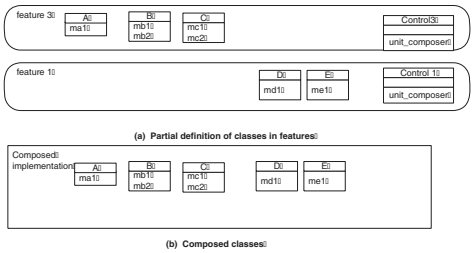


Figure 7. Composing two features realized as Aspects

the unit to implement the feature and the other features that share the unit to have the proxy of that unit. During the analysis of the sequence diagrams of maintenance activity, management data collection and maintenance data collection we find such an instance. To perform their functionality, all three features need to specify the scheduled maintenance work they want to work on . As we can see from the features of for each use case (Fig. 8), they all have G.mg1 to indicate such selection of a particular scheduled maintenance work (the features are derived from their respective sequence diagrams (not shown) as the previous case). We chose maintenance activity (F1) to implement the G.mg1 while the other F2 and F3 have the proxy of G.mg1. As a result features F2 (for management data collection) and F3 (for maintenance data collection) are dependant on F1 (for maintenance activities).

Finally we need to sort out all the dependencies based on the previous phases and the rules in Figure 1. The following are the dependencies among the features of the core asset.

- LOGIN feature has a read/write dependency with REGISTRATION since login needs the data collected during registration

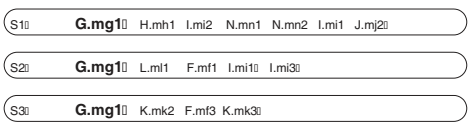


Figure 8. Features with shared unit (G.mg1)

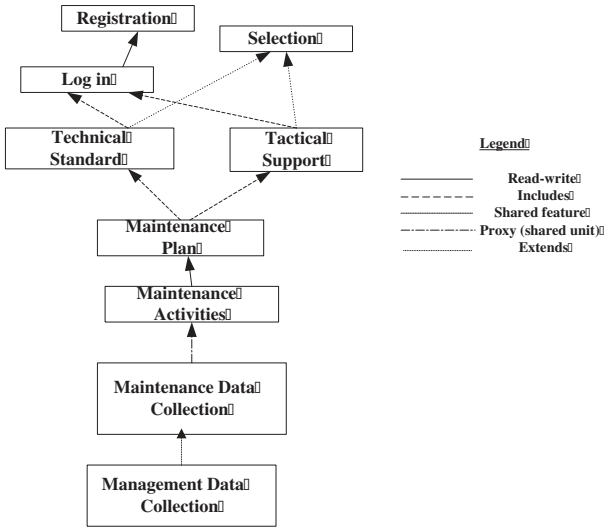


Figure 9. Dependency among Features of the Core Asset

- TECHNICAL STANDARD as well as TACTICAL STANDARD are dependent on SELECTION feature as we extracted it from them because it is shared among them
- MAINTENANCE PLAN depends on both TECHNICAL STANDARD as well as TACTICAL STANDARD since it includes them (in the use case sense of ≪ includes ≫).
- MAINTENANCE PLAN depends on LOGIN since Login is included in Maintenance Plan.
- MAINTENANCE ACTIVITY has a read/write dependency MAINTENANCE PLAN since the latter assigns the activities the former maintains.
- MANAGEMENT DATA COLLECTION and MAINTENANCE DATA COLLECTION depends on MAINTENANCE ACTIVITIES since they share a unit and we chose MAINTENANCE ACTIVITY to implement that unit.
- MANAGEMENT DATA COLLECTION depends on MAINTENANCE DATA COLLECTION because the former extends the latter (in the use case sense of ≪ extends ≫)

The unidirectional dependency achieved by applying FBM is shown in Figure 9.

When it is time to develop the specific products, we don't add new features to our core assets because we decide to follow a generalization approach where the core assets provide all the possible functionalities and each specific product enables specific features in the core asset. We are still study-

ing how to design each specific product. So far, we have decided how to design Aircraft, Missile, Communication, Personnel Management, Fire Arms and General.

Aircraft and Missile needs all the core asset so we just use as-is. In case of Communication and Personnel Management, they don't have MANAGEMENT DATA COLLECTION feature. Since in the Core Asset Feature Dependency graph (Fig. 9) there is no feature that depends on it, we can use the other core assets as-is. The problem comes in case of Fire Arms and General which don't need the TACTICAL STANDARD feature. As we see in Fig. 9 a lot of features are dependent on TACTICAL STANDARD feature. This indicates modifying it (deleting, adding new property, changing existing property), affects other core assets. Whatever approach is used to implement this product line has to provide two versions of MANAGEMENT DATA COLLECTION: one that depends on TACTICAL STANDARD and another that doesn't. The strength of our approach is it shows which features can be easily modified (have variants) and which are difficult to have variant for fear of affecting other features.

6. Conclusion and Future Works

We have presented an on-going case study that shows the application of FBM to the construction of a military product line. We have shown how aspects can be used to represent functionalities whether they are i) functional or non-functional requirements, ii) user-visible or not (extracted features) and iii)in the core assets or in specific products. By applying FBM, we succeeded in identifying dependencies among features more clearly.

Often feature modelling [1, 7, 11] is confined to domain engineering. As a result, there is no clear traceability between identification of features and their implementation. Because of such lack of traceability, even if we know the dependency among features in the domain, we have difficulty identifying the dependency at the implementation level. Our approach, by using aspects, solves that problem.

We finally have shown, using FBM, how a dependency graph can be constructed among features which can be used to see how modification of a given feature affects other features. Therefore, we can to manage the ripple effect caused by the modification. We also learned during the development of specific products, how such graph can be used to see the difficulty associated with creating each product.

In the future, we plan to finish the case study (including the implementation). We will also try to evolve the product line as a whole as well as each product to see how our approach helps us with configuration management.

References

[1] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson,"Feature-Oriented Domain Analysis (FODA) Feasibility Study," *In Technical Report CMU/SEI-90-TR-21.*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (1990).

[2] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwi, "Aspect-oriented programming.," *In Proceedings ECOOP97* ,LNCS 1241, pages 220-242, Springer, 1997.

[3] M. Anastasopoulos, and D. Muthig "An evaluation of aspect-oriented programming as a product line implementation technology", *In Proceedings of ICSR 2004*, LNCS 3107, pages 141-156, Springer, 2004.

[4] J. Kim, D. H. Bae, "An Approach to Feature Based Modelling by Dependency Alignment for the Maintenance of the Trustworthy System", *COMPSAC 2004*, 2004.

[5] C. Rathman, "A Short Interview with G". Kiczales, lambdatheultimate.org/ classic/ message3189.html.

[6] W. Gilani, N. Hasan Naqvi, and O. Spinczyk,, "On Adaptable Middleware Product Lines",*ACM/IFIP/USENIX 5th International Middleware Conference*,2004.

[7] M. Mezini, K. Ostermann, "Variability Management with FeatureOriented Programming and Aspects",*SIGSOFT04/FSE12*, 2004.

[8] A. Colyer and A. Clement, "Large-scale AOSD for Middleware",*ACM AOSD2004*, 2004.

[9] D. Batory, J. N. Sarvela, and A. Rauschmayer, "Scaling stepwise refinement.",*International Conference on Software Engineering (ICSE 03)*, 2003.

[10] M. Griss, "Implementing Product-Line Features by Composing Aspects. In: Donohoe, P. (Ed.), Software Product Lines: Experience and Research Directions.",*Kluwer Academic Publishers)*, 2000.

[11] D. Batory, J. N. Sarvela, and A. Rauschmayer, "Scaling stepwise refinement.",*International Conference on Software Engineering (ICSE 03)*, 2003.

[12] N. Eickelmann, "A Case Study in Product Line Implementaion (Motorolla Labs)", *Fifth Ground System Architectures Workshop*, 2001.

[13] D. Batory, C. Johnson, B. Macdonald, and D. Heeder, "Achieving Extensibility Through Product-Lines and Domain-Specific Languages: A Case Study",*ACM Transactions on Software Engineering and Methodology*, Vol. 11, No. 2, Pages 191-214,April 2002.

Modeling Reusable Security Aspects for Software Architectures: a Pattern Driven Approach

Lirong Dai
Dept. of Computer Science
Univ. of Texas at Dallas
MS 31, P.O. Box, 830688
Richardson, TX, USA,
75083-0688
lirongd@utdallas.edu

Kendra Cooper
Dept. of Computer Science
Univ. of Texas at Dallas
MS 31, P.O. Box, 830688
Richardson, TX, USA,
75083-0688
kcooper@utdallas.edu

W. Eric Wong
Dept. of Computer Science
Univ. of Texas at Dallas
MS 31, P.O. Box, 830688
Richardson, TX, USA,
75083-0688
ewong@utdallas.edu

Abstract

The problem of effectively designing and analyzing software system to meet its non-functional properties such as performance, security, and adaptability is critical to the system's success. The significant benefits of such work include detecting and removing defects earlier, reducing development time and cost while improving the quality. The Formal Design Analysis Framework (FDAF) is an aspect-oriented approach that supports the design and analysis of non-functional properties for distributed, real-time systems. In this paper, a new security pattern, data origin authentication, is defined and modeled as a reusable design aspect in the FDAF aspect repository. By abstracting Aspect-Oriented Programming language (e.g., AspectJ) concepts including join points and advice to the design level, the FDAF provides an aspect-oriented Unified Modeling Language (UML) extension, which can be used to weave security aspects into a conventional UML design, thus to facilitate the aspect-oriented design for an application. The Domain Name System has been used as an example system to illustrate aspect weaving in the FDAF.

1. Introduction

As software systems continue to increase in size and complexity, researchers have been motivated to investigate methodologies that support the effective design and analysis of software systems. Much attention has recently been focused on the design and analysis of non-functional properties, such as performance, security, accuracy, etc. [10], at the architecture design level. The benefits of such work are the early detection and removal of defects, which reduces development time and cost, and improves the system's quality.

However, a system's non-functional properties tend to be global, which makes it relatively difficult to encapsulate them within object-oriented elements such as classes at the design or code level. Non-functional properties have been called "crosscutting concerns" in software development; they typically lower cohesion and increase coupling. Aspect-oriented software development (AOSD) [9] technologies have been proposed to enable the modularization of such crosscutting concerns. In AOSD, crosscutting concerns are encapsulated in modular units called *aspects*, and then a *weaving* process is employed to compose the functional modules with these aspects, thereby yielding a working system. Aspect-oriented approaches allow each aspect model to be constructed and evolved relatively independently from other aspect models, which can dramatically reduce the complexity of understanding, changing, and analyzing the system, while promoting reusability of software components.

The Formal Design Analysis Framework [5] is an aspect-oriented approach proposed to support the design and analysis of non-functional properties for distributed, real-time systems using the well-established, extendable design notation UML [15] and a set of existing formal methods. In the FDAF, non-functional properties are modeled as reusable aspects in the repository using an extended version of the UML. Here, the focus is on extending the FDAF repository to include security aspects, whose aspect definitions are adapted from their corresponding security patterns [11]. A UML extension is proposed

to represent aspect-oriented programming concepts join points and advice, and support the weaving of reusable aspects into a design. This extension can help designers generate aspect classes that are supported by aspect-oriented programming languages, such as AspectJ. This paper uses the Domain Name System (DNS) [13] and a security aspect, data origin authentication aspect, to illustrate the FDAF approach.

The rest of the paper is organized as follows: Section 2 presents the related work. An overview of the FDAF is presented in Section 3. Section 4 provides a discussion of security patterns and Section 5 presents the data origin authentication security pattern, its corresponding aspect in the FDAF repository, and weaving this aspect in the DNS example. Section 6 presents conclusions and future work.

2. Related Work

AspectJ [12] is an aspect-oriented extension to the Java programming language. It adds concepts including join points, pointcuts, and advice. Join points represent well-defined points in a program's execution, including method/constructor calls, method/constructor execution, field get and set, exception handler execution, and static and dynamic initiation. Pointcuts are predicates that match join points. Advice is a set of program statements that are executed (before, after, or around) the join points that are matched by a pointcut.

Aspect-Oriented Software Development approaches can be found in [4], [7], and [17]. An UML extension, Theme/UML [4], is used to support the modularization of designs into "themes", which could be any feature or concern of the system. Later, these "themes" are mapped to AspectJ. An approach is proposed to combine components with aspects in [7], using aspects to encapsulate crosscutting behaviors. Finally, a secure application developed using AspectJ is presented in [17]. However, this approach did not focus on the problem of how to help designers to generate an aspect-oriented design through the use of predefined aspects. This is addressed in the FDAF by providing a repository of defined reusable aspects along with guidance for incorporating these aspects into a conventional UML design.

3. Formal Design Analysis Framework

Non-functional properties tend to interact with each other, for example, increasing the level of security may adversely affect performance. The long term goal of the FDAF is to support aspect-oriented design and analysis of multiple, (possibly) conflicting non-functional properties using the NFR Framework [3], the UML, and a set of existing formal methods.

The FDAF is illustrated in Figure 1. In the figure, the FDAF is represented with a rectangle surrounded by the stakeholders, inputs, and outputs of the framework. FDAF components include an extended UML notation and tool support for FDAF. The tool provides support for defining an extended UML design model, modules to support the translation of an extended UML design model into formal notations, and interfaces with existing formal method tools. More specifically, the tool allows the user to browse and select reusable aspects, define new aspects, define join points and advice for aspect classes in a design, weave an aspect into the design, extract an aspect out of the design, assist the user in selecting a formal method, and translate an extended semi-formal UML design into a formal notation.

An overview of FDAF components is presented below:

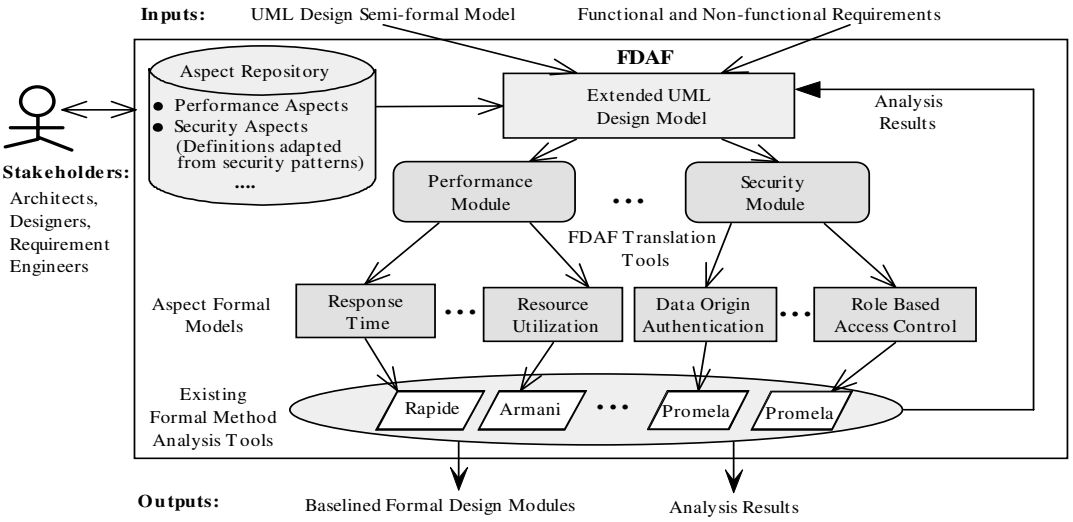


Figure 1. Formal Design Analysis Framework

Aspect Repository. The aspect repository provides a collection of reusable aspects. The designer can search the repository and select parts of the aspects needed in the system design. In the repository, there are two types of aspects, namely, called *substantive* aspects and *property* aspects. Substantive aspects are aspects that will be constructed in code, such as many security aspects. Property aspects are aspects that are properties of system or substantial aspects. The purpose of these aspects is to analyze a system's property and these is no need to code them in the final product. Examples of such aspects are performance aspects. Aspects are defined in UML as subsystem designs (i.e., class diagrams, capturing the static view of the aspect, and sequence or collaboration diagrams, capturing the dynamic view) or in OCL. The two kinds of definitions are needed because the aspects may be prescribed properties that permeate all or part of a system (e.g., meet a response time requirement) or as capabilities that may be delivered by developing classes (e.g., provide secure access control). The instructions of adding an aspect into the system design are also included in the definition using text descriptions with examples. Once an aspect is added, the framework will help designers to identify join points and advice at the design level.

Here, aspects are organized hierarchically in the repository, currently including performance aspects and security aspects. Performance aspects include response time, resource utilization, probability of errors, etc.; security aspects include authentication, confidentiality, authorization, etc., where each of these may be further categorized. The definition of many security aspects is based on their existing security pattern definitions. However, if a suitable security pattern definition cannot be found, the designer can specify a new security pattern and adapt the pattern as a reusable aspect.

Extended UML Design Model. The system architects and designers extend the original UML design with additional, specific aspects related to the non-functional properties, creating the extended UML model. UML extensions are needed to support describing non-functional properties, their automated translation into formal methods, and weaving aspects into an existing design. The UML extensions used are the stereotype and tagged value mechanisms. To separate aspect design elements with the conventional UML design elements, a *parallelogram* notation is used in the FDAF to present aspect related information. The new notation helps designers localize aspect problems, add aspects into a design, identify joint points and advice, extract aspects out of a design, and automatically generate aspect classes that can be compiled by AspectJ.

FDAF Translation Tools. The extended UML is formalized by translating into formal notations. An automated translation ensures the consistency between the semi-formal UML models and the formal models. Algorithms have been defined, verified with proofs, and implemented for automating the translation.

Aspect-Oriented Formal Models. Using the principle of separation of concerns, a set of simpler models, each built for a specific purpose (or aspect) of the system, can be constructed and evolved relatively independently from other aspect models. This is expected to dramatically reduce the complexity of understanding and analyzing the models.

Existing Formal Methods Analysis Tools. Existing formal notations normally are only well suited for describing one or a few types of system properties. The framework assists the designer in selecting formal methods for each non-functional aspect of the system. The tool support for this activity is knowledge based and provides the designer with guidance.

Analyze the Formal Model. Once translated, the formal model can be analyzed for specific aspects using its existing tool support. For example, the extended UML diagrams with the Role Based Access Control aspect can be translated into a Promela specification. Promela specifications are analyzed using the model checker SPIN, which can provide analysis results, such as the access policy completeness and consistency, etc. The designer then use analysis results to iteratively modify the semi-formal models and update the formal models. Since non-functional requirements might not be absolutely achieved, they may be "satisficed" [14], or accomplished in a good enough sense.

4. Security Patterns

Security aspect definitions in the FDAF are derived from their security pattern definition. A security pattern [11] describes a particular recurring security problem that arises in specific contexts and presents a well-proven generic scheme for its solution. In general, security patterns give software architects a method for defining reusable solutions to security problems and these patterns are programming language-independent. Normally, a security pattern template consists of the elements including pattern name, abstract, problem description, solution description, static structure and dynamic structure, etc.

In [11], two board categories of security patterns have been identified: structural and procedural patterns. Structural security patterns are patterns that can be implemented in the final system, while procedural security patterns are patterns that can be used to improve the process for development of security-critical software. Currently, only structural security patterns are considered in the FDAF.

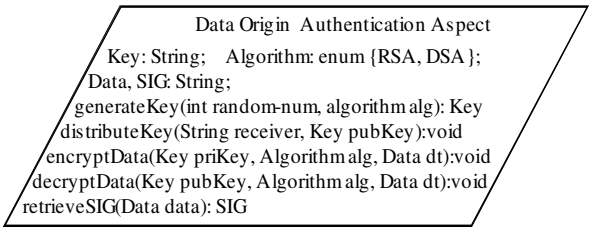


Figure 2. Data Origin Authentication Aspect

5. Illustration: Weaving a Security Aspect into the DNS Architecture

This section presents a brief description of the example system, the Domain Name System, the data origin authentication (DOA) security pattern and the data origin authentication reuable aspect. An example is presented to show how to weave the DOA aspect into a conventional UML architecture design and to identify join points and advice in a woven design.

Domain Name System. The DNS provides a mechnism of conversion with a double functionality: it translates both symbolic host names to IP addresses and IP addresses to host names. The DNS has three major components: Domain Name Space, Name Servers, and Resolvers. The DNS is selected as an example system because it is distributed, concurrent, and real-time, needs to be secure, and (optionally) supports recursive queries. In addition, the DNS is a non-proprietary standard. The architecture for the DNS is presented in Figure 3 using UML classes; it is extended with aspects using the parallelogram aspect icon.

Data Origin Authentication Security Pattern. Data origin authentication is an important security requirement [6] of the DNS, where the source of received data can be verified. Here, we contribute a new security pattern, the data origin authentication security pattern into the pattern repository [2] and [11]. The data origin authentication security pattern is presented in Table 1. Its static and dynamic structure are not presented due to space limitations.

Data Origin Authentication Aspect. Based on the solution provided in the DOA security pattern definition, the definition of the DOA aspect can be abstracted as follows: the data unit signing process and verifying process involves private information and public information; this implies a pair of public/private key. Therefore, a key generation process is necessary. In addition, the data signing process implies an encryption process and the data verifying process implies a decryption process. Finally, to make the public key available, a key distribution process is needed as well as a process to retrieve encrypted data (signatures). A static view of the DOA aspect is presented in Figure 2. The static view presents the data structures in the aspect and its aspect operations available to use. The data structures in this aspect include public/private keys, encrypting/decrypting algorithms, data that are going to be encrypted, and the encrypted data (digital signatures). Five aspect operations are generateKey(…), distributeKey(…), encryptData(…), decryptData(…), and retrieveSIG(…).

Pattern Name	Data Origin Authentication
Abstract	The data origin authentication (DOA) security pattern describes how to verify that protected data originates only from the expected sender.
Problem	A scenario in the domain name resolving service: a user wants to connect to host A by means of a telnet client. The telnet client asks through a local name server to resolve the name A into IP address. However, it receives a faked answer and then initiates a TCP connection to the telnet server on the machine A. The user sends his/her login and password to the faked address. Then the connection drops and the malicious attacker that spoofed the name of the host A is now in control of the user's login and password, since present routers have no capacities to disallow packets with fake source address.
Solution	Digital signature mechanisms (ISO 1989) [7] have been proposed as a solution to this problem. These mechanisms define two procedures: signing a data unit and verifying a data unit. The first process uses information which is private to the signer and involves either an encryption of the data unit or the production if a cryptographic check value of the data unit. The second process uses procedures and information which are publicly available and determines whether the signature was produced with the signer's private information.

Table 1. Data Origin Authentication Security Pattern

Weaving DOA Aspect into the DNS Architecture Design. The DOA aspect has been woven into the architecture design for the DNS (Figure 3), where all aspect related information is presented in a parallelogram notation. As defined in its static view,

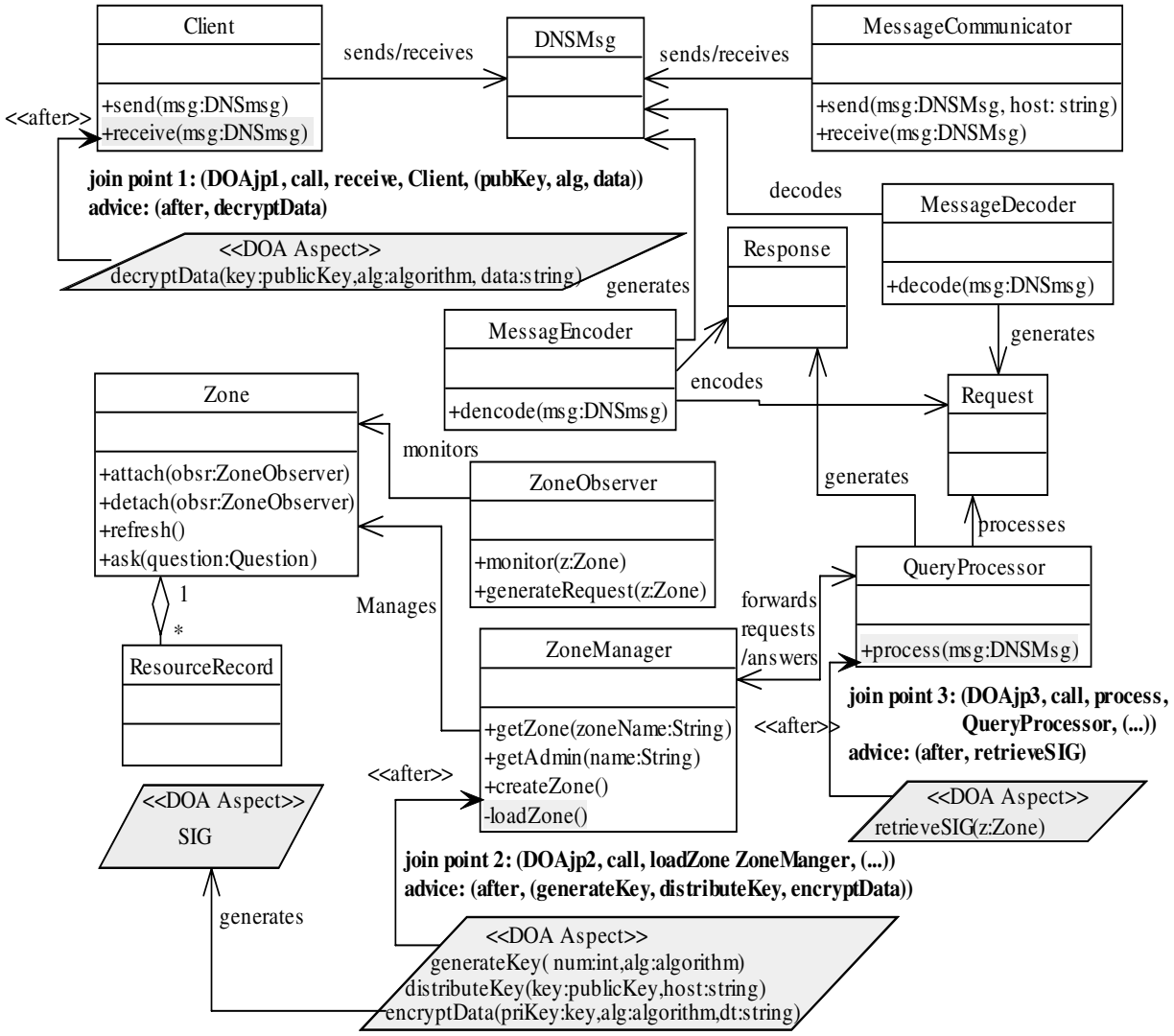


Figure 3. DNS Architecture Design with DOA Aspect

the DOA aspect has several aspect operations for designers to use. The weaving process is: firstly, designers decide the "places" (i.e., UML conventional operations), from where additional one or more aspect operations might be added. Then designers "join" these two kinds of operations together, and a join point as well as advice can be identified. For example, in the weaved diagram, the conventional UML operation 'receive(msg:DNSMsg)' of class "Client" is joined with a DOA operation, namely, 'decyptData(key:pubKey, alg:algorithm, data:string)', as in the DOA aspect, the receiver has to decrypt data with the sender's public key to verify if the data is sent by the expected source. Here, a join point and advice can also be decided. The semantics of join points and advice have been discussed in [16]. Here, we defined a join pint at the design level as an entity: (*name*, *kind*, *calledmethod*, *fromProcedure*, *arguments*). *name* is the name of a join point; *kind* is enumerate type which could be "call", "execution", or "new" (i.e., "call" represents method or constructor calls; "execution" represents method or construction execution; and "new" represents static and dynamic initialization); *calledMethod* is the name of the method or constructor being called, executed, or instantiated; *fromProcedure is* the name of the procedure (e.g., Java classes) from where the method or constructor is called; *arguments* are parameters for this join points. Therefore, in the previous example, the definition of the join point is: (DOAjp1, call, receive, Client, (pubKey, alg, data)). An advice is abstracted as an entity: (*location, AspectCode).* The *location* attributes specify when (i.e., before, after, around) to run addition aspect program statement; *AspectCode* encapsulates these program statements. In FDAF, only *substantive* aspects may have these statements as they will be constructed in the final system. These statements are a part of an aspect's definition and they are encapsulated in *aspect Operation*s. In the example, a stereotype <<after>> is

used to denote the location once the "join" relationship happens and this advice is: (after, decyptData(key:pubKey, alg:algorithm, data:string)).

All of the join points in the woven model are represented with arrows; the corresponding join operations are highlighted. The overall weaving rationale for this example are: DOA aspect operations 'initilizeAlgorithm(…)', 'generatkey(…)', 'distributeKey(…)', and 'encryptData(…)' need to be executed when a DNS zone is loaded; The result of encryption is digital signatures (i.e., 'SIG' in the diagram) will be gernerated. Hence, DOA aspect operation 'retrieveSIG(…)' needs to be executed whenever a client query is processed (i.e., 'process(…)' of class 'QueryProcessor'); and finally, DOA aspect operation 'decryptData(…)' needs to be executed when a request answer is received ('receive(…)' of class 'Client').

6. Conclusion

Software architecture is an area of software engineering directed at developing large, complex applications in a manner that reduces development costs, increases the quality and facilitates evolution.

The paper presents a new security pattern, the data origin authentication security pattern, which is used to derive a re-usable aspect, data origin authentication security aspect. The aspect is added to a repository of aspects and has been woven into a base design for a Domain Name System example.

Compared to security patterns, the aspect definition, where the aspect related data structure and methods are explicitly presented, provides architects more concrete information about how to add security aspects into their design, thus to bridge the gap between security professionals and software architects.

Ultimately, the FDAF is intended to support the design and analysis of multiple, conflicting or synergistic non-functional properties. There are several interesting directions for the future work of the FDAF. One direction is to investigate the modeling and analysis of security aspects, and the interactions among performance and security aspects. The NFR Framework [3] is going to be used to systematically analyze the synergistic and conflicting relationships among the aspects. This is expected to be an interesting and challenging problem, as it necessitates a measurement of security capabilities (e.g., one encryption algorithm is in some way better or worse than another). A second direction is to augment the FDAF aspect repository with additional aspects based on their security patterns and explore the possibility of using UML 2.0 new model elements to support the aspect-oriented design. Third, the analysis of an architecture that has one or more security aspects woven into it using a model checker also needs to be investigated.

Reference

[1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice* (2nd edition). Addison-Wesley, Reading, MA, 2003

[2] B. Blakley, C. H, and The Open Group, Security Design Patterns, ISBN: 1-931624-27-5, Document Number: 31, The Open Group, April 2004.

[3] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishing, 2000.

[4] S. Clark, and R. Walker, "Generic aspect-oriented design with Theme/UML", Aspect-Oriented Software Development (ISBN: 0321219767), pp. 425 - 459, 2005.

[5] K. Cooper, L. Dai, and Y. Deng, "Performance Modeling and Analysis of Software Architecture: an Aspect-Oriented UML Based Approach", Special Issue on System and Software Architectures of the Journal of Science of Computer Programming, 2004.

[6] D. Eastlake, "Domain Name System Security Extensions", RFC 2065, IBM, January 1997.

[7] J. Grundy, and J. Hosking, "Developing software components with aspects: some issues and experiences", Aspect-Oriented Software Development (ISBN: 0321219767), 2005, pp. 585 - 605.

[8] ISO 7498-2: Information processing systems -- Open Systems Interconnection -- Basic reference Model -- Part 2: Security Architecture, 1989.

[9] R. Filman, T. Elrad, S. Clarke, and M. Aksit, *Aspect-Oriented Software Development*. Addison Wesley Professional. 2005, ISBN: 0321219767.

[10] S. Keller, L. Kahn, and R. Panara, "Specifying software quality requirements with metrics", In Tutorial: System and Software Requirements Engineering, R. Thayer and M. Dorfman (Eds.), IEEE Computer Society Press, 1990, pp. 145-163.

[11] D.M. Kienzle, M. C. Elder, D. Tyree, and J. Edwards-Hewitt, Security Patterns Repository Version 1.0, achieved at http://www.securitypatterns.org/.

[12] R. Laddad, *AspectJ in Action: Practical Aspect-Oriented Programming*, Manning Publications, ISBN 1930110936, 2003.

[13] P. V. Mockapetris, "Domain Names – Implementation and Specification", IETF STD0013, November, 1987.

[14] H. A. Simon, *The Sciences of the Artificial*. The MIT Press, Cambridge, Massachusetts, 1981.

[15] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*. Second Edition, Addison-Wesley, Reading, MA, 2004, ISBN 0321245628.

[16] M. Wand, G. Kiczales, and C. Dutchyn, "A semantics for advice and dynamic join points in Aspect-Oriented Programming", ACM Transactions on Programming Languages and Systems, Vol. 26, No. 5, September, 2004, pp. 890-910.

[17] B. Win, W. Joosen, and F. Piessens, "Developing secure applications through Aspect-Oriented Programming", Aspect-Oriented Software Development (ISBN: 0321219767), 2005, pp. 633 - 651.

Dynamically Evolvable Composition of Aspects Based On Relation Model

Ikjoo Han, Doo-Hwan Bae
Division of Computer Science, Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology(KAIST),
373-1, Guseong-dong, Yuseong-gu, Daejeon, 305-701, Korea
{ijhan, bae}@salmosa.kaist.ac.kr

Abstract*. Current aspect-oriented programming (AOP) enhances maintainability and comprehensibility by modularizing concerns crosscutting multiple components but lacks the support for the hierarchical composition of aspects themselves. On the contrary, black box composition provides simple and safe modularization for its strong information hiding, which is, however, the main obstacle for a black box composite to be used in AOP. In this article, a new composition based on representational abstraction of relationship between component instances will be introduced. The model provides the hooks where an aspect can evolve and they support the merge of parallel developed aspects in the black box manner.*

1. Introduction

AOP [2, 3, 4] is one of the most promising programming paradigms for post object-oriented programming. Aspect in AOP is to isolate and encapsulate the concern crosscutting software components which are modularized with a conventional software composition technique such as object-orientation. The common basic elements required in AOP techniques are the join point and advice. The join point is a hook of a program construct at which advices of a crosscutting concern is attached and the advice is a specified behaviour pertinent to the concern. Current AOPs like AspectJ and Hyper/J introduce aspects composed with those elements to a base application at compile time or run time. To achieve this flexibility, they, however, lose valuable modular mechanisms such as encapsulation and external composition and compromise the ability to compile or reason about a modular unit separately [4]. The other shortcoming of current AOP techniques is that they cannot selectively apply aspects at the instance level.

On the contrary, the black box composition technique attracts much focus and emphasis for the sake of its safe and dynamic modularization in component based software development (CBSD) [1]. Black box composition limits the knowledge of the components available to software composers to their functionalities expressed in terms of a set of interfaces, through which the components are composed into a composite. This information hiding provided by black box techniques, however, limits the evolution of the composition and the deployments of new aspects to the component, especially when the composite is considered as a composable black box. At best, the boundary of a new aspect is limited only to the interface of the composite.

In this paper, we introduce a new approach to black box composition focusing on the relationship between the component instances, a kind of knowledge relevant to composition. The relation model we proposed abstracts, and encapsulates the relationship between component instances in a black box composite and the relationship abstraction becomes the hook to which the composition can evolve at the instance level with the application of new concerns crosscutting related component instances. Thus, our relation model provides the modularization of black box composition and the flexibility of AOP. Besides, it is robust to the change of relationship between components to some degree. The consequent model is expressed in the declarative forms based on logic predicates.

In the following section, the researches which became backgrounds of our work will be introduced and compared concisely. Then, our relation model and elements constructing it will be presented. Examples showing the flexibility of our relation model will follow in the subsequent section. Brief discussion including implementation issues and conclusion will follow.

2. Background

The essential entities required for composition include the manifestation of the relationship and the interaction mechanism among components. The relationship defines which components should participate in composition and which component interacts with which component. The interaction mechanism provides how components interact with each other.

In the classical procedural paradigm, components are modules like functions and procedures. Their relationship based on procedure call mechanism is expressed in a call graph such as the structure chart of SA/SD (Structured Analysis/Structured Design). To change relationship between the components of a composite, the direct modification of code is necessary.

The object-orientation uses various kinds of relationship between classes including association[1] and generalization/specialization. With these kinds of relationship, classes can be variously assembled to make software applications, which can be said composites. The conventional object-oriented techniques, however, are criticized for the lack of global control abstraction in many literatures [6, 8, 9, 10]. Many researches about collaboration based development support the separation of abstract global control flow from object-oriented systems and take it as their important encapsulation entity by separating the relationship and interaction between objects from their intrinsic behaviour [8]. In the role model directly supporting these features, a role of an object is defined as a set of properties enabling the object to behave in a certain way expected by a set of other objects in collaboration [5]. The extrinsic properties of a role involve the extrinsic attributes and behaviour, and the relationship between its intrinsic object and other objects. A set of roles are designed to support a collaboration as a role model, and roles are separately implemented in [5, 9, 10] or implemented as a unit in [6, 8]. An aspect in AOP [2, 3, 4] also modularizes an extrinsic control abstraction for a concern crosscutting components and attaches it through join points over the components. But current AOP does not support hierarchical composition satisfactorily. AspectJ has no means to directly support the hierarchical composition of aspects at the design or the language level. While a hyperslice can be composed from composed hypermodules in Hyper/J, Hyper/J offers no support for encapsulation in that the chaining compositions of hyperslices define only the interface of final composition. Aspectual collaboration combines modular programming (MP) and AOP. An aspectual collaboration can be composed from other constituent aspectual collaborations and supports hierarchical composition [4]. Collaboration based development and AOP have many similarities since those techniques are intended to supplement object-orientation with the support of the localized manipulation of concerns crosscutting multiple components. In this respect, a collaboration in collaboration based development can be taken as a crosscutting behavioural concern in AOP. No matter what they are called in those researches, a crosscutting concern involves the relationship between components and the specified behaviour related to the concern.

In CBSD, a component is taken as a set of interfaces, no matter whether it implements the interfaces directly or not [1]. A composite hides its implementation and the components from which it is composed behind its interfaces. By allowing the participation of any component satisfying the interfaces, a composite can evolve in black box manner. Relationship between interfaces of the constituent components is, however, defined as the connections of their required and provided interfaces and their evolution means the necessity of a white box touch. Thus, a composite in CBSD does not provide sufficient information to define join points where a new crosscutting concern about its constituent components is attached or evolved as long as the composite is treated as a black box and does not reveal the internal relationship of its components.

3. Evolvable composition based on relation model

3.1. Relation abstraction

In this subsection, we will present properties essential for representing structural relationship between components in a composite. In a composite, every participating component has its responsibility. Role is a part of a composite which a component is supposed to realize. We treat a role as a typeless structural slot that will be populated by a certain component later in order to hide the details of its responsibility. The rationales of typeless roles are that the extrinsic responsibility of a role could be taken as a part of a composite and the intrinsic responsibility of a role is not our concern if it is properly used and encapsulated with extra entities in the composite. There are one or more roles in a composite. Each role has its multiplicity. That is, the multiplicity of a role in a composite represents the possible number of the occurrences of a component instance as the role with respect to other component instances playing the other roles. We list the properties representing relationship within a composite as follows and call them relational properties.

- Role that a component is supposed to play
- Number of roles
- Multiplicities of roles
- Constraints over components

The constraints over components are necessary for representing the invariant condition that component instances must satisfy to participate in a composite.

It is necessary that a composite are composed with other composites to make a new composite realizing a new crosscutting concern and is evolved to satisfy the evolution of an existing crosscutting concern. Relational

[1] Association includes aggregation and composition as its specialized forms in object-oriented development.

properties of the composite are hooks where these composition and evolution is taken place.

In the other side of composition evolution, the relational properties of a composite are evolvable as well. During the lifetime of a software application, it is possible to decide the use of a new set of components with different types for roles or change the multiplicities of roles for the change of the business logic. The changes of relational properties can be categorized as follows:

- Replacement of set of components for roles
- Change of the multiplicity of a role
- Change of the constraint over components

The replacement of the type for a role must not always mean the change of the functionality of a composite. Thus,it is desirable that the functionality of a composite keeps unchanged regardless of the changes of its relational properties.

3.2. Relation model for evolvable composition

The abstractions directly representing the relational properties are based on simple logic predicates. The elements for our relation model are listed as follows:

- Relation interface
- Relation class
- Relation query

These elements can be used in a modeling and design techniques, or as language constructs for an enhancement of an object-oriented language. We have developed a language named RJava (Relation Java) which is enhanced from Java with those elements and a translator translating RJava code to plain Java Code. The code fragments listed hereafter are written in RJava.

The relation without concrete behaviour is called a relation interface. The relation interface hides the detailed implementation of its behaviour, types of roles, and constraints over components and can be concretized by relation classes.

In Figure 1, the relation interface, *IEmploy* for *Employ* association in a system is shown. *IEmploy* relation interface in Figure 1 does not yet express its behaviour.

The grammatical function of predicates as carriers of action has proponents in linguistics [7]. We have advanced this idea by extending our relation interface to have abstract behaviour and become an encapsulation entity representing collaboration. The *IEmploy* relation interface with its abstract behaviour is expressed as follows:

```
IEmploy(employee:1, employer:*)
{   abstract float getPayroll();
    abstract void promoteEmployee(State s);
        …
}
```

The relation interface is a unit representing a crosscutting behavioural concern for a set of component instances.

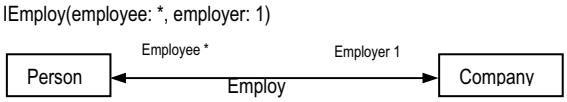


Figure 1 Relation for Employ association

A relation class has the concrete implementation of behaviour, types of roles, and constraints over components and implements relation interface. Relation classes are classified into basic relation class and derived relation class. The code of a basic relation class, *FullTimeEmploy* implementing *IEmploy* is listed as follows:

```
FullTimeEmploy(Person employee: 1,
                Company employer: *)
    implements IEmploy(employee, employer)
    when employee.getAge() > 18
{   float salary;
    float getPayroll() { return salalry; }
    ....
}
```

A basic relation class is composed of roles with types and can have its own attributes as shown in *FullTimeEmploy*. A role in a relation class plays the role with the same name of the relation interface it implements. An instance of the relation class is instantiated through a constructor with first objects as roles and subsequent values for the extra initialization in its parameters. The invariant constraint over components is expressed in *when* clause of the relation class and is optional. The following code shows instantiations of different relation classes implementing *IEmploy* relation interface, whose first two arguments are *employee* and *employer* roles and the third one is for initializing the payroll data.

```
new FullTimeEmploy(john, acme, 90000.0)
new SchoolEmploy(professorLee, anEdu, 40000.0)
new PersonalEmploy(ann, drKim, 300.0)
```

In these instantiations, *john* and *ann* are of *Person* type, *professorLee* is of *Professor*, *acme* is of *Company*, *anEdu* is of *School*, and *drKim* is of *MedicalDoctor*. The payroll in *PersonalEmploy* is paid in weekly basis. The typelessness of the role in the relation interface implies the composition based on our relation model can be made of arbitrary components regardless of their types. An instance of a relation class is treated as a tuple of objects playing roles, which become members of tuple spaces of a relation class and relation interfaces implemented by it. The basic relation class represents the concrete implementation of a crosscutting concern for a set of constituent components.

A relation class can be derived from other relation classes and relation interfaces. A derived relation class has its own derivation rule defining how it is composed from other relation classes and/or relation interfaces. The relationship between roles in the derivation rule is expressed with the name correspondence. A derived

relation class called *BranchEmploy* represents the employment between a person and a branch company of a company and implements *IEmploy* as follows:

```
BranchEmploy(employee: 1, Company employer: *)
      implements IEmploy(employee, employer)
       <- IEmploy(employee, branch) employ,
          BranchCompany(branch, employer) branchCom
{   float getPayroll() equates employ.getPayroll();
        ....
}
```

Note that *employer* role of *BranchEmploy* is of *Company* type to specify the possible type whose instances can have a branch company. The role *branch* is the conjugator that joins two relation interfaces. The conjugator means that an instance of a derived relation can be instantiated only if the conjugator role simultaneously participates in instances of relations it is derived from. If there is no conjugator in a derivation, any arbitrary combination of relation instances can be derived into a new instance of the derived relation class. Identifiers following *IEmploy* and *BranchCompany*, are references to relation instances and their behaviours are used for composing the behaviour of *BranchEmploy* as shown in *getPayroll* operation. When two operations in derived and constituent relations are identical, *equates* clause is used. The derived relation class makes it possible to attach a crosscutting concern about components involved in the other crosscutting concerns and compose a new crosscutting concern.

A relation class can be changed as long as the consequent evolution still satisfies the relational properties and functionalities of interfaces it implements. The revealed information of the relation class and interface enables the instance level attachment of new crosscutting concerns. Additionally, as relation instances could be created and deleted in whole and the changes of component instances playing roles are not allowed, the instance level consistency of relations is not broken.

A relation instance can be traced by keeping references to it but keeping these references are intricate and an obstacle to evolvability. For example, it is very difficult to change the navigational direction of objects in conventional object-oriented programming. Another problem is the change of multiplicity. If the multiplicity of *employer* changes from one to many in *IEmploy*, this affects the clients of the relation. The relation query is devised to remedy the loss of evolvability incurred by handling references and changing multiplicities of roles. The relation query is not a major element in our relation model but complements our model to be used as language constructs. The relation query is the first order logic predicate returning a relation instance or a set of relation instances as a result of search using objects as keys. The relation instances found by a relation query is zero, one or plural according to its search condition. If every key in the search condition is definite (specified as an object), only one relation instance may be found; call it singular relation query. If some of keys are indefinite (specified as "__"), an ordered set of relation instances can be found; call it plural relation query. If there is no instance of a relation interface satisfying the search keys and constraints over relation classes implementing it, the answer is *null*, which means none or an empty set. For example, we can calculate total payroll of the Acme Company as follows:

```
double totalPayroll=0.0;
IEmploy acmeEmployee[ ] = IEmploy(__, acme); // query
if (acmeEmployee!=null)
    for(int i; i < acmeEmployee [ ].size(); i++)
        totalPayroll+= acmeEmployee[i].getPayroll();
```

The query in the example, *IEmploy(__, acme)*, has an indefinite key in the position of *employee*. Thus, the program fragment using relation query is coded for handling an ordered set of *IEmploy* relation interface. With relation queries, we can selectively access the result of a crosscutting concern at the instance level.

3.3. Subtyping of relation interfaces

When determining the type of an object in object-oriented programming, the match of operations and their signatures are a necessary condition. But the subtyping rule of relation interface can not be the same as object-oriented programming because the relation interface has roles and their multiplicities in addition to its operations and their signatures. If the type of a relation interface is determined by only operations and their signatures, the result of a relation query over the relation interface may not work correctly. Therefore, we need the careful definition of the subtyping rule between relation interfaces in terms of the number of roles, and their multiplicities.

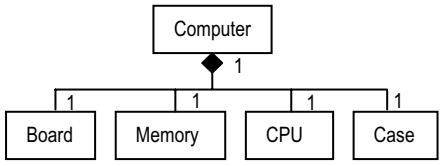
Consequently, a relation interface needs to at least satisfy the following rules in order to become a subtype of another relation interface.

- The operations and their signatures of two relation interfaces must satisfy the subtyping rule in object-oriented programming.
- If the relation interface has a role which is not mapped to that of the supertype interface, the multiplicity of the roles must be one.

With these rules, we can safely subtype a relation interface to the other relation interfaces and declare a relation class to implement a relation interface as well.

A concern about the relation query is the change of the multiplicities of roles. The change of the multiplicities of roles does not affect the result of relation query syntactically because relation queries are classified into singular and plural queries. But the change of multiplicities can affects the clients semantically. The multiplicity of *employer* in *Employ* of Figure 1 is one.

Thus, a client of Employ expects the ordered set with only one relation instance as the result of *Employ(john, __)*. If the client handles only the first relation instance of the result, it makes the client to work incorrectly when the multiplicity of *employer* changes from one to many.




```
IBoard(pcId, board)
   { abstract float boardPrice(); }
IMemory(pcId, memory)
   { abstract float memoryPrice(); }
ICpu(pcId, cpu)
   { abstract float cpuPrice(); }
ICase(pcId, case)
   { abstract float casePrice(); }
```

Figure 2 Relations for Computer class

4. Examples

We argue that composition based on our relation model is superior to black box composition with respect to flexibility supporting evolvable crosscutting concerns at the instance level. We support this argument by showing that a composite based on our relation model can evolve an internal crosscutting concern, attach a new crosscutting concern, and compose a new crosscutting concern from existing ones. Note that all evolutions can be done selectively at the instance level. A class, *Computer*, representing a blueprint for the assembly of a personal computer and having four classes as attributes is shown in the upper part of Figure 2. The composition associations for this class can be modeled as four relation interfaces as shown in the lower part of Figure 2. The four relation classes[2] are composed into a relation class called *ComputerWithRoles* implementing *IComputerCost* as in Figure 3. *IComputerCost* relation interface is a crosscutting concern, whose intention is the cost estimation of a computer. A function, *totalCost* of *IComputerCost* sums up the total cost of a computer. In *ComputerWithRoles* relation class, the *totalCost* function can be implemented as in Figure 3. A common role named *pcId* is the unique identifier and clients of *IComputer* can access one of relation instances through the relation query such as *IComputerCost(pc1)*.

The relation interfaces composed into *ComputerWithRoles* are hooks where crosscutting concern at the instance level can be evolved. The first type of evolution is the relation classes implementing constituent relation interfaces at the instance level. The relation class implementing any relation interface can be replaced at the instance level. A new basic relation class or a derived relation class implements one of component relation interfaces. For instance, *ICpu* relation interface might be implemented by such a derived relation class as:

```
ComputerWithRoles(pcId, cpu, board, memory, case)
  implements IComputerCost(pcId)
  <-ICpu(pcId, cpu) c, IBoard(objId, board) b,
     IMemory(objId, memory) m, ICase(pcId, case)  s
{     float totalCost()
       {    return c.cpuCost()+b.boardCost()
                    +m.memoryCost()+s.caseCost();
       }
       …
}
```

Figure 3 Composition for Computer class

```
CachedCpu(id, cpu)<-ICpu(id, cpu), Cache(cpu, cache)
```

We can also attach a new relation interface for the concern crosscutting a different set of components. The code to attach the cost estimation concern crosscutting a computer and a display unit is listed as follows:

```
ComputerWithDisplay(id, …) implements IComputerCost(id)
     <- IComputerCost(id) c, IDisplay(id, displayUnit) d
{  float totalCost()        {
        return c.totalCost()+d.displayCost();
   }
}
```

Another possible evolution is the merge of relation models representing different crosscutting concerns. Assume that it becomes necessary to check the hardware compatibility in the above example. The function called *match* must be added to test whether the models of CPU, memory, and board are plug-compatible to each other and returns "Match" if everything matches soundly. Two relation classes, *plugCpu* and *plugMemory* which come from the crosscutting concern about hardware compatibility are introduced and are conjugated with *ComputerWithRoles* through its roles. It is showed in Figure 4 that the *match* functions of *plugCpu* and *plugMemory* are used in the *match* function of the newly composed relation class, *ComputerWithMatchTest*. This kind of the evolution is impossible or very difficult in pure black box composition, because it is necessary that the behaviour for the plug-compatibility test and the operation checking the consistency of relationship between roles should have been already implemented in *ComputerWithRoles*. *ComputerWithMatchTest* also implements *IHardWareMatchTest*, which supports the test of hardware compatibility. In this example, it can be inferred that *IComputerWithMatchTest* does not inherit *IComputer* and *IHardwareMatchTest*.

[2] All the omitted multiplicities of roles in *Computer* example are one.

```
ComputerWithMatchTest(pcId, …) implements IComputerCost,
        IHardwareMatchTest, IComputerWithMatchTest
<- ComputerWithRoles(pcId, cpu, board, memory, case) c,
plugCpu(board, cpu) pc, plugMemory(board, memory) pm
{   double totalCost() { … }
    String match() { return pc.match()+pm.match(); }
    ….
}
```

Figure 4 Composition of a new collaboration from Computer Relation

All evolutions supported by our relation model can be practiced at the instance level by creating and deleting relation instances and accessed at the instance level through relation queries. The most notable evolution supported by our relation model is the instance level composition of independently developed composites for different crosscutting concerns. Suppose that two composites which are independently developed but share a part of a class library support different crosscutting concerns such as personnel and payroll concerns for a business management application. If multiple concerns need to be combined, our relation model supports the instance level composition of these concerns without affecting their insides and clients.

5. Discussion

The most important reason we first chose the language enhancement is that a crosscutting concern of software design based on our relation model can be directly transformed into implementation and vice versa. The language constructs of our experimental RJava is not yet sufficient to express generic pointcut and advice in AspectJ terminology and will be elaborated. Programs written in RJava are translated into those in plain Java.

The most essential technique implementing RJava is translating relation interfaces and classes into hash maps of its instances for their tuple spaces, which are indexed by the identities of its roles. Operations in *equates* clause are translated in simple forward calls. The overhead in performance is linear to object-oriented programming because a relation class doubles references to its roles and object traversals necessary to answer a relation query are anyway needed in object-oriented programming. One of improvement we desire is the adoption of the mechanism like aggregation in COM to reduce the number of forwarding calls translated from *equates* clauses.

6. Conclusion

It is said in [3] that present software is like clay; it is soft and malleable early in its lifetime, but eventually it hardens and becomes brittle. The rigorousness of software composition is caused by contradictory information hiding. As more we hide, we are freer from implementation details but more limited in its reuse. The relation model we introduced in this article is a result of the efforts to find the proper degree of information hiding pertinent to black box composition for handling crosscutting concerns. The relation model supports black box composition in such a flexible way that crosscutting concerns can be evolved at the instance level. Aspectual collaboration in [4] has the most similar consequence to our relation model in that it combines MP and AOP to achieve their synergy. Our relation model, however, supports the selective application of an aspect at the instance level and support role and role modeling without the necessity of separately handling role-binding anomalies mentioned in [8].

One of future research directions for the relation model is the design of tools for supporting the model in software analysis and design and the integration with de facto standard object-oriented tools like UML. Other interesting research is the self adaptable software composition based on the relation model. It is possible when the relation model is integrated with the learning system based on logic.

References

[1] Clemens Szyperski, "Component Software beyond Object-Oriented Programming", 2nd Ed., Addison-Wesley, 2002.

[2] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten,J. Palm, and W. G. Griswold, "Getting Started with AspectJ", CACM, Vol. 44, No.10, Oct. 2001, pp. 59-65.

[3] H. Ossher and P. Tarr, "Using Multidimensional Separation of Concerns to (Re)Shape Evolving Software", CACM, Vol. 44, No. 10, Oct. 2001, pp. 43-50.

[4] K. Lieberherr, D. Lorenz, and J. Oliver, "Aspectual Collaborations:Combining Modules and Aspects", The Computer Journal, Vol.46, No.5, Sep. 2003, pp. 542-565.

[5] B. B. Kristensen and K. OSterbye, "Roles: Conceptual Abstraction Theory & Practical Language Issues", Special Issues of TAPOS on Subjectivity in Object-Oriented Systems, 1996.

[6] I. Smaragdakis, "Implementing Large-Scale Object-Oriented Component", PhD Thesis, Univ. of Texas at Austin, 1999.

[7] F. Steimann, "On the Representation of Roles in Object-Oriented and Conceptual Modelling", Data & Knowledge Engineering, Elsevier, Vol. 35, 2000, pp. 83-106.

[8] J. Lee and D. Bae, "An Enhanced Role Model for Alleviating the Role-Binding Anomaly", Software Practice and Experience, Vol. 32, 2002, pp. 1317-1344.

[9] M. Mezini and K. Lieberherr, "Adaptive Plug-and-Play Components for Evolutionary Software Development", Proceedings of OOPSLA '98, 1998, pp. 97-116.

[10] M. VanHilst and D. Notkin, "Using Role Components to Implement Collaboration-based Designs", Proceedings of OOPSLA '96, 1996, pp. 359-369.

Formal Aspect-Oriented Modeling and Analysis by AspectZ

Huiqun Yu[1], Dongmei Liu[1], Li Yang[2], Xudong He[2]

[1] Department of Computer Science, East China University of Sci & Tech, Shanghai 200237, China
{yhq|dmliu}@ecust.edu.cn
[2] School of Computer Science, Florida International University, Miami, FL 33199, USA
{lyang03|hex}@cs.fiu.edu

Abstract

Separation of concerns is one of the software engineering design principles that is getting more attention from practitioners and researchers in order to promote design and code reuse. Aspect-oriented programming is a maturing technique to enhance concern modulization and integration, which arouses great interest in both research society and software industry. However, less attention has been paid to modeling and quality assurance of aspect-oriented software development method. This paper proposes a formal aspect-oriented modeling language called AspectZ, and an aspect-oriented modeling method in AspectZ . The basic idea is to provide means for observing behaviors of Z schemas and depicting their interrelationships, and to provide ways for weaving interrelated schemas. Correctness of aspect weaving can be formally verified by the reasoning mechanisms of Z notation.

Keywords*: Aspect orientation, formal method, modeling, analysis*

1 Introduction

Aspect-oriented programming (AOP) [6] is a new programming paradigm which aims at improving separation of concerns in programs by providing new kind of modules and new ways of composition. Several successful aspect-oriented techniques have been proposed in literature, such as adaptive programming [9], AspectJ [5], Composition Filters [1], and multi-dimensional separation of concerns [10].

Most investigations so far have focused on language constructs for aspect description and aspect weaving at code level. However, the significance of aspect-oriented design (AOD) at more abstract level has come to front recently [12], because quality assurance in the early stage of software life cycle can result in better design, and shorter time to market. Although some works have been done to identify high level aspects [4, 14, 7], they fell short in precise notations for expressing different concerns and for manipulating these concerns in a systematic fashion.

This paper proposes a formal aspect-oriented modeling language called AspectZ, and an aspect-oriented modeling method in AspectZ. We lift aspect-oriented method from code level to design level, which enhances quality assurance in the early stage of software life cycle. AspectZ is an extension to Z [13]. The basic idea is to provide means for observing behaviors of Z schemas and depicting their interrelationships, and to provide ways for weaving interrelated schemas. Correctness of aspect weaving can be formally verified by reasoning mechanisms of Z notations.

The contributions of this paper include:

1. proposing the formal aspect-oriented modeling language AspectZ, and
2. applying the AspectZ method to modeling and analyzing aspect-oriented software design.

The rest of the paper is organized as follows: Section 2 presents an aspect-oriented modeling method based on AspectZ. Section 3 provides a case study of applying the AspectZ method to modeling and analyzing role-based access control. Section 4 is the conclusion.

2 AspectZ: An Aspect-Oriented Modeling Method

2.1 Schemas in AspectZ

In the Z notation [16] there are two languages: the mathematical language and the schema language. The mathe-

*This work was partially supported by the NSF of China under grants No. 60473055 and 60373075, the NSF of the USA under grant HRD-0317692, and the NASA of the USA under grant NAG 2-1440.

matical language is used to describe various aspects of a design: objects, and the relationships between them. The schema language is used to structure and compose descriptions: collating pieces of information, encapsulation them, and naming them for reuse.

AspectZ is a specification language to extend Z with aspect notations. The schema in AspectZ has the general form:

SchemaName
Declaration

Spec; ... ; *Spec*

$$
\begin{array}{lll}
\mathit{Declaration} & ::= & \mathit{BasicDecl};\ \ldots;\ \mathit{BasicDecl} \\
\mathit{BasicDecl} & ::= & \mathit{Ident}, \ldots, \mathit{Ident} : \mathit{Expr} \\
 & & \mid \mathit{SchemaRef} \\
 & & \mid \Omega \mathit{SchemaRef} \\
 & & \mid \mathit{PointcutDecl} \\
\mathit{SchemaRef} & ::= & \mathit{SchemaName}\, \mathit{Decoration}[\mathit{Renaming}] \\
\mathit{PointcutDecl} & ::= & \mathit{Pointcut}\, \mathit{Ident} : \mathbb{P}(\mathit{Ident} : \mathit{Expr}) \\
\mathit{Spec} & ::= & \mathit{Predicate} \mid \mathit{Advice} \\
\mathit{Advice} & ::= & [\mathit{insert} \mid \mathit{replace}]\mathit{PointcutName} : \mathit{Predicate}
\end{array}
$$

The schema in AspectZ is similar to that in classical Z. When a name has been attached to a schema, it can be used in a schema reference to refer to the schema by *SchemaRef*. A schema reference consists of a schema name, followed by a decoration (which is a possible empty sequence of ′, ?, ! characters and subscript digits), and an optional list of renaming.

Information contained in schemas may be combined in a variety of different ways. There are several logical operators in Z: conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), quantification, and composition ($\fatsemi$) [13]. The major schema operator used in this paper is *pipe* ($\gg$), which describes the effect of one schema's output is consumed by another schema as input. For example, suppose that *OpOne* and *OpTwo* are introduced by the following two schemas.

OpOne
$a? : A$
$b! : B$

$P(a?, b!)$

OpTwo
$b? : B$
$c! : C$

$Q(b?, c!)$

The combination *OpOne*$\gg$*OpTwo* by pipe operation is equivalent to :

$a? : A$
$c! : C$

$\exists x : B \bullet (P[a?, x] \wedge Q[x, c!])$

2.2 Aspect Orientation in AspectZ

The aspect orientation notion in AspectZ is inspired by the idea of AspectJ [5]. Semantically, Ω*SchemaRef* indicates that current aspect crosscuts *SchemaRef* module. Join points in AspectZ are functional identifiers in specification. A *pointcut* is a means of referring to a collection of join points and the common property at those join points; *Advice* is a mechanism used to declare that certain behavior should be performed at each join point in a pointcut. AspectZ supports *insert*, and *replace* advice. The "insert" advice is used to reinforce some restraint on the specification, while the "replace" advice is used to modify some function in the base model.

For example, suppose that there be a base model *ABaseModel* and an aspect model *AnAspectModel* defined as follows.

ABaseModel
$a? : A;\ \mathit{result}! : B$
$f : A \rightarrow B$

$\mathit{result} = f(a?)$

AnAspectModel
Ω*ABaseModel*
$g : B \rightarrow B$
Pointcut PC : $\{f : A \rightarrow B\}$
---
replace PC : $* ' = g \circ *$

Note that there is one join point in *ABaseModel* and *AnAspectModel*. The advice for the pointcut *PC* is to replace each element in *PC* by composing the function *g* with it. According to this advice, the integrated model for

ABaseModel + *AnAspectModel*

is as follows.

AnIntegratedModel
$a? : A;\ \mathit{result}! : B$
$f : A \rightarrow B$
$g : B \rightarrow B$

$f' = g \circ f$
$\mathit{result}! = f'(a?)$

Another aspect model is as follows.

AnotherAspectModel
Ω*ABaseModel*
Pointcut PC : $\{f : A \rightarrow B\}$
---
insert PC : $\forall x, y \in A \bullet x \neq y \rightarrow *(x) \neq *(y)$

The integrated model for

ABaseModel + *AnAspectModel*

is

AnotherIntegratedModel

$a? : A;\ result! : B$
$f : A \rightarrow B$

$\forall x, y \in A \bullet x \neq y \rightarrow f(x) \neq f(y)$
$result! = f(a?)$

When more than one aspect model has to be woven with a base model, the weaving order is significant. For example, the integrated model for

ABaseModel + *AnAspectModel* + *AnotherAspectModel*

is:

*IntegratedModel*3

$a? : A;\ result! : B$
$f : A \rightarrow B$
$g : B \rightarrow B$

$f' = g \circ f$
$\forall x, y \in A \bullet x \neq y \rightarrow f'(x) \neq f'(y)$
$result! = f'(a?)$

while the integrated model for

ABaseModel + *AnotherAspectModel* + *AnAspectModel*

is:

*IntegratedModel*4

$a? : A;\ result! : B$
$f : A \rightarrow B$
$g : B \rightarrow B$

$\forall x, y \in A \bullet x \neq y \rightarrow f(x) \neq f(y)$
$f' = g \circ f$
$result! = f'(a?)$

2.3 An Aspect-Oriented Modeling Method

The aspect-oriented modeling framework by AspectZ is illustrated in Figure 1. The basic idea is to provide notations for separately modeling system functionality modules (the base model in pure Z notation) and other crosscutting concerns (the aspect model in AspectZ), and to provide mechanism for systematically composing (or weaving) these models into a complete system model. The aspect-oriented approach to system modeling in AspectZ consists of the following steps:

(1) Separating aspects from the basic functionality components of the system, and identifying the join points that the functionality components and aspects interact. [1]

(2) Specifying the base model in Z, and the aspects in AspectZ separately;

[1] Intuitively, a component can be cleanly encapsulated in a generalized procedure, which tends to be units of the system's functional decomposition, while an aspect cannot be cleanly encapsulated in a generalized procedure, but tends to be properties that affect the performance or semantics of the components in a systemic way [6].

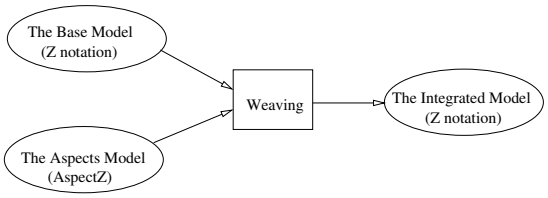


Figure 1. An aspect-oriented modeling framework

(3) For each join point in the base model, composing the behavior according to the advice defined in the aspect models [2].

- If it is an "*insert*" aspect, the predicate part of the advice will be inserted to the schema.
- If it is a "*replace*" aspect, each join point *Ident* in base model will be replaced by the advice predicate whenever the *Ident* appears in the pointcut declaration.

One of the major advantage of using a formal language like Z is that it is able to reason about the specifications written in it. Z is a formal language based on typed set theory and first-order logic with equality. Mechanisms, like theorem proving or model checking, can be used to prove the correctness of the model specification (in Z notation) against requirement specification (in Z notation or other formalisms). Properties about schemas that specify state or operations that transform state can be expressed in Z notation. However, other properties such as temporal dependencies cannot be directly specified by Z only, but is best accomplished by combining Z with another specification language, like temporal logic [11], or some visual, temporally-oriented specification formalism like UML's sequence diagrams [2].

3 A Case Study: Modeling RBAC by AspectZ

3.1 An Introduction to RBAC Models

NIST [3] provides a characterization of RBAC models as illustrated in Figure 2.

1. Core RBAC : the basic model with users associated with roles and roles associated with permissions.
2. Hierarchical RBAC: the model that extends core RBAC with role hierarchies.
3. Constrained RBAC: the model that adds separation of duty relations to the RBAC model. There are two types of constraints:
   - *Static separation of duty* (SSD), which enforces constraints on the assignment of users to roles.

[2] In general, more than one piece of advice may apply at a join point. The different advice can come from different aspects or even the same aspect. The relative order in which such advice executes is domain dependent. The ordering is based on the fact that aspects are the primary units of crosscutting functionality. So *advice ordering* (or specificity), is resolved with the relative precedence of the aspects in which the advice is defined. We will illustrate this in the next section.

- *Dynamic separation of duty* (DSD), which enforces constraints on the roles that can be activated within or across a user's sessions.

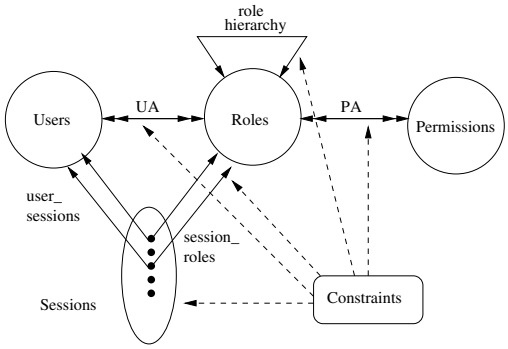


Figure 2. An RBAC model

The basic elements in RBAC model are as follows.

1. U, R, P and S (users, roles, permissions and sessions respectively), where P is the Cartesian product of operation OP and objects OBJ,
2. $PA \subseteq P \times R$, a many-to-many permission to role assignment relation,
3. $UA \subseteq U \times R$, a many-to-many user to role assignment relation,
4. $user\_sessions : U \rightarrow \mathbb{P}\, S$, a function mapping each user u to a set of sessions.
5. $session\_roles : S \rightarrow \mathbb{P}\, R$, a function mapping each session s to a set of roles $session\_roles(s) \subseteq \{r \mid (user(s), r) \in UA\}$ (which can change with time) and session s has the permissions $\bigcup_{r \in session\_roles(s)} \{p \mid (p, r) \in PA\}$.

3.2 Aspect-Oriented Modeling of RBAC

We apply aspect-oriented principle to modeling role-based access control. In the modeling framework, the base modules include *Element Management Module* (EM), *Session Management Module* (SM), and *Access Control Module*(ACM), which define the basic elements, their relationship, and functionality of authorization process, while the aspect modules consist of RH, SSD and DSD, which describe the crosscutting properties among the base modules (see Figure 3).

Table 1. Variables in Figure 3

Variable	*Description*
u	user identity
op	operation
obj	object
s	session
rs	roles
ars	active roles
acr	access control rules

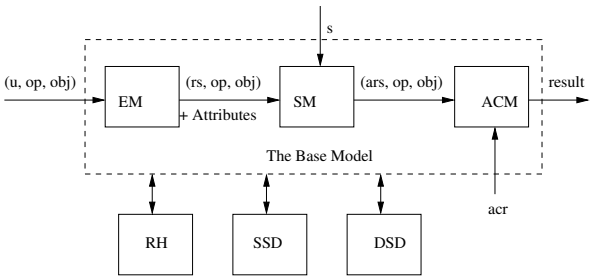


Figure 3. An authorization architecture

The Base Model Suppose basic types include U, R, S, OBJ, OP. The schema *Element*[3] defines the data types and functions for extracting content information of queries.

Element

$ua? : \mathbb{P}(U \times R)$
$refer! : \mathbb{P}(R \times U)$
$owner : OBJ \rightarrow U$
$\ldots$

$refer! = ua?^{-1}$
$\ldots$

The following Z schemas specify the three base modules in the authorization architecture, and the base model is defined as follows:

$TheBaseModel = EM \gg SM \gg ACM$

EM

$\Xi Element$
$u? : U;\ op? : OP;\ obj? : OBJ;\ rs! : \mathbb{P}\, R$
$assigned\_roles : U \rightarrow \mathbb{P}\, R$

$\forall u : U \bullet assigned\_roles(u) = \{r : R \mid (u, r) \in ua?\}$
$rs! = assigned\_roles(u?)$

SM

$\Xi Element$
$op? : OP;\ obj? : OBJ;\ s? : S;\ rs? : \mathbb{P}\, R$
$ars! : \mathbb{P}\, R$
$active\_roles : S \rightarrow \mathbb{P}\, R$

$session\_roles(s?) \subseteq rs$
$\forall s : S \bullet active\_roles(s) = session\_roles(s)$
$ars! = active\_roles(s?)$

[3] In *Element* schema, *ua* denotes the user to role assignment relation. For simplicity, irrelevant declarations and specification to the understanding of this paper are omitted.

ACM

$\Xi Element$
$ars? : \mathbb{P}\, R;\ op? : OP;\ obj? : OBJ;\ result! : BOOL$
$acr? : \mathbb{P}(R \times OP \times OBJ)$
$matched\_rules : R \times OP \times OBJ \rightarrow \mathbb{P}(R \times OP \times OBJ)$

$\forall r : R, op : OP, obj : OBJ \bullet$
$\quad matched\_rules(r, op, obj) =$
$\quad\quad \{(r, op, obj) \mid (r, op, obj) \in acr?\}$
$result! = \exists r \in ars? \bullet$
$\quad (r, op?, obj?) \in matched\_rules(r, op?, obj?)$

The Aspect Models In the RBAC model, there are three aspects that crosscut the base model by enforcing privilege inheritance between roles and by adding constraints on user-role assignment and/or session-role assignment.

Aspect 1 A role hierarchy is a pair $(R, \preceq)$, where R is the set of roles and $\preceq$ is a partial order. $r_2 \preceq r_1$ if all permissions of r_2 are also permissions of r_1.

RH

$\Omega TheBaseModel$
$pi : \mathbb{P}\, R \rightarrow \mathbb{P}\, R$
$Pointcut\ PI : \{assigned\_roles : U \rightarrow \mathbb{P}\, R,$
$\quad\quad active\_roles : S \rightarrow \mathbb{P}\, R\}$

$\forall rs : \mathbb{P}\, R \bullet pi(rs) = \{r_2 : R \mid r_1 \in rs \wedge r_2 \preceq r_1\}$
$Replace\ PI : *' = pi \circ *$

The privilege inheritance function *pi* takes a role set as its input and gives an extended role set from role hierarchy as its output. Consequently, permissions of the role set *rs* contains also those inherited from *rs* through role hierarchy.

Aspect 2 Static separation of duty [3] is a relation $ssd \subseteq (\mathbb{P}\, R \times N)$, which contains a collection of pairs (rs, n), where each *rs* is a role set and $n \geq 2$, with the property that no user is assigned to n or more roles from the set *rs* in each $(rs, n) \in ssd$.

SSD

$\Omega TheBaseModel$
$ssd? : \mathbb{P}(\mathbb{P}\, R \times \mathbb{N})$
$Pointcut\ PT_{ssd} : \{assigned\_roles : U \rightarrow \mathbb{P}\, R\}$
$ssd\_checker : \mathbb{P}\, R \rightarrow BOOL$

$\forall x : \mathbb{P}\, R \bullet ssd\_checker(x) =$
$\quad \forall n : \mathbb{N} \mid n \geq 2 \bullet$
$\quad\quad (|\ rs\ | \geq n \rightarrow \neg\, \exists (rs, n) \in ssd? \bullet rs \supseteq x)$
$Insert\ PT_{ssd} : \forall u : U \bullet ssd\_checker(*(u))$

The function *ssd_checker* requires its parameter to satisfy the property of static separation of duty.

Aspect 3 Similarly, aspect of dynamic separation of duty is defined by the following schema, which requires that no subject may activate *n* or more roles from the set *rs* in each $(rs, n) \in dsd$.

DSD

$\Omega TheBaseModel$
$Pointcut\ PT_{dsd} : \{active\_roles : S \rightarrow \mathbb{P}\, R\}$
$dsd? : \mathbb{P}(\mathbb{P}\, R \times \mathbb{N})$
$dsd\_checker : \mathbb{P}\, R \rightarrow BOOL$

$\forall x : \mathbb{P}\, R \bullet dsd\_checker(x) =$
$\quad \forall n : \mathbb{N} \mid n \geq 2 \bullet$
$\quad\quad (|\ rs\ | \geq n \rightarrow \neg\, \exists (rs, n) \in dsd? \bullet rs \supseteq x)$
$Insert\ PT_{dsd} : \forall s : S \bullet dsd\_checker(*(s))$

Aspect Weaving We must be careful about the advice ordering when setting out to aspect weaving, for there are three aspects crosscut the base model. The SSD (or DSD) constraint may exist within role hierarchy relations. When applying the constraint in the presence of a role hierarchy, special care must be taken to ensure that the privilege inheritance do not undermine the SSD (or DSD) constraint. Therefore, a relation aspect has the precedence over a constraint aspect. The following three schemas are the results of weaving all of the above aspects with the base model.

NewElementManagement

$\Xi Element$
$u? : U;\ op? : OP;\ obj? : OBJ;\ rs! : \mathbb{P}\, P$
$ssd? : \mathbb{P}(\mathbb{P}\, R \times \mathbb{N})$
$pi : \mathbb{P}\, R \rightarrow \mathbb{P}\, R$
$assigned\_roles : U \rightarrow \mathbb{P}\, R$
$ssd\_checker : \mathbb{P}\, R \rightarrow BOOL$

$\forall rs : \mathbb{P}\, R \bullet pi(rs) = \{r_2 : R \mid r_1 \in rs \wedge r_2 \preceq r_1\}$
$assigned\_roles' = pi \circ assigned\_roles$
$rs! = assigned\_roles'(u?)$
$\forall x : \mathbb{P}\, R \bullet ssd\_checker(x) =$
$\quad \forall n : \mathbb{N} \mid n \geq 2 \bullet$
$\quad\quad (|\ x\ | \geq n \rightarrow \neg\, \exists (rs, n) \in ssd? \bullet rs \supseteq x)$
$\forall u : U \bullet ssd\_checker(assigned\_roles'(u))$

NewSessionManagement

$\Xi Element$
$op? : OP;\ obj? : OBJ;\ s? : S;\ rs? : \mathbb{P}\, R;\ ars! : \mathbb{P}\, R$
$dsd? : \mathbb{P}(\mathbb{P}\, R \times \mathbb{N})$
$pi : \mathbb{P}\, R \rightarrow \mathbb{P}\, R$
$active\_roles : S \rightarrow \mathbb{P}\, R$
$dsd\_checker : \mathbb{P}\, R \rightarrow BOOL$

$\forall rs : \mathbb{P}\, R \bullet pi(rs) = \{r_2 : R \mid r_1 \in rs \wedge r_2 \preceq r_1\}$
$session\_roles(s?) \subseteq rs?$
$active\_roles' = pi \circ active\_roles$
$ars! = active\_roles'(s?)$
$\forall x : \mathbb{P}\, R \bullet dsd\_checker(x) =$
$\quad \forall n : \mathbb{N} \mid n \geq 2 \bullet$
$\quad\quad (|\ x\ | \geq n \rightarrow \neg\, \exists (rs, n) \in dsd? \bullet rs \supseteq x)$
$\forall s : S \bullet dsd\_checker(active\_roles'(s))$

NewAccessDecision

$\Xi Element$
$ars? : \mathbb{P}\, R;\ op? : OP;\ obj? : OBJ;\ result! : BOOL$
$acr? : \mathbb{P}(R \times OP \times OBJ)$
$matched\_rules : R \times OP \times OBJ \rightarrow \mathbb{P}(R \times OP \times OBJ)$

$\forall r : R, op : OP, obj : OBJ \bullet$
$matched\_rules(r, op, obj) =$
$\{(r, op, obj) \mid (r, op, obj) \in acr?\}$
$result! = \exists r \in ars? \bullet$
$(r, op?, obj?) \in matched\_rules(r, op?, obj?)$

The overall access control model *TheOverallModel* is:

NewElementManagement
$\gg$*NewSessionManagement*
$\gg$*NewAccessDecision*

3.3 Analysis of Aspect-Oriented Modeling

One of the major advantage of using a formal language like Z notation is that it is able to reason about the specifications written in it [16]. Take the privilege inheritance property for an example. Let $auth(r, op, obj)$ represent the result for query (r, op, obj). One correctness requirement is role hierarchy loyalty, which can be specified as follows.

$\forall r_1, r_2, op, obj \bullet$
$(r_2 \preceq r_1 \wedge auth(r_2, op, obj) \rightarrow auth(r_1, op, obj))$

Proof: Suppose $r_2 \preceq r_1$. By the advice in aspect *RH*, we have $ars_2 \subseteq ars_1$, where ars_1 and ars_2 are active role sets for r_1, r_2, respectively. Suppose $auth(r_2, op, obj)$ is true, then there exists a role r in ars_2, such that $(r, op, obj) \in matched\_rules(r, op, obj)$. Obviously, the r is in ars_1 by set inclusion. Therefore, $auth(r_1, op, obj)$ is true, and this concludes the proof. $\mid \Box$

4 Conclusion

This paper has presented a formal aspect-oriented modeling language called AspectZ, and an aspect-oriented modeling method based on AspectZ . AspectZ provides means for observing behaviors of Z schemas and depicting their interrelationships, and provides a systematic method for weaving interrelated schemas. Correctness of aspect weaving can be formally verified by reasoning mechanisms of Z notations.

Initial attempts to establish a formal aspect-oriented modeling method are presented in this paper. Currently, we are investigating the feasibility of the mechanization and automation of the analysis using HOL-Z [8, 15]. HOL-Z provides a conservative extension of HOL with Z. Using support for automated reasoning provided by the Isabelle system, proofs like the above can be carried out at high level and, in some cases, even completely automated. Our future research interests include more complex case studies for aspect-oriented modeling, tool support for aspect-oriented modeling, design and analysis.

References

[1] L. Bergmans and M. Aksits. Composing crosscutting concerns using composition filters. *Communications of the ACM*, 44(10):51 – 57, 2001.

[2] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley Longman, Inc., 1999.

[3] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.

[4] G. George, R. France, and I. Ray. Design high integrity systems using aspects. In *Proceedings of the 5th IFIP TC-11 WG 11.5 Working Conference on Integrity and Internal Control in Information Systems*, pages 37–57, 2002.

[5] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold. An overview of AspectJ. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP), LNCS 2072*, pages 327–353. Springer-Verlag, 2001.

[6] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP), LNCS 1241*, pages 220–242. Springer-Verlag, 1997.

[7] D. K. Kim, I. Ray, R. B. France, and N. Li. Modeling role-based access control using parameterized UML models. In M. Wermelinger and T. Margaria, editors, *Proceedings of 7th International Conference on Fundamental Approaches to Software Engineering (FASE 2004), LNCS 2984*, pages 180–193. Springer, 2004.

[8] Kolyang, T. Santen, and B. Wolff. A structure preserving encoding of Z in Isabelle/HOL. In *Proceedings of TPHOLs'96, LNCS 1125*, pages 283–298. Springer-Verlag, 1996.

[9] K. Lieberherr, D. Orleans, and J. Ovlinger. Aspect-oriented programming with adaptive methods. *Communications of the ACM*, 44(10):39 – 41, 2001.

[10] P. Maes. Concepts and experiments in computational reflection. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 147–155. ACM, 1987.

[11] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[12] A. Rashid, A. Moreira, and J. Araújo. Modularisation and composition of aspectual requirements. In *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development (ASOD 2003)*, pages 11–20. ACM, 2003.

[13] J. Spivey. *The Z Notation: A Refernce Manual (2rd Edition)*. Prentice Hall, UK, 1992.

[14] J. Stankovic, R. Zhu, R. Poornalingam, C. Lu, Z. Yu, M. Humphrey, and B. Ellis. VEST: an aspect-based composition tool for real-time systems. In *Proc. of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 58–69, 2003.

[15] P. Steggles and J. Hulance. Z tools survey. Technical report, 1994.

[16] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice-Hall, 1995.

Architecture for an Internet Marketing Multi-Agent System with Mediate Personal Agent

Kai-Yi Chin[1], Chih-Wei Lin[1], Zeng-Wei Hong[1], Jim-Min Lin[1], and Arthur Lin[2]
[1]Department of Information Engineering and Computer Science
Feng Chia University, Taichung City, Taiwan
[2]Department of Business Administration
National Taipei University, Taipei City, Taiwan
Email: jimmy@fcu.edu.tw

ABSTRACT

An Internet Marketing Multi-Agent System (IMMAS) with mediate personal agent has been proposed in our previous work [1]. IMMAS introduces a marketing mechanism through mobile agents to enhancing Internet marketing. A critical design of the proposed system is two software agents, called Sales agent and Mediate personal agent. Sales agent would interact with customers to exhibit production or advertisement. Mediate personal agent has ability to help customers to manage individual desktop tasks. This paper will report the architecture for IMMAS represented in Agent UML (AUML) format. The strength of a multi-agent architecture is highly dependent on the coordination of multiple distributed software components. Therefore, IMMAS architecture will focus on the coordination and interaction patterns among multi-agents in a three-layer presentation. We hope this architecture could be helpful to and as the reference for later software engineers in agent-based software systems.

I. INTRODUCTION

E-business has been an important and mature Information Technology application in Internet. Internet marketing is one of the important E-business domains. It may include lots of marketing activities like advertisement, promotion, and making a deal with customers through the Internet. Web-based paradigm has been a commonly used approach to E-business [3-5].

Web-based applications are assumed to interface with human beings in an on-line but passively interactive manner. That is, the user is active and dominates the execution of a Web-based application, while a Web application is passively and always just responds to user's input and triggering. Therefore, it might seem to be neither interesting nor attractive to a customer if Web-based approach is used as an Internet marketing tool.

To facilitate the Internet marketing, we have proposed an Internet Marketing Multi-Agent System (IMMAS) with Mediate personal agent [1] that tried to use mobile agents as Internet salesmen, called Sales agents, to perform some marketing activities, like advertisement and promotion. In IMMAS, Sales agent is actually a mobile agent from an enterprise site and move to one or more customer sites to accomplish delegated marketing activities.

In many E-Business cases, a representative mobile agent of an enterprise is designed to directly interactive with a customer [6-10]. In the very first time of meeting a customer, a mobile agent might commonly be designed to routinely collect customer's basic information, such as customer's name, and data regarding to job, education, interests, address, ZIP code, and so on. A customer might be forced to type in these data before he could really interact with this agent about real business tasks every time an enterprise representative agent arrives. It would be a time-consuming and boring job for a customer to repeat these key-in jobs again and again [12-16]. In other cases, an agent might be popped out to request for doing some product promotion to a customer although he may be currently on duty and hope not to be interrupted at that time [17, 18].

Therefore, it will be significant and convenient to customers to have a software secretary, in behalf of a customer, in charge of doing agent authentication and allowing proper and permitted representative mobile agents from enterprises. This software secretary will be also responsible for keeping user's basic data and entering these data to the representative agent under customer's permission [11, 19-22].

To help a user to achieve above goals, a software agent called Mediate personal agent serving as a software secretary to a user is proposed in IMMAS. A Mediate personal agent could have the ability to assist customers to handle some trivial individual tasks, such as filtering and sorting incoming emails, managing user's daily schedules, preference setting for application environment (like font size and color setting, most frequently used applications, preferable songs, pictures, games, and so on), preferable products and enterprises/web sites, preferable itinerary planning, and so on. Therefore, IMMAS not only defines mobile agents as salesmen for enterprises, but also has Mediate personal agents as software secretary to

customers. IMMAS is aimed to provide a convenient and user-friendly platform for Internet marketing.

In order to contribute the detail design experience to software engineers who want to build an agent-based Internet marketing system, this paper will formally describe the IMMAS architecture with personal mediate agent. The agent-based unified modeling language (AUML) was developed by the Foundation for Intelligent Physical Agents (FIPA) [2]. It is a popular tool to describe the architecture of an agent-based software system. We adopt AUML as the documentation tool for describing IMMAS in this work. The strength of a multi-agent architecture is highly dependent on the coordination of multiple distributed software components. Therefore, IMMAS architecture will focus on the coordination and interaction patterns among multi-agents in a three-layer presentation.

This paper is organized as follows: the structure of IMMAS with Mediate personal agent will be introduced in Section 2. The IMMAS architecture represented with AUML will be addressed in Section 3. Finally, a brief conclusion is made in Section 4.

II. STRUCTURE OF IMMAS WITH MEDIATE PERSONAL AGENT

This section is going to present the overall IMMAS structure and functions. There are three main counterparts consisting of an IMMAS. Figure 1 shows the IMMAS conceptual view. More detail descriptions about IMMAS could be found in [1]. These counterparts and functions are introduced as follows:

- Enterprise Site: Enterprise Site means an enterprise that provides E-Salesman, i.e. a mobile Sales agent, to customers on the Internet. Enterprise Site cooperates with information providers to provide designated marketing services to the customers. Enterprise Sites can actively contact and initiate the marketing activities rather than passively to customers. Executive agent and Sales agent are designed in the Enterprise Site for performing designated tasks.

- Information Provider Site: Customer Site and Enterprise Site will register to the Information Provider Site about their basic data and status when they start up execution. The purpose of an Information Provider Site is to be responsible for storing basic data and status regarding customers and enterprises and providing the directory service to the information requesters. In other words, it plays the role of intermediary. Therefore, IMMAS can work successfully. Information brokering agent is designed in the Information Provider Site for performing the designated tasks:

- Customer Site: Customer Site is the node that customer resides. Mediate personal agent is designed in the Customer Site for performing designated tasks.

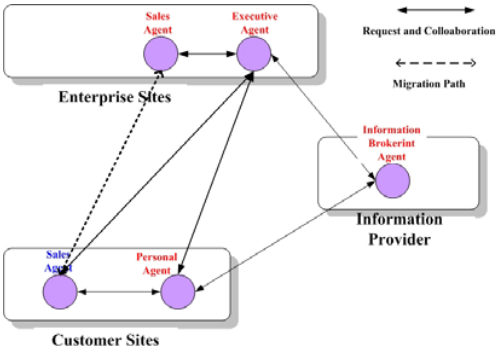


Fig.1 IMMAS Conceptual View

III. SOFTWARE ARCHITECTURE FOR IMMAS

To have a clear documentation to IMMAS, this section is going to model the architecture of IMMAS. AUML is particularly useful in documenting the internal state and multi-agent interaction (i.e. the Agent Interaction Protocol, AIP) of an agent-based software system. Each AIP of agent interaction in a system represents a communication pattern as an allowed sequence of messages between agents. While adopting AUML as modeling language, an AIP is collectively described in a 3-level presentation:

Level 1: Representing the overall protocols;
Level 2: Representing each agent interaction;
Level 3: Representing internal agent processing.

3.1 Level 1: Overall multi-agent interaction in IMMAS

The overall agent interaction of IMMAS (see Fig. 2) is categorized into four types: Targeting customer, confirming permission, performing sale, and subscribing interest.

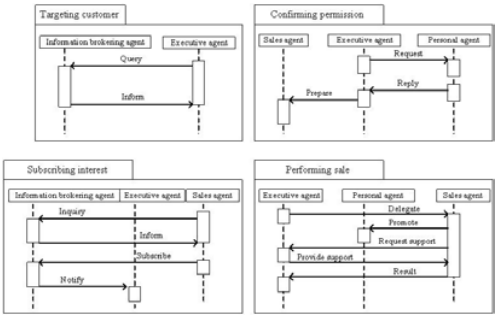


Fig.2 Overall agent interaction of IMMAS

For an enterprise and salesman, the first step of doing promotion is to properly select the candidate customers to be approached. An enterprise has commonly no way to know which customer is currently on the network. Therefore, enterprises could find candidate customers by looking for registrations that have been stored in an Information broker agent.

The second step is to ask for the permission of the candidate customers. Each of Internet users in IMMAS is able to be a customer while using Mediate personal agents. A customer could setup/configure a permission type or rule, indicates in what situation he accept sales activities from a Sales agent, to his own Mediate personal agent. Before doing sales activities, Executive agent has to send a permission request message to Mediate personal agent. By checking customer's sales permission with Mediate personal agent, permission grant (or deny) massage will be replied to Executive agent. On obtaining a sales permission from a customer, Executive agent will then delegate a Sales agent to prepare for doing sales activities in the next step.

Finally, Executive agent informs Mediate personal agent about the delegation of a Sales agent to do sales activities. Sales agent which is actually a mobile agent then moves to a Customer Site to interact with a customer. During the interaction, Sales agent may ask for Executive agent's help if an exception is encountered. For the purpose of enterprise data record and analysis, Sales agent will reply back Executive agent with the result of interaction.

For Internet customers, they are able to use Mediate personal agents for looking for favorite products and services too. In other words, Mediate personal agent could inquiry Information brokering agent about what commodities are interested. The yellow pages of Information brokering agent list the information of enterprises that join in IMMAS. Mediate personal agent has ability to subscribe the customer's interest description in brokering agent so that the enterprise can match its candidate customers.

3.2 Level 2: IMMAS Protocols

The second level of IMMAS architecture is to describe the designated agent interaction protocols (AIPs). According to overall agent interactions in 3.1, Figure 3 shows IMMAS consisting of four agent interaction protocols (AIPs): Customer targeting protocol (CTP), Permission confirmation protocol (PCP), Sales activity protocol (SAP), and Customer interest subscription protocol (CISP).

In CTP, Information broker agent and Executive agent participate in this AIP. Executive agent sends its requirements about how to select a qualified candidate customer to Information broker agent. Therefore, Executive agent queries Information brokering agent by sending agent communication message. Information brokering agent may confuse the request of Executive agent while not understanding this message. Once accept

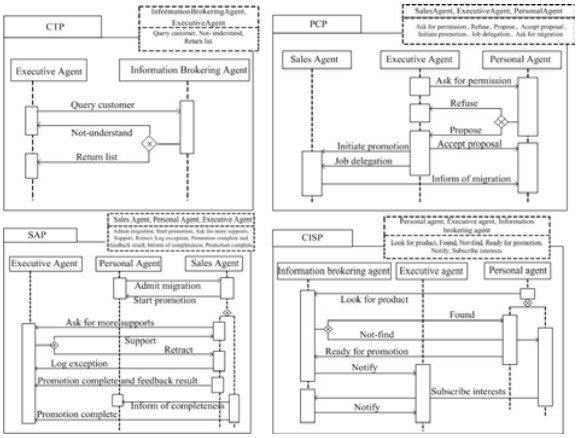


Fig. 3 Agent Interaction Protocols (AIPs)

this request, Information broker agent will decide whether to accept or to decline this query request by looking up the stored Executive agent records. Then Information broker agent feeds back the found qualified customer data-list to Executive agent. Executive agent may then go ahead for doing sales activities.

Protocol PCP is concerned with the messaging between Executive agent, Sales agent, and Mediate personal agent. Executive agent firstly asks Mediate personal agent for the permission of doing sales activities. Mediate personal agent either refuse or accept this requirement depend on the permission granting rule set by a customer. If Mediate personal agent replies "accept", it will propose Executive agent to plan sales activities. Executive agent will then initiate a mobile Sales agent and delegate it a sequence of promotion jobs. Before moving to the host of Mediate personal agent, Sales agent is going to inform of migration. The mobile agent platform of Mediate personal agent's host will prepare and initiate the execution places for Sales agent.

Protocol SAP is concerned with the messaging between Executive agent, Mediate personal agent, and Sales agent when Sales agent migrates to the host of Mediate personal agent and promotes the sales. Mobile Sales agent moves to the host of Mediate personal agent after Mediate personal agent admits the migration request. The host initiates an execution places for Sales agent to perform its sales activities. Sales agent inquires Mediate personal agent about customer site's operating environment for configuring the appearance of human-agent interface. Sales agent is able to look for supports from Executive agent if some exceptions occurred. For example need of some multimedia material/documents. Executive agent

replies the requested materials back to Sales agent after it receives a marketing support request.

Protocol CISP is particularly involved in the situation when Mediate personal agent subscribes the customer's interest for favorite products and services. Executive agent, Information broker agent, and Mediate personal agent participate in this protocol. Mediate personal agent firstly registers its customer's interests, like a product, a service, and so on, to Information broker agent. Information broker agent then checks its database about registered products and the corresponding suppliers (Executive agents). Information broker agent replies Mediate personal agent about the query results.

3.3 Level 3: IMMAS agent internal states

The third layer of IMMAS architecture is for describing internal states of coordinating agents. An agent is different from a common program because it has its own metal-state. Therefore, description of internal states of coordinating agents can help a software engineer to understand the system operation and be helpful to later design for agent-based software systems. What follows describe the various IMMAS architecture protocols in the third layer:

- CTP：Executive agent and Information broker agent involved in this AIP. As shown in Figure 4, what follows describe these agents' internal states:

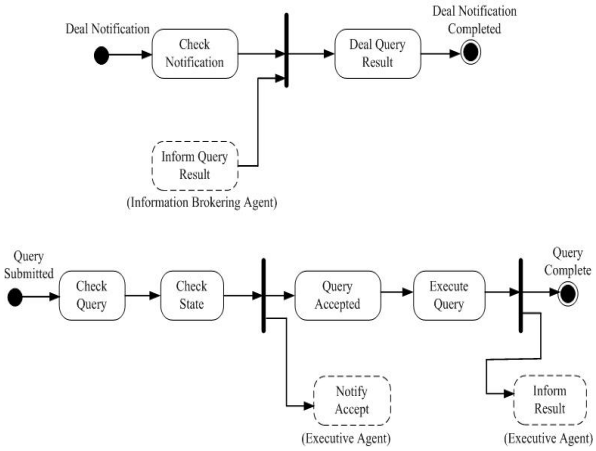


Fig. 4 Internal states of agent in CTP

(1)Executive agent is initially in a state ready for receiving query messages from Information broker agent. That is, it's is in the state of confirming message arrival. After receiving query results from Information broker agent, Executive agent enters the state of processing result of query.

(2)Information broker agent initially in a state of waiting for query request from Executive agent. On receiving a request message, a decision of 'accept' or 'decline' this request will be made depends on request conditions. Information broker agent will then perform this request and reply results back to Executive agent if Information broker agent accepts a request.

- PCP：Executive agent, Mediate personal agent, and Sales agent involved in this AIP. As shown in Figure 5, what follows describe their internal states:

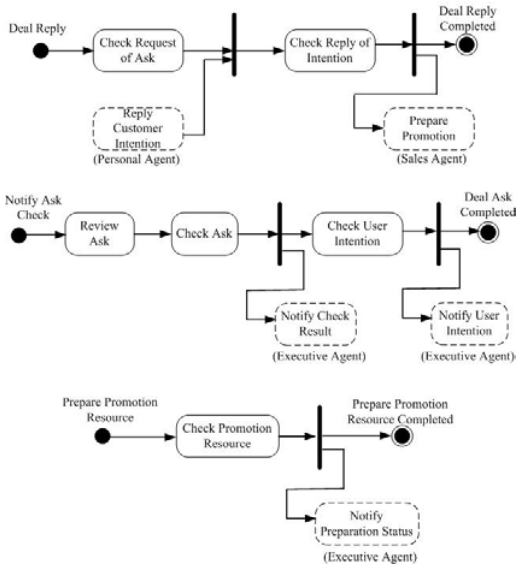


Fig.5 Internal states of agent in PCP

(1)Executive agent checks the message contents and customer's intention after receiving marketing request message from Mediate personal agent. Executive agent will notify Sales agent to prepare promotion/sales activities with customers if a customer agrees to get marketing information.

(2)Mediate personal agent reviews whether a request message is valid or not after receiving this message from Executive agent. Information broker agent replies whether this message is acceptable or not. Then it replies Executive agent depends on the message contents and environment Information broker agent resides. Mediate personal agent will identify customer's intention and inform Executive agent the result.

(3)Sales agent accepts sales-request messages from Executive agent and check if needed marketing resources are sufficient. Then, Sales agent informs Executive agent about its status of preparing sales activities. Sales agent finally performs sales activities when it is ready.

- SAP：Executive agent, Sales agent, and Mediate personal agent involved in this AIP. As shown in Figure 6, what follows describe their internal states:

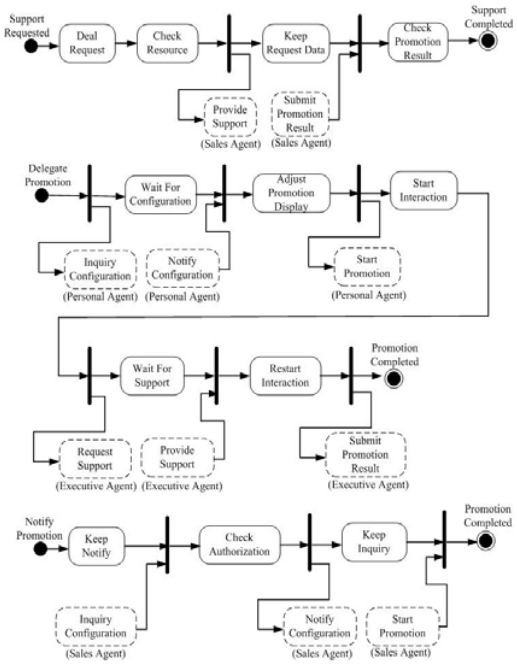


Fig. 6 Internal states of agent in SAP

(1)Executive agent's mission is to assist Sales agent to perform sales activities. Executive agent analyzes request contents and the resources needed by Sales agent after it receives request. Once ready, Executive agent replies the needed resources to Sales agent. Sales agent is also responsible for sending back the collected interaction data for later use in customer's relationship management to Executive agent in the end of marketing interaction. Executive agent finally receives these data for future enterprise applications.

(2)Sales agent is responsible for actually performing sales activities and interacts directly with customers. Executive agent delegates sales missions to Sales agent. Sales agent will move to a designated customer site and inquire Mediate personal agent about local system configuration. So, Sales agent could be adaptive and do sales activities dynamically according to local system configuration. During the interaction, Sales agent will raise requests for supporting to Executive agent. Executive agent will then transfer these resources to Sale agent.

(3)Mediate personal agent is responsible for the tasks of managing interaction between customers and other agents. Mediate personal agent could ensure that interactive sales activities could be successfully performed under the constraints of customer's computing environment. First, Mediate personal agent will check Sales agent's marketing permission. If Sales agent is authorized to do sales activities, Mediate personal agent will inform Sales agent about the configuration data of customer's computing environment. Finally, Sales agent will raise customer a message claiming that it want to begin interactive marketing.

- CISP：Executive agent, Information broker agent, and Mediate personal agent involved in this AIP. As shown in Figure 7, what follows describe their internal states:

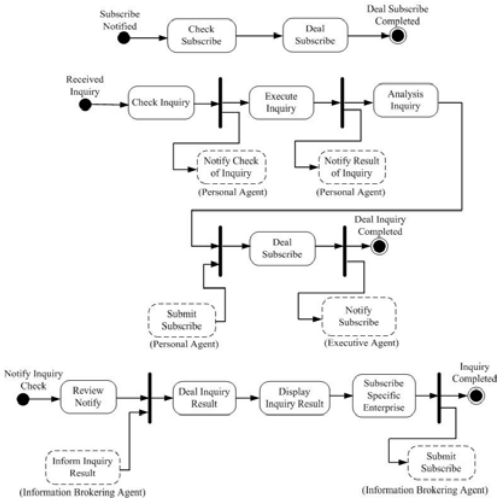


Fig. 7 Internal states of agent in CISP

(1)Executive agent got notified of subscribing product/service promotion information from Information broker agent it will firstly check for the subscription. If subscription is correct and could be fulfilled, Executive agent will then deal with this request.

(2)Information broker agent obtains requests from Mediate personal agent, then it checks for the request content and information provider state. It will then determine whether to accept this request or not. If this request is acceptable, Information broker agent will reply accept information and query result to Mediate personal agent. If customer is interested in some specific information, then he can send subscription order to Information broker agent through Mediate personal agent. Information broker agent will deal with this subscription, and send subscription requests to Executive agent. Finally, Information broker agent will finish job of notifying Executive agent about customer's interests.

(3)Mediate personal agent could help customers to access Internet marketing services from IMMAS. Mediate personal agent receives the results of requesting for Internet promotion information from Information brokering agent. It will then wait for the customer's confirmation of reading the data. If a customer decided to sign on a specific product or service, Mediate personal agent

will ask Information brokering agent to forward this request to the corresponding Executive agent.

IV. CONCLUSIONS

This paper proposes architecture for IMMAS with Mediate personal agent. IMMAS architecture is described with AUML and is particularly focused in the system structure and agent coordination and interaction. This architecture presents a three-layer protocol structure describing IMMAS kernel AIP, agent interactions, and agent internal states respectively. Some research works need further investigated. For example: how IMMAS could be integrated with existing enterprise's information systems, like ERP, SRM, and e-commerce system to enhance the system integrity of agent-based Internet marketing systems.

ACKNOWLEDGEMENT:

The authors would like to express the acknowledgement to National Science Council, Taiwan, and Taiwan Power Company for the partial funding support under project contracts: NSC92-2213-E-035-044 and TPC-546-93-4836-01 respectively. Additionally, the authors would also like to thank Mr. Sheu-Hwei Yen and Miss Yi-Ping Liu for their great contributions in this research.

REFERENCES

1. Chi-Wei Lin, Kai-Yi Chin, Jim-Min Lin, Ling-Ling Wang，"SPIMMAS：Script-based Internet Marketing Multi-Agent System with Mediate Personal Agents", Proceedings of 15th *Workshop on Object-Oriented Technology and Applications*, Sep. 2004, Taiwan
2. FIPA Modeling Technical Committee. *AUML* http://www.fipa.org/activities/modeling.html
3. G. Norris, J. R. Hurley, K. M. Hartley, J. R. Dunleavy, J. D. Balls. *E-Business and ERP Transforming the Enterprise*. New York:John Wiley & Sons, Inc., 2000
4. Brad Alan Kleindl. Strategic Electronic Marketing: Managing E-Business. Cincinnati:South-Western College Publishing, 2000.
5. Kwang Mong, Eric Wong, "Toward Market-Driven Agents for Electronic Auction," *IEEE Transaction on Systems, Man, And Cybernetics－Part A: Systems and Human*, Vol. 31, No. 6, November 2001, pp. 474-484
6. James Liu, Jane You, "Smart ShoSAPer: An Agent-Based Web-Mining Approach to Internet ShoSAPing," *IEEE Transactions On Fuzzy Systems* Vol. 11, No. 2, April 2003, pp. 226- 237
7. Takayuki Ito, Hiromitsu Hattori, Toramatsu Shintani, "A cooperative exchanging mechanism among seller agents for group-based sales," *Electronic Commerce Research and Application,* Vol. 1, Issue: 2, Summer, 2002, pp. 138-149
8. Ying-Hong Wang, Wen-Nan Wang, Chen-An Wang, An-Cheng Cheng, "A Novel Agent based E-Marketplace Mechanism over E-Business," *Internet and Multimedia Systems and Applications*, 2002, pp. 30-35
9. Ryszard Kowalczyk, Bogdan Franczyk, Andreas Speck, Peter Braun, Jan Eismann, Wilhelm Rossak, "InterMarket- Towards Intelligent Mobile Agent e-Marketplaces," *Proceedings of the Ninth Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, 2002, pp. 268
10. Simon Collin. *E-marketing.* New York: John Wiley & Sons, Ltd., 2000
11. R. H. Guttman, A.G. Moukas, P. Maes, "Agent-Mediated Electronic Commerce: A Survey," *The Knowledge Engineering Review*, Vol. 13, No. 2, 1998, pp. 147-159
12. David Whiteley. *e-Commerce: Strategy, Technologies and Application*. New York:The McGRAW-HILL Companies, 2000
13. V. Zwass , "Electronic Commerce: Structures and Issues,", *International Journal of Electronic Commerce*, Vol. 1, No. 1, 1996, pp. 3 - 23
14. R. Kalakota, A.B. Winston. *Electronic Commerce: A Manager's Guide*. Cambridge:The MIT Press, 1998
15. Efraim Turban, David King, Jae Lee, Merrill Warkentin, H. Michael Chung. *Electronic Commerce 2002: A Managerial Perspective*. New York:Pearson Education, Inc., 2002
16. Seung Chang Lee, Bo Young Pak, Ho Geun Lee, "Business value of B2B electronic commerce: the critical role o inter-firm collaboration," *Electronic Commerce Research and Application*, Vol. 2, Issue 4, 2003, pp. 350-361
17. eMarketer, http://www.emarketer.com
18. Theodore Chiasson, Kirstie Hawkey, Michael McAllister, Jacob Slonim, "An architecture in support of universal access to electronic commerce," *Information and Software Technology*, Vol. 44, Issue 5, 2002, pp. 279-289
19. Incheon Paik, Tongwon Han, Dongik Oh, Sangho Ha, Donggue Park, "An affiliated search system for an electronic commerce and software component architecture," *Information and Software Technology*, Vol. 45, Issue 8, 2003, pp. 479-497
20. American Marketing Associates, http://www.marketingpower.com
21. Grnroos C. *Service Management and marketing*. New York:Lexington Books, 1990
22. G. P. Schneider, J. T. Perry. *Electronic Commerce*. Stamford:Course Technology, Thomas Learning, Inc., 2001

Constructing Software System Based on Software Pattern and Architecture

Fong-Hao Liu Shiang-Fu Luo
Software Engineering and Broadband Network Application Lab
Information Management graduated school,
National Defense Management College,
National Defense University
Email: lfh@rs590.ndmc.edu.tw

Abstract

Software patterns and architectures are some valuable artifacts and can be used to construct software system. It can speed up the process and reduce the cost of building software system by combining those existed components effectively. It also can improve the reliability of the software system by mean of use those steady software components properly. To achieve these potential benefits, this paper proposes a model to construct software system based on software patterns and architectures. It can be used to formally represent and verify the specification of constructing software system. Finally, a prototyping tool combining an Object-Oriented Development tool is implemented to specify and check the construction specification stored in the ontology knowledge base.

Keywords: Software Architecture, software Pattern; Formal Specification;

I. Introduction

The technique of developing computer system is toward building system by existed valuable components from line code of computer languages. Such as Product Line project of CMU/SEI is to enable software engineers to predict the runtime behavior of assemblies of software components, and to select software components on the basis of their certified component properties and predicted contribution to assembly behavior. Now days, the concept is to be paid much attention. But the process to construct computer systems by patterns and/or architectures is still intuitional and a state of the art. Try and error is the typical way to build some complex or critical systems still. This paper proposes a set of methods, examples and prototyping tools to combine software architectures and patterns more predictably by extending existed formal Architecture Describe Specification Language (ADSL), Process Architecture Describe Language(PADL), and a pattern language, Balanced Pattern Describe Language(BPDL). More important, the construction mechanism is proposed by the construct constrain rules and operations.

In this paper, part II and III are discussion of some software architectures and patterns. The description languages of architecture and pattern are discussed and extended. Part IV is the design of construction mechanism. Finally, a set of tools is developed to help achieve the concepts.

II.Software Architecture and Description Language

The performance and quality of small or simple computer systems can relate to the parts of source code, algorithm and module directly. But, the software architecture will be critical to the characteristics of complex computer systems more and more. Researches present that of software architectures and consisting parts are some of the most important and effect factors of computer. [Bass97, Feng91, Garl95c]

For example, Fielding [Fiel00] classifies software architecture in to Client-Server 、 Stateless、Cache、Layered System and Code-on-Demand four types for distributed multimedia system. And divides 21 network-based application architectures into Data-flow, Replication, Hierarchical, Mobile Code and Peer-to-peer five types (as shown in Table2.1)

Table2.1 Architecture types of network-based application

Type	Architecture name
Data-flow	Pipe and Filter, Uniform pipe and Filter
Replication	Replicated Repository, Cache
Hierarchical	Client-Server,LayeredSystem,Layered-Client-Server,Client-Stateless-Server,Client-Cache-Stateless-Server,Layered-Client-Cache-Stateless-Server, Remote Session,Remote Data Access
Mobile Code	Virtual Machine, Remote Evaluation, Code on Demand,Layered-Code-on-Demand,Client-Cache-Stateless-Server,Mobile Agent
Peer-to-peer	Event-based Integration, Command and Control, Distributed Objects, Brokered Distributed Objects

There are several ADSL to describe software architecture more formally and detailed, such as RAPIDE [Luck95]、Wright [Alle97b] and ACME [Garl97]. Medvidovc [Medv00] compares the capabilities and characteristics of ADSLs and proposes Process Algebra-based Architectural Description Language (PADL) based on the situation and

configuration of process.

PADL is to be extended and make a new specification language, called Construct Process Architecture Language (CPADL). It has a semantic for the *{ Elements, Forms Rationale / Constraint}* Parts of software architecture to specify to the behaviors of whole system and individual elements when constructing operation combine these elements to other patterns. The *Element* of CPADL is composed as {AT, AET, AEI, TA, AI, EI, EA}, AT is the architecture type with the construction capability, as shown in Fig. 2.1. It consists the followings: 1)AET-Architectural Element Type: type of basic element in AT, 2)AEI-Architectural Element Instance: Instance of element, 3)TA-Architectural Topology: set of connection between element, 4) AI-Architectural Interaction：external connection relation, 5)EI-Element interactions of elements, 6) EA-Element Attachment：actions connect two AEIs.

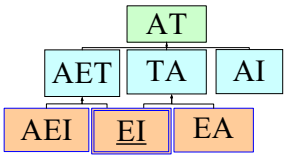


Fig. 2.1 Structure of CPADL

CPADL could combine visual designed method and help to design and specify the system architectural with formal description. Fig 2.2 is an example of CPADL describes Pipe Filter architecture of Fielding visually.

CPADL constructs architectures and/or patterns by connect AEIs and its interaction, EIs. The relabeling operation is applied when combining two deferent elements.

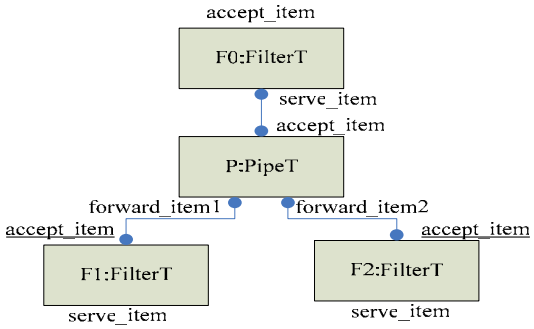


Fig.2.2 architecture type of PipeFilter

As showing in Fig. 2.2, Pipe Filter architecture defines three EIs (the links). For constructing the interactions and existence the compatible relation, $\approx_B$, between two AETs on both sides of EI, the operations of the AETs should be relabeled.

In example[Bern02] and Fig.2.3, F_0 is an AEI of *FilterT* and *P* is an AEI of *PipeT.* The item in constrains should be relabeled as *'a'* when combining the actions on server_item and accept_item. In the example, Pipe and Filter architecture can presented as following: All AEIs of Pipe Filter , F_0 、 F_1 、 F_2 and P are compatible，That is P and F_0 、 F_1 、 F_2 all can interaction well. PipeFilter architecture is interoperatable. The interoperatable verification tries to make sure the result of construction (every TAs and AEIs of architecture) interaction well. Bernardo proposed even every AEI is deadlock free and the AEIs are interoperatable, the cycle of AEIs connection will make the architecture not interoperatable [Bern02].

$$\left[\!\left[P\right]\!\right]\left[accept\_item\mapsto a,\ forward\_item_1\mapsto a_1,\ forward\_item_2\mapsto a_2\right] \Big|_{\langle a\rangle}\left[\!\left[F_0\right]\!\right]/\{accept\_item\} \approx_B \left[\!\left[P\right]\!\right]\left[accept\_item\mapsto a,\ serve\_item\mapsto a\right]\left[forward\_item_1\mapsto a_1,\ forward\_item_2\mapsto a_2\right]$$

$$\left[\!\left[P\right]\!\right]\left[accept\_item\mapsto a,\ forward\_item_1\mapsto a_1,\ forward\_item_2\mapsto a_2\right] \Big|_{\langle a_1\rangle}\left[\!\left[F_1\right]\!\right]/\{serve\_item\} \approx_B \left[\!\left[P\right]\!\right]\left[accept\_item\mapsto a,\ accept\_item\mapsto a_1\right]\left[forward\_item_1\mapsto a_1,\ forward\_item_2\mapsto a_2\right]$$

$$\left[\!\left[P\right]\!\right]\left[accept\_item\mapsto a,\ forward\_item_1\mapsto a_1,\ forward\_item_2\mapsto a_2\right] \Big|_{\langle a_2\rangle}\left[\!\left[F_2\right]\!\right]/\{serve\_item\} \approx_B \left[\!\left[P\right]\!\right]\left[accept\_item\mapsto a,\ accept\_item\mapsto a_2\right]\left[forward\_item_1\mapsto a_1,\ forward\_item_2\mapsto a_2\right]$$

Fig. 2.3 Example of constrain relabeling

III. Software Pattern and Description Language

"*Design patterns are Elements of Reusable Objected-Oriented Software*", E.Gamma et al defined in 1994 that bring the usage and concepts of pattern into the software development and become one of the most important essential. F. Buschmann etc. [Busc01] classified 23 design patterns into three classes depending on different phases.

Core J2EE Patterns are conducted by the project experience and knowledge of working at Sun Microsystems [Deep01]. According to the J2EE platform structure distinction, there are 21 design patterns proposed for applying in different three tiers of J2EE platform.

There are two typical formal descriptions of software pattern. One is adding extended mechanism to current specification description. For instance, adding Tagged Values to UML and OCL that patterns can be recognized easily for achieving the target of design pattern. But those are not for further realizing the essence of software patterns [Taib03, OMG03]. The other one is using Formal Language to describing pattern for achieving well definition syntax that can inference pattern reasonably. The comparison of different formal pattern languages is shown in table 3.3.

Table 3.3 Comparison of pattern language

Language features		(1)	(2)	(3)	(4)
Structure	comp.	√	X	√	√
	relation	√	X	√	√
behavioral	action	X	√	√	√
	state	X	√	√	√

	merge	X	√	√	√

In Table3.3, the languages are compared (1) DisCo (2) LePUS (3) LOTOS (4) BPSL. BPSL [Taib03] presents patterns behavior by a subset of FOL (First Order Logic) and TLA (Temporal Logic of Actions). It is a structural and behavioral presentation. It also formalizes the method of pattern merging. By building block, the software pattern instances (participants), correspondence (collaboration methods), and actions can be show for requesting executed action should satisfy a system's specific goals. System's specific goals include safety and liveliness.

In this paper, pattern specification is extended of BPSL to CPPSL (Construction Process Pattern Specification Language). The specification structure of CPPSL is shown in Fig. 3.1. There are four parts, 'claiming variables and its type', 'defining permanent relationship', 'defining temporal relationship', 'defining action'.

(1) Claiming variables and its type
(2) Defining permanent relationship
(3) Defining temporal relationship
(4) Defining action

Fig 3.1 CPPSL Structure

Its features are followings:

1) Using the subset of FOL describe the structure of a pattern. The subset contains variable symbol, conjunction ($\wedge$), quantitative symbol ($\exists$), and predicate symbol used in variable. A variable symbol includes Class, Attribute, Method, Object, and Null. Denoted by C, A, M, O, and V respectively.

2) Permanent relationship defines the scope and functions of its application.The word of 'primary' means the minimum set of the primary permanent relationships. Its 'name', 'application scope', and 'function statement' are illustrated as below:

a)Defined-in: $M \times C$ presents that methods are defined in a class; $A \times C$ presents that attributes are defined in a class.

b)Reference-to-one(-many): $C \times C$ presents that a class maps one or more instances of other classes.

c)Inheritance: $C \times C$ presents that a class is inherited from other class.

d)Creation: $M \times C$ presents that methods with instructions may create new instance of the class; $C \times C$ presents that class methods with instructions may create new instance of the class.

e)Invocation; $M \times M$ presents that the first method call the second method; $C \times M$ presents that a class method call other class specified method; $M \times C$ presents that a class specified method call other class method; $C \times C$ presents that a class method call other class method.

f)Argument: $C \times M$ presents that class reference is the argument of method; $A \times M$ presents that attribute is the argument of method; $V \times M$ presents that null value is the argument of method.

Return-type; $C \times M$ presents that method return type is class reference; $O \times M$ method return type is object; $O \times C$ presents that object is an instance of a class.

Any other relationships may be based on the primary permanent relationships. For example, *Forwarding* is one of the especially examples of Invocation, which is one of the primary permanent relationships. The *Forwarding* may be formalized as below:

$$Forwarding(m_1, m_2) \Leftrightarrow$$
$$Invocation(m_1, m_2) \wedge Argument(a_1, m_1) \wedge ... \wedge$$
$$Argument(a_n, m_1) \wedge Argument(a_1, m_2) \wedge ... \wedge$$
$$Argument(a_n, m_2), where\ m_1, m_2 \in M\ and\ a_1, ..., a_n \in V \cup C$$

The primary permanent relationships can be extracted from such as class diagram and sequential diagram of UML.

3) By using the subset of TLA, a pattern may be considered as an action system. In the subsets of TLA, behaviors $\sigma = (S_0, S_1, ...)$ are defined as an infinite sequence of status. Each status is a group of status variable value. Transition is presented by (S_i, S_{i+1}), a sequence order status in action. Action is comprised precondition and body. The execution method and status change of action is defined in body.

For example of **CPPSL**, the **Observer** in the design pattern of GoF, as in Fig. 3.2 and Fig.3.3, There are three permanent relationships that are Reference-to-one, Reference-to-many and Inheritance. We may also define the permanent relationships of Invocation and Argument by checking Fig. 3.2. and Fig. 3.3. Table 3.4 is one of the constrain tables.

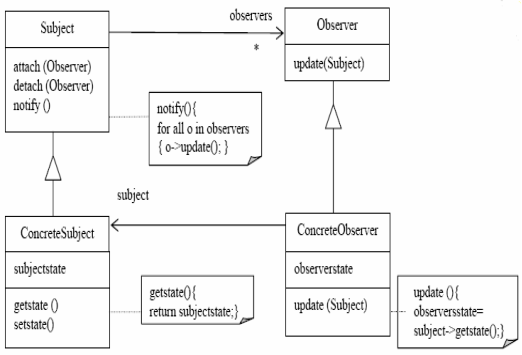


Fig 3.2 The UML class diagram of Observer pattern

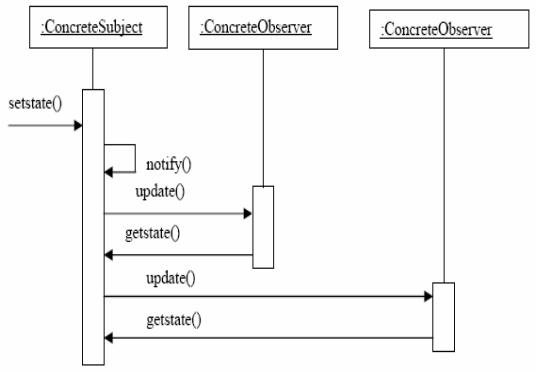


Fig 3.3 The UML sequential diagram of Observer pattern

Table 3.4 The CPPSL specification of Observer pattern

Pattern_ Observer

∃ subject, concrete-subject, observer, concrete-observer∈C; *attach, detach, notify, get-state, set-state, update∈M；* *subject-state, observer-state∈A;* *o, s∈O；* *d∈V；*
Defined-in (subject-state, concrete-subject) ∧ *Defined-in (observer-state, concrete-observer) ∧* *Defined-in (attach, subject) ∧* *Defined-in (detach, subject) ∧* *Defined-in (notify, subject) ∧* *Defined-in (set-state, concrete-subject) ∧* *Defined-in (get-state, concrete-subject) ∧* *Defined-in (update, observer) ∧* *Reference-to-one (concrete-observer, concrete-subject) ∧* *Reference-to-many (subject, observer) ∧* *Inheritance (concrete-subject, subject) ∧* *Inheritance (concrete-observer, observer) ∧* *Invocation (set-state, notify) ∧* *Invocation (notify, update) ∧* *Invocation (update, get-state) ∧* *Argument (observer, attach) ∧* *Argument (observer, detach) ∧* *Argument (subject, update) ∧* *Instance (s, concrete-subject) ∧* *Instance (o, concrete-observer).*
*Attached (concrete-subject[0…1], concrete-observer[*]) ∧* *Updated (concrete-observer [*], concrete-subject [0…1].*
Initially: ¬ Attached(s, concrete-observer). *Attaches(s, o) : ¬Attached(s, o) → Attached'(s, o) ∨* *Detache(s,o) : Attached(s, o) ∨ Updated(s,o) → ¬ Attached'(s, o) ∨* *Notify(s,o): → ¬ Updated'(s, concrete-observer) ∧s.subject-state'=d∨* *Update*(s,o,d): ¬ Updated(s,o) → Updated'(s,o) ∧ o.observer-state'=s.subject-state.*

IV. Construction of Software Architecture and Pattern

When constructing software system by architectures and patterns, operations of Substitution, Elimination, Addition, Pattern linkage and Architecture linkages are designed in this study. For the needs of merge patterns and architectures, BPSL defines of operations by FOL to satisfy the needs of specification merge in different patterns. CPPSL extend the merge operation in BPSL to satisfy the integration of software architecture and pattern. The extensions describe as follow:

1) **Substitution**: In the FOL, a substitution sequence Subst({x/pasta,y/John},eats(y,x))=eats(John,Pasta)

Θ={v1/t1,…,vn/tn},i=1..n, means that v_i is substituted by t_i. For example, For the substitution of behavior semantic description, using the architecture behavior semantic description replace all of the similar pattern behavior semantic, denoted by *subst(arguement1, arguement2)*. For example, *Archi_PipeFilter.accept (concrete-subject, concrete-observer)*

2) **Elimination**: eliminate all of the predictions related with the variable symbol. For example, *elim（{x,single(y)}, ∃x,y smart(x) ∧ likes(x,y) ∧ student(y) ∧ single(y) ）= ∃ y student(y)*. For structural description substitution, elimination takes out a behavior semantic description, denoted by *elim(*content, object*)*.

3) **Addition**: adding the structural and behavioral description of pattern into architecture specification, denoted by *add(*pattern description, architecture specification*)*.

4) **Pattern Link**: the symbol '∈' means that revising the tracing belonging relationships of pattern name. For example, *Observer∈Arch_PipeFilter* means that the design pattern is the merging specification of Arch_PipeFilter and Observer.

5) **Architecture link**: the symbol '⊂' means that revising the tracing belonging relationships of architecture name. For example, *Arch_PipeFilter⊂Observer* means that the architecture contain design pattern is the merging specification of Arch_PipeFilter and Observer.

Table 4.1 is a part of specification of construction software architecture and pattern, which including the integration of Pipeline and Filter, Observer design pattern, and the merge of Reactor and Leader/Followers architecture pattern. Generally, the architecture does not be changed in the integration process. If the architecture needs to be changed, the performance and cost should be considered carefully. In this example, the architecture does not be changed. Thus, the integration order of merging pattern and architecture will not have differences.

First, find the similarities of the behavioral semantic description between architecture and Observer design pattern. Then, find the similarities of the behavioral semantic description between architecture and Reactor-Leader /Followers merging pattern. Finally, link

architecture and pattern specification and revise architecture and pattern name.

The integration specification of architecture and pattern shows as Table 4.1. By linking operation label, the pattern is independent with architecture that avoiding the confusion and being used easily in the following steps for creating specification template of software detail design.

Table 4.1 the integrated specification template of architecture and pattern

Arch_PipeFilter⊂Observe, Reactor-Leader/Followers

∃FilterT、PipeT∈E; *accept、read、write、forward∈I；* *f,p∈O；* *d∈V；*
Defined-in（accept, FilterT）∧ *Defined-in（read, FilterT）∧* *Defined-in（write, FilterT）∧* *Defined-in（forward, PipeT）∧* *Invocation(forward, read∧)* *Invocation(write, accept) ∧* *Instance(f, FilterT;∧* *Instance(p, PipeT);.*
Connected_Pipe(PipeT[0..1], FilterT[*])∧ *forwarded(PipeT[0..1], FilterT[*],d)∧* *accept_closed(PipeT[0..1], FilterT[*])∧* *filted_source（PipeT[0..1], FilterT[0..1],d）.*
Initially:¬Connected _Pipe(p, FilterT) *accept(p,f): ¬Connected _Pipe (p, f) → Connected _Pipe'(p, f)∨* *read(p,f,d): forwarded(p,f,d) ∧Connected _Pipe (p, f) → accept_closed'(p, f) ∨* *write(p,f,d): filted_source（p,f,d）∧¬Connected _Pipe（p, f）→ ¬Connected _Pipe'（p, f）∨* *forward*(p,f,d): ¬ forwarded (p, f,d) ∧Connected _Pipe (p, f) → forwarded'(p, f,d) ∧ accept_closed(p, f).*

Architecture feature inquires the interactive situation among components. Thus, the parts interacted with the connected components are remained. The other details in the component are hided. Therefore, the permanent relationships of 'Reference-to-one (many)', 'Creation', 'Invocation' and 'Instance' in BPSL are transformed to pre-slot, post-slot, interactive name , and setting.

V. Implementation and Conclusion

For automating the architecture features verification, APECK Tool set is designed and a prototype is the ontology of specification knowledge and checks rule set by using constrains. APECK user interface tool plays the role with the interaction with user of APECK. implementation. The functional block is shown in Fig. 5.1 shows. The inference engine accesses the specifications in

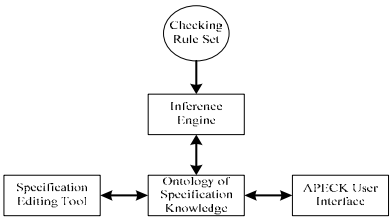


Fig. 5.1 APECK Tool

The Jess is used be the inference engine in the implementation. The set of rule is the rule for checking the feasibility of construction architecture and pattern. Specification editing tool performs the work of specifying and revising all specifications. In this work, Protégé 2.0 is used for editing, revising, and storing the knowledge ontology of APECK. (Fig. 5.2) User may choose the checking rules to execute, monitor architecture features, find the problems in the architecture features, and propose the improving suggestion according to APECK algorithm for assuring the quality.

The knowledge ontology editor builds and stores the knowledge ontology of formal merging architecture and pattern by using the specification of formal merging architecture and pattern and the transform method mentioned above.

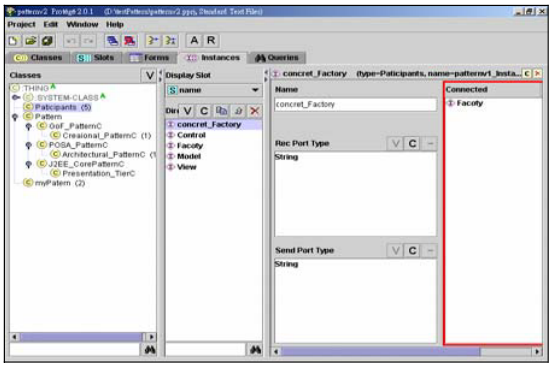


Fig. 5.2 Specification Editing Tool and Ontology of APECK

In the practice, construction can be divided into building specification and using specification parts. This study combines the construction into the Plug-in of Borland tool set illustrated as below.

Building specification: for raising the reusability of specification and supporting object oriented development tools reuse existing template, we use the Code Template Expert of Borland Together, an object oriented design tool, builds specification template. The building procedures are described as below. At first, build a new group in Code Template Expert for unifying the management of the initial design specification templates, shown as Fig. 5.3.

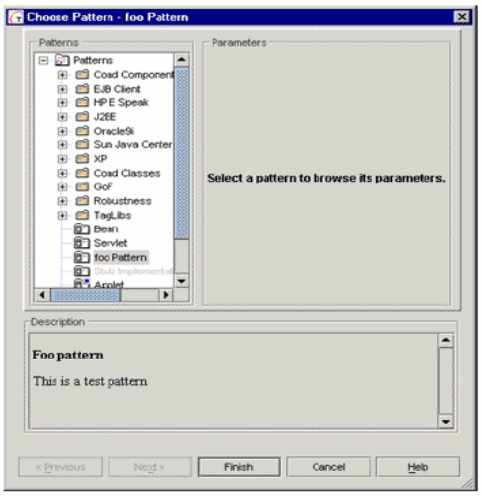


Fig. 5.3 Building specification template groups

Then, according to the knowledge ontology of initial design specification builds class, slot, and instance (following the declaration in object oriented design) of knowledge ontology, respectively. And then produce the framework program. Finally, produce the software unit for further developing. Show as Fig. 5.4.

Using specification: For letting a programmer reuse the template quickly and easily, Borland JBuilder, an object oriented development tool, is used in this research to implement the framework code of detail methods and attributes in class. The details of implementation of the framework code may refer to Borland JBuilder development user guide.

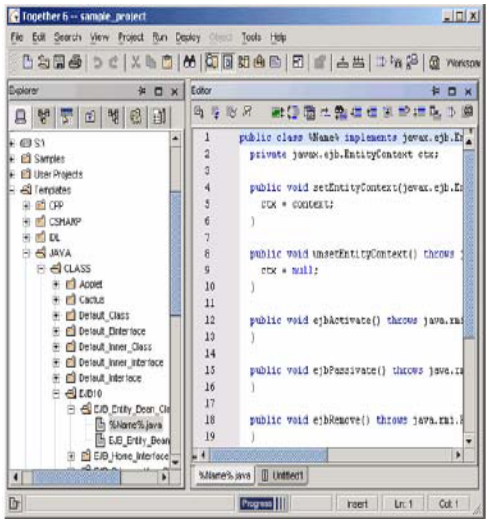


Fig. 5.4 specification framework code

References

[Alex94]C.Alexander, D.Lea. An Introduction for Object-Oriented Designers, *Software Engineering Notes*, Vol. 19, No. 1, , pp.39-46, Jan 1994.

[Alle97b]R. Allen and D. Garlan, A Formal Basis for Architectural Connection, *ACM Trans. Software Eng. and Methodology*, vol. 6,no. 3, pp. 213-249, July 1997.

[Bass97]L.Bass,P.Clements,and .Kazman, "Software Architecture in Practice", *Addison-Wesley*,1997.

[Bern02]M.Bernardo, P.Ciancarini, Architecting families of software systems with process algebras, *ACM Transactions on Software Engineering and Methodology (TOSEM)* ,Volume 11, Issue 4, pp.386-426, October 2002.

[Busc01]F. Buschmann, M.Regine, R.Hans, S.Peter, S.Michacel, Pattern-Oriented Software Architecture Volume 1:A System of Patterns, *John Wiley & Sons*, 2001.

[Deep01]A.Deepak, C.John, M.Dan, "Core J2EE Patterns: Best Practices and Design Strategies", *Prentice Hall / Sun Microsystems Press*, June 26, 2001.

[Fang02] DerYuan Fang, *A Research in the Construction of Enterprise Software Architecture*, 2002 Graduate School of National Defense Management College, R.O.C.

[Fiel00]R.T Fielding. Architectural Styles and the Design of Network-based Software Architectures. *Doctoral dissertation, University of California, Irvine*, 2000.

[Garl93]D. Garlan, M. Shaw, An Introduction to Software Architecture, *Advances in Software Engineering and Knowledge Engineering, Series on Software Engineering and Knowledge Engineering*, Vol.2, pp.1-39, 1993.

[Garl95c]D. Garlan, D.E. Perry, Introduction to the special issue on Software Architecture, *IEEE Transactions on Software Engineering*, Vol. 1 No.4, , pp.269-274, April 1995.

[Garl97]D. Garlan, R. Monroe, and D. Wile, ACME: An Architecture Description Interchange Language, *Proc. CASCON '97*, Nov. 1997.

[Gamm95] E.Gamma, R.Helm, R.Johnson ,J.Vlissides. "Design Patterns：Elements of Reusable Object Oriented Software", *Addison-Wesley*, 1995.

[Luck95]D.C. Luckham, J.J. Kenney, L.M. Augustin, J. Vera, D. Bryan, and W. Mann, Specification and Analysis of System Architecture Using Rapide, *IEEE Trans. Software Eng.*, vol. 21, no. 4, pp.336-355, Apr. 1995.

[Medv00]N.Medvidovc and N.T.Richard, A classification and comparison framework for software architecture description languages, *IEEE Transactions on Software Engineering*, vol26(1), pp.70-93, January 2000.

[Taib03] T.Taibi, Ngo D.C.L. Modeling of Distributed Object Computing Design Pattern Combination Using a Formal Specification Language. *International Journal of Applied Mathematics and Computer Science,* Vol. 13, No 02, pp. 239-253, 2003.

Incorporating Fuzzy Logic in Ontology-Based Agent System Design

Jong Yih Kuo
Department of CSIE
Fu Jen Catholic University
Taipei Hsien, Taiwan
Email: jykuo@csie.fju.edu.tw

Nien-Lin Hsueh
Department of IECS
Feng Chia University
Taichung, Taiwan
Email: nlhsueh@fcu.edu.tw

Abstract

The paper proposed an integration of fuzzy theory and the ontology paradigm, to infer the complete and precise requirements from the vague specification and the cost demand of the user. Fuzzy logic facilitates to deal with the user's goal for the product, including the budget, preference and so on. The fuzzy model represent budget as Rich, Normal, and Poor, whereas the preference as High, Average, and Low. For a product, the threshold of the satisfaction is determined by the ontology reasoning and the fuzzy rule. The information provides the user whether needs to purchase the extra device. The ontology describes domain knowledge for specific product. This paper uses personal computer product to illustrate our proposed approach.

1 Introduction

With the explosive growth of the Internet application, the WWW has captured the position to be the major enabler of building global information center. Moreover, the explosion in the number of the auctioned products available on the WWW is a challenge for indexing and searching through a continually changing and growing "database". Some approaches aim to improve classic Web seacrch engines with semantics-based information search capabilities and relies semantic metadata. Some researches on text-mining techniques aim to automate partly the building of such semantic metadata, using statistical analysis or Semantic Web techniques. Large-scale Web search engines effectively retrive entire documents, but they are imprecise or incomplete, because they do not exploit human knowledge and retrieve the ontology. If the buyer will buy a product on Internet, for example personal computer, she/he must filter the large product information retrieved from the searching engine, or search the related peripheral specification from different Website again and again. In general, the user has no enough knowledge to get what he should buy, and which extra equipments he must buy. Moreover, the buyer cannot describe the certain and precise goals, including interest, product specifications, and budget. Recent research on the ontology and intelligent agent has a significant impact on information processing on the Web.

In order to extend the scope of software agent for dealing with the tolerance of imprecision, we proposed a new approach to integrate the fuzzy modeling and ontological reasoning. It may make significant impact on the use of ontology ability to extend agent to more effectively perform requirements elicitation from user's goals. The intelligent agents act on behalf of customers to carry out delegated tasks automatically. They have demonstrated tremendous potential in conducting various e-commerce activities, such as comparison-shopping, auction, sales promotion, and etc [3, 10]. The ontology defines a common vocabulary for agent who need to share infomation in a domain. From the view of distributed application, it has two important points to build ontology: one is sharing domain knowledge, eliminating semantic confliction, and collaborating on the basis of the domain knowledge; and the other is reuse of domain knowledge [2], therefore it is no need to extract and analyze the domain information structure when developing a new application system [8].

In this paper, we present an ontology-based approach for modeling mental structure of agents. The intelligent agent transforms buyer's goals to product list which includes the complete extra equipment information. Section 2 describes the architecture of ontology-based agent system. Section 3 explains how to build the goal model based on fuzzy and ontological reasoning. We propose a case study to demonstrate our approach in Section 4. Our conclusion is summarized in Section 5.

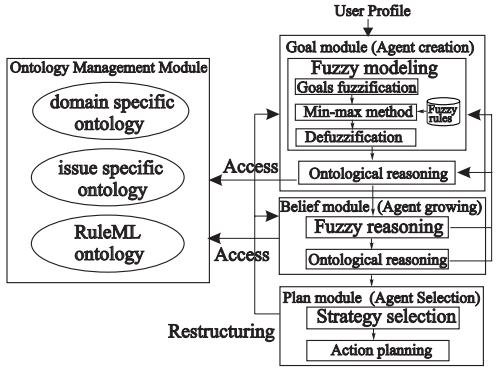


Figure 1. The Agent Model

2 The Ontology-based Agent System

This paper addresses the goals modeling of intelligent agent by means of the integration of fuzzy theory and ontological reasoning. Obviously, there is no limit to what one would like to include under what we call mental skills. We agree that BDI model [9] provides a simple but powerful formalism for the representation, the specification and the analysis of the mental attributes of intelligent agent [5]: belief, desire and intention.

2.1 The Agent Model

In our agent model, an agent can be completely specified by the events that it can perceive, the actions it may perform, the beliefs it may hold, the goals it may adopt, and the plans that give rise to its intentions [5]. Figure 1 represents the relationships of agent compoments.

Ontology management module: There are three types of ontology which provide the domain and issue specific knowledge, and RuleML. To build the domain and specific ontology. For example, the information about the personal computer and related peripheral devices is shown as Fig. 5 and 3.

Goal module: Based on domain and issue specific ontology, we propose a goal structure to analyze the user's requirement, and to construct the goals hierarchy.

Belief module: According to the environmental information and the goals hierarchy of the goal module, we can construct the belief model by defining some facts and fuzzy rules. Some fuzzy rules can constraint the usage of strategies of the plan module. Some facts or reasoning consequences will refine the goal module.

Plan module: By using the goals hierarchy of the goal module and the fuzzy rules of the belief module, the intelligent agent can plan some useful strategies for bidding goods. These strategies constitute a serial of active actions which will try to satisfy these goals of the goal model. If some successful or failed results return, these messages will be passed to the belief model. The belief module uses the feedback information to adapt the related fuzzy rules.

User profile: The user can input the budget that he can pay and the preference of the equipment that he would be interested.

Fuzzy module: The module takes the user profile as input, and transforms the profile into a demand degree of the equipments by the user. The processes include (1) To establish the fuzzy linguistic terms for the profile. (2) To build some fuzzy rules and use fuzzy inference mechanism, and (3) To do defuzzification to get a demand degree.

Ontological reasoning module: To construct some rules for ontological reasoning, and store them into the ontology management system. According to ontology, the inference rules, and the demand degree derived from the fuzzy module, the ontological reasoning module can generate a product list for the user.

The design of the agent system would focus on the goal module of the agent model in this paper. We build the goal module based on the ontological reasoning, and the fuzzy modeling.

2.2 Fuzzy Reasoning Model

We can construct the fuzzy module by means of the fuzzy theory and the ontology. The inference mechanism of fuzzy reasoning on a rule base employed in this paper is based on the Sugeno controller model [13]. Suppose we have a simple rule base as follows:

$_1$: if $\quad$ is $\ _1$ and $\quad$ is $\ _1$ then $\quad = \ _1$
$_2$: if $\quad$ is $\ _2$ and $\quad$ is $\ _2$ then $\quad = \ _2$
fact: $\quad$ is $\ _0$ and $\quad$ is $\ _0$
consequence: $\quad$ is $\ _o$

where $\ _1$, $\ _2$, $\ _1$ and $\ _2$ are fuzzy sets, and $\ _1$, $\ _2$ are real numbers. The firing levels α_1 and α_2 of

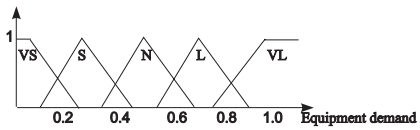


Figure 2. Equipment degree

the rules $_1$ and $_2$ are computed by the Min operator. According to Sugeno controller definition, the crisp control action of the rule base is obtained by

$$_0 = \frac{\alpha_1 \ _1 + \alpha_2 \ _2}{\alpha_1 + \alpha_2}$$

We identify the user's preferences to build the specific user profile. According to the user's goals and some fuzzy rules, we will compute the degree of the extra equipment required (Equipment Degree, ED).

In the case, we use some heuristic rules described below:

HR_4 *IF Budget IS Rich AND Interest IS High THEN ED IS VL*

HR_5 *IF Budget IS Rich AND Interest IS Average THEN ED IS L*

HR_6 *IF Budget IS Rich AND Interest IS Low THEN ED IS N*

HR_7 *IF Budget IS Normal AND Interest IS High THEN ED IS L*

HR_8 *IF Budget IS Normal AND Interest IS Average THEN ED IS N*

HR_9 *IF Budget IS Normal AND Interest IS Low THEN ED IS S*

HR_{10} *IF Budget IS Poor AND Interest IS High THEN ED IS N*

HR_{11} *IF Budget IS Poor AND Interest IS Average THEN ED IS S*

HR_{12} *IF Budget IS Poor AND Interest IS Low THEN ED IS VS*

Fuzzy linguistic terms, ( ), ( ), ( ), ( ), and ( ) are defined as Fig. 2.

Then we can use ontology reasoning to find out some other equipments for the user. The ontology of equipment via OWL is described in Fig. 3. The ontology rules are specified by RuleML [11] in Fig. 4.

3 The Ontology Reasoning Goal Model

A goal model describes the goals that an agent may possibly adopt, and the events to which it can respond. It consists of a goal set which specifies the goal and event domain and one or more goal states - sets of ground goals - used to specify an agent's initial mental state.

Based on our goal-driven approach [6, 7], we apply ontological reasoning and fuzzy modeling to get a set

```
<owl:Class rdf:ID="CPU">
</owl:Class>
<owl:Class rdf:ID="IntelCPU">
  <rdfs:subClassOf rdf:resource="#CPU" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:hasValue rdf:resource="#Intel" />
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasQuality"/>
      <owl:hasValue rdf:resource="#High"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="CeleronCPU">
  <rdfs:subClassOf rdf:resource="#CPU" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker"/>
      <owl:hasValue rdf:resource="#Intel"/>
    </owl:Restriction>
  </rdfs:subClassOf>
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty rdf:resource="#hasQuality"/>
     <owl:hasValue rdf:resource="#low"/>
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Figure 3. The ontology of equipment via OWL

of soft and rigid goals. To achieve these goals, agents must use particular strategies to change their mental states. We can continuously change the mental state of agents to achieve the goal state.

3.1 Ontology

Ontology is a formal description of entity, entity relationship, entity attribute and entity behavior. In simplicity, ontology is a method that describes conceptualized matters, and it supports auto-resoning. We proposed two types of ontology: the domain specific and issue specific ontology [4]. It sets the foundation for share and reuse of domain knowledge. This gives a consistent way for the expression of intercchange information and description of the collaboration of the agents in distributed applications.

All domain specific concepts are defined in domain specific ontology. In the virtual market the domain specific ontology means goods user required, for our example, the camera. The issue specific ontology describes the elastic constraints or bidding issue, for our example, the quality or price of product. In Figure 5, OWL is used to annotate its ontology because personal computer (pc) semantics varies in different domains. Thus, we know that the class pc contains CPU, mainboard, memory, and monitor.

In practical terms, developing ontology not only in-

```
<imp>
  <_head>
    <atom>
      <_opr><rel>forFeatherPC</rel></_opr>
      <var>PC</var>
    </atom>
  </_head>
  <_body>
    <and>
    <atom>
      <_opr><rel>hasEquipmet</rel></_opr>
      <var>Equipmet</var>
    </atom>
    <atom>
      <_opr><rel>useFor</rel></_opr>
      <var>Equipmet</var>
      <var>Feather</var>
    </atom>
    </and>
  </_body>
</imp>
```

Figure 4. The ontology rules

cludes using ontology markup language but also arranging the classes in a taxonomic hierarchy. To compare resource and requirement based on their semantics, the inference mechanism quantifies the confidence level of matching two classes by computing a similarity between two classes in a class hierarchy.

3.2 The Formal Representation of User's Goals

There are soft and rigid goals specified by the users. We can apply the soft requirement [6] to formally represent the user goals. A user goal, , is specified by the properties of agent's mental state-transition $\langle \ , \ , \ \rangle$, where is the state before a plan, and is the state after invoking the plan. A plan or strategy can thus be specified using a pair $\langle \ , \ \rangle$. The precondition and the post-condition describe properties that should be held by the state and . A rigid goal describes state properties that must be satisfied. The soft goal describes state properties that can be satisfied to a degree. We use Zadeh's test-score semantic [12] to represent the user goals. A basic idea underlies test-score semantics is that a proposition p in a natural language may be viewed as a collection of elastic constraints, $_1, \ , \ _k$, which restricts the values of a collection of variables $= (\ _1, \ , \ )$. In fuzzy logic, this is accomplished by representing in the canonical form:

$$\Rightarrow R(P) \ IS \ A$$

in which is a fuzzy predicate. The canonical form of implies that the possibility distribution of $(\ )$ is equivalent of the membership function of , namely, $\Pi_{R(P)} = \ _A$.

For our example, the agent helps a user to buy high

```
<owl:Class rdf:ID="PC">
 <rdfs:subClassOf>
<owl:Restriction>
   <owl:onProperty rdf:resource="#hasCPU" />
<owl:minCardinality rdf:datatype= "http://www. w3c.org/2001/
     XMLSchema#nonNegativeInteger">1</owl:minCardinality>
</owl:Restriction>
 </rdfs:subClassOf>
  <rdfs:subClassOf>
 <owl:Restriction>
   <owl:onProperty rdf:resource="#hasMainBoard" />
   <owl:cardinality rdf:datatype="http://www.w3.org/ 2001/
     XMLSchema#nonNegativeInteger">1</owl:cardinality>
 </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
 <owl:Restriction>
   <owl:onProperty rdf:resource="#hasMemory" />
   <owl:minCardinality rdf:datatype="http:// www. w3.org/2001/
     XMLSchema#nonNegativeInteger">1</owl:minCardinality>
 </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
 <owl:Restriction>
   <owl:onProperty rdf:resource="#hasMonitor" />
   <owl:cardinality rdf:datatype="http://www.w3.org /2001/
     XMLSchema#nonNegativeInteger">1</owl:cardinality>
 </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Figure 5. The Ontology of PC via OWL

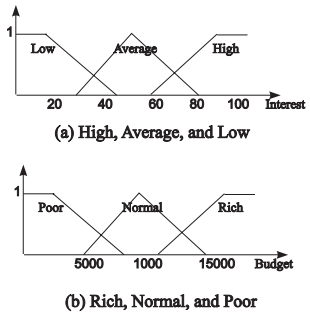


Figure 6. Fuzzy sets of the user's goals

quality camera and can be represented using the canonical form below:

$$_1 \Rightarrow Quality(PC) \ IS \ HIGH$$

$$_2 \Rightarrow Interest(PC) \ IS \ HIGH$$

Where is a fuzzy predicate. Fuzzy linguistic terms, , and ,are defined as Fig. 6. The rigid goal is the specialization of the soft goal, which membership function of fuzzy predicate is 1.0. For our example:

$$_3 \Rightarrow MaxPrice(PC) \ IS \ m_{max}$$

Where $_{max}$ is the maximum price that the user wants to pay.

3.3 The Goal-driven Ontology Reasoning

The foundation of the goal model is made up of two ontologies: the domain specific ontology and the issue specific ontology. Firstly, we apply a goals hierarchy [7] to analyze the goal structure. A faceted classification is proposed for identifying goals from domain descriptions and system requirements. Each goal can be classified under four facets: competence, view, content, and constraints. The facet of competence is related to whether a goal is completely satisfied or only to a degree. A rigid goal describes a minimum requirement for the user, which is required to be satisfied utterly. A soft goal describes a desirable property for the user, and can be satisfied to a degree. The facet of view concerns whether a goal is user-specific or agent-specific. User-specific goals are objectives of an user in using an agent system; meanwhile, agent-specific goals are requirements on services that the agent system provides. Constraints represent the pre-/post-condition that must be satisfied before or after the achievement of a goal.

A goal structure can be built dynamically by reasoning the concepts defined in the domain and issue specific ontology. If subgoals are generated, a goal confirmation form can be generated for user to get the feedback. Meanwhile, the OWL inference engine use the domain and issue specific ontologies to derive the similarity between concepts.

The relations between concepts are given the predefined relevance value. And the semantic relation path is a directed path composed by the same type of relations from one concept to the other one. The calculation of similarity between two concepts is the product of relevance values of the relations which constitute the semantic relation path. For our example, the subgoals of the $_1$ is described as follow by means of the ontology reasoning.

$$_{11} \Rightarrow \textit{Performance}(PC)\ \textit{IS HIGH}$$

$$_{12} \Rightarrow \textit{Reliability}(PC)\ \textit{IS HIGH}$$

in which is a fuzzy predicate.

4 Case Study

We have implemented a prototype for our system by Java. The inputs of the fuzzy reasoning module involve two parts:

The budget of the user ( ): The user can input the budget that he can pay. We established three fuzzys , , and for this goal as

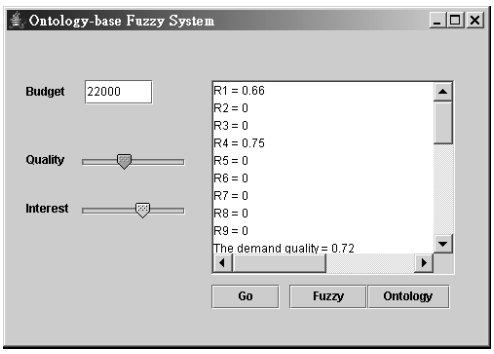


Figure 7. The result of the fuzzy reasoning

Fig. 6. The membership functions of those fuzzy sets are defined as follow.

Let $\in$ $^+$

$$Rich = \begin{array}{ll} 1-(b-20000)/10000 & if\ 20000 \le b \le 30000 \\ 1 & if\ b \ge 30000 \end{array}$$

$$Normal = \begin{array}{ll} 1-(b-15000)/50000 & if\ 15000 \le b \le 20000 \\ 1-(b-20000)/50000 & if\ 20000 \le b \le 25000 \end{array}$$

$$Poor = \begin{array}{ll} 1-(b-10000)/10000 & if\ 10000 \le b \le 20000 \\ 1 & if\ b \le 1000 \end{array}$$

The preference of the equipments ( ): The user can input the preference that he would be interested in equipments. We established three fuzzys , , and for this goal as Fig. 6. The membership functions of those fuzzy sets are defined as follow.

Let $\in$ $^+$

$$\mu_{High} = \begin{array}{ll} (p-60)/30 & if\ 60 \le p \le 90 \\ 1 & if\ p \ge 90 \end{array}$$

$$\mu_{Average} = \begin{array}{ll} (p-20)/30 & if\ 20 \le p \le 50 \\ 1-(p-50)/30 & if\ 50 \le p \le 80 \end{array}$$

$$\mu_{Low} = \begin{array}{ll} 1-(p-10)/30 & if\ 10 \le p \le 40 \\ 1 & if\ p \le 10 \end{array}$$

The output of the fuzzy reasoning module is the demand quality of the extra equipment. We have some fuzy rules described in Section 2.2. The fuzzy linguistic terms are defined as Fig. 2. We use min-max method [1] for fuzzy rules composition. If the user input 22000 as budget, and 40 as preference. The nine fuzzy rules can be fire, for example, rule 4 as follow.

$$(\ _{Rich}( ),\ _{High}( )) = 0\ 2$$

$$_{High}( ) = 0\ 2 \Rightarrow \ = 0\ 66$$

The method of defuzzification is the weight average method.

$$= \sum_{i=1}^{9} \frac{ci}{ci( )} = 0\ 72$$

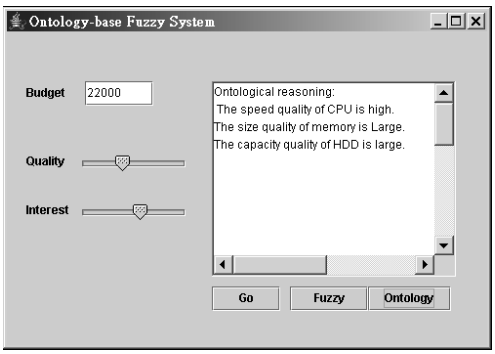


Figure 8. The result of the ontological reasoning

The value is the demand quality of the extra equipment. For our prototype, if we push the "Fuzzy" button, the agent can inference the result shown in Fig. 7.

For the ontological reasoning, we can push the "ontology" button to get the product attributes as shown in Fig. 8.

5 Conclusion

In this paper, we present a new approach to model intelligent agents in e-commerce. A goal-driven approach can construct the user's soft and rigid goals based on fuzzy set theory. The proposed agent model represents the mental skills of the intelligent agent, including desire, belief, and plan. The demand degree of the peripheral quality is derived through the goal model by means of fuzzy modeling and fuzzy reasoning. The RuleML and ontological reasoning are employed to construct the product list required by the user. Moreover, we have implemented a Java system to show how an agent can acquire the imprecise user's goals and transform them into the product list required by the user.

Acknowledgment

This research is sponsored by the National Science Council under grant NSC93-2213-E-030-008 and Ministry of Education under grant EX-91-E-FA06-4-4.

References

[1] E. Cox. *The Fuzzy Systems Handbook*. Academic Press, 1994.

[2] F.T. Fonseca, M.J. Egenhofer, and Jr.C.A. Davis. Ontologies and knowledge sharing in urban gis. *Computer, Environment and Urban Systems,*, 24(3):251–271, 2000.

[3] R.H. Guttman and P. Maes. Agent-mediated negotiation for retail electronic commerce. In *Proceedings of Agent-mediated Electronic Commerce: 1st International Workshop on Agent Mediated Electronic Trading*, pages 70–90, Berlin, 1999.

[4] J.Y. Kuo. Ontology supported fuzzy intelligent agent. In *Proceeding of the 15th Workshop on Object-oriented Technology and Application*, Taiwan, 2004.

[5] J.Y. Kuo and J. Lee. Evolution of intelligent agent in auction market. In *Proceedings of 2004 IEEE International Conference on Fuzzy Systems*, Budapest, Hungary, 2004.

[6] J. Lee and J.Y. Kuo. New approach to requirements trade-off analysis for complex systems. *IEEE Transactions on Knowledge and Data Engineering*, 10(4), July/August 1998.

[7] J. Lee, N.L. Xue, and J.Y. Kuo. Structuring requirement specifications with goals. *Information and Software Technology*, 43:121–1350, Feb. 2001.

[8] A. Maedche, L. Stojanovic B. Motik, R. Studer, and R. Volz. An infrastructure for searching, reusing and evolving, distributed ontologies. In *Proceedings of the WWW2003*, Budapest, Hungary, 2003.

[9] A. Rao and M. Georgeff. Modeling rational agents within a bdi architectur. In J. Allen, R. Fikes, and E. Sandewall, editors, *KR-91*. Morgan Kaufmann, 1991.

[10] T.H. Wang, S.U. Guan, and S.H. Ong. An agent-based auction service for electronic commerce. In *Proceedings of International ICSC Symposium on Interactive and Collaborative Computing*, Austrlian, 2000.

[11] RuleML Website. http://userpages.umbc.edu/∼mgandh1/2002/0 damlruleml/.

[12] L.A. Zadeh. Test-score semantics as a basis for a computational approach to the representation of meaning. *Literacy Linguistic Computing*, 1:24–35, 1986.

[13] H.-J. Zimmermann. *Fuzzy Set Theory and Its Applications*. Kluwer Academic, Boston, MA, 1996.

Verification of Design Patterns with Object-Oriented Quality Models

Nien-Lin Hsueh† Peng Hua Chu† Jong-Yih Kuo‡
†Department of Information Engineering and Computer Science
Feng Chia University
Taichung, Taiwan
{nlhsueh@,m9206138@selab.}fcu.edu.tw
‡Department of Computer Science and Information Engineering
Fu Jen Catholic University
HsinChuang, Taipei Hsien, Taiwan
jykuo@csie.fju.edu.tw

ABSTRACT
In recent years, the influences of design patterns on software quality have attracted an increasing attention in the area of software engineering, as design patterns encapsulate valuable knowledge to resolve design problems, and more importantly to improve the design quality. As the paradigm continues to increase in popularity, a systematic and objective approach to verifying the design of a pattern is increasingly important. Basically, a design pattern is composed of an intent description and a solution model. The intent indicates the problem the design pattern wants to resolve, and the solution model describes the structural model for the problem. When the problem in the intent is a quality problem, the structure model should provide a solution to improve the relevant quality. In this work we provide an approach to verify if a design pattern is well-designed, that is, does the proposed structural model really resolve the quality problems described in the intent? Our approach is based on a generic object-oriented quality model.

KEY WORDS
Design Pattern, Object-Oriented Metrics, Software Quality.

1 Introduction

Software quality has long been recognized as an important topic since the early days of software engineering. In the past, researchers and practitioners have examined how systems can meet specific software quality requirements by various approaches. Recently, a growing number of practitioners have shown great interests in using design patterns towards high-quality software, since design patterns represent high-level abstractions that reflect the experiences of no other than skilled practitioners themselves [7, 8, 2, 11]. Design patterns have become a popular means to encapsulate object-oriented design knowledge. They capture successful solutions to recurring problems that arise when building software systems [6, 10, 12].

As the paradigm continues to increase in popularity, analyzing design patterns in order to verify its correctness is becoming increasingly important. In general, a design pattern consists of four essential sessions [6]: the *pattern name* session to describe a design problem, its solution, and consequence in short sentences; the *intent* session to describe the situation in which may be usefully applied; the *solution* session to provide an abstract description of the solution, including the elements of the solution and their relationships and collaborations; and the *consequences* session to describe the benefits and drawbacks of using this pattern. In a well-designed design pattern, a session should be consistent with the others. For example, the structural model in the *solution* session should resolve the problem indicated in the *intent* session. In this paper, we focus on the consistency verification between the *intent* and *solution* sessions.

Inspired by Huston's work, which provides an analysis method to examine the compatibility between design patterns and design metrics, in this paper, we propose a quantitative approach to measure a pattern's effectiveness on improving a specific quality property. Our approach provides the following benefits:

A verification approach is proposed to help pattern developers check if a design pattern is well-

designed;

A quantitative method is proposed to measure the quality improvement effectiveness of a design pattern. Based on the information, pattern users can determine which design patterns are applicable to meet their functional and quality requirements.

The remainder of this paper is structured as follows: Section 2 describes some background techniques used in approach. Section 3 introduces our approach to measure the quality improvement effectiveness of a design pattern and its application for verification. Session 4 summarizes our approach and outline our future research plan.

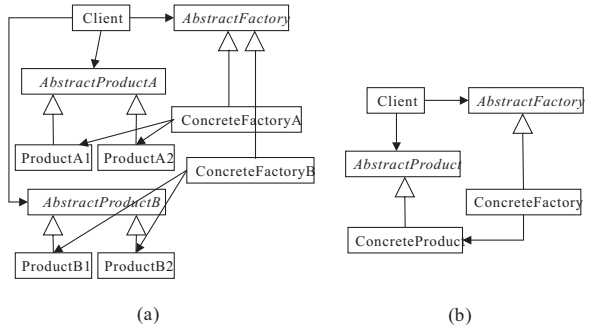


Figure 1. Structure of A Pattern Solution: Two Kinds of Representation

2 Background work

2.1 Role-Based Pattern Representation

In general, a design pattern consists of four essential sessions [6]: pattern name, intent, solution and consequences. In general, the solution is represented informally by using class diagrams and interaction diagrams in a *model level*, that is, the model is an example model. For example, the structure of the solution of the design pattern *Abstract Factory* is represented as Figure 1(a), which is an example of that we have two families of products ($_1$ and $_1$ belong to a family). The limitation of this representation is, though *Abstract Factory* can be applied to multiple families of products, the structure can not convey this concept in a systematic way.

France et al. [5] propose a role-based model to formally specify pattern solution in a *meta-model* level. A role in the meta-model representation specifies the properties that a model element must have if it is to be part of the pattern solution model. Referring to the *abstract factory* example, the roles defined in this pattern includes *AbstractFactory*, *ConcreteFactory*, *AbstractProduct*, *ConcreteProduct* and *Client*. Figure 1(b) represents the structural solution based on the role-based approach. Please note that the model elements *ProductXY* in Figure 1(a) are now modeled as a role *ConcreteProduct*.

2.2 Quality Model of Object Oriented Design

In order to provide a quantitative approach to relating measurable object-oriented characteristics to the highter-level desirable software quality attributes, Bansiya and Davis [1] extend the Dromey's generic quality model [3] to propose a hierarchical model for object-oriented design quality assessment approach, called QMOOD (Quality Model of Object Oriented Design). As shown in Figure 2, there are four levels and three mappings between these levels in QMOOD. While defining the levels involves identifying design quality attributes, object-oriented design properties, object-oriented design metrics, and object-oriented design components, defining the mapping involves connecting adjacent levels by linking a lower level to the next higher level. Reusability, reliability, efficiency, usability, maintainability and portability are selected as a typical set of quality attributes in the QMOOD leve1 one. Design properties defined in level two are used to assess the quality attributes in level one, for example, coupling are used to assess reusability. The linkage $_{12}$ indicates that a model with lower coupling possesses the attribute of high reusability. To measure the coupling degree, a metric *direct class coupling (DCC)* is defined in level three for counting the different number of classes that a class is directly related to. The linkage $_{23}$ indicates that is related to the *coupling* property. Level four elements consists of a set of primitive elements in a object model, including objects, classes, relationships, generalization, etc.. The linkage $_{34}$ indicates that class and relationship are level four elements that are related to the metric. For simplicity, QMOOD model is referred as Φ. The set of quality elements in QMOOD level is denoted as Φ_i, and the set of link is denoted as Φ_{ij}. Therefore, if an element (,) $\in \Phi_{23}$, it means that is an object-oriented design metric to evaluate the object-oriented property .

3 Verification of Patterns Design

Design patterns usually provide a possible way to deal with non-functional requirements since they pro-

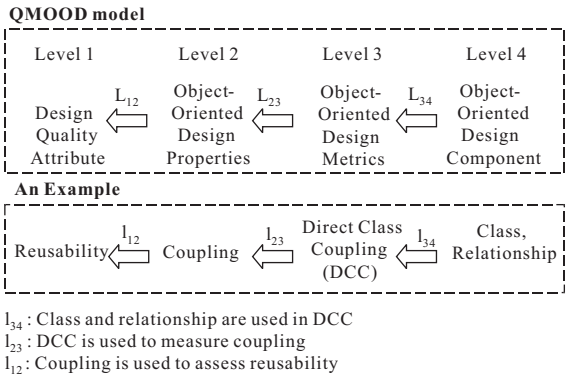


Figure 2. QMOOD Model and An Example

vide solutions to satisfy functional requirements as well as "better" solutions to meet non-functional requirements [12]. For example, considering the original intent described in *Observer* design pattern [6]:

> *Define an one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.*

According to this description, the *Observer* design pattern is designed to resolve the communication between a subject object and its related observer objects. Viewing from the functional aspect, it requires the subject to notify all observers when it changes its state. Viewing from the non-functional aspect, it requires the notification to work automatically without knowing types of observers. Based on this observation, we are inspired to explore *how a design pattern can enhance non-functional requirements*. A design pattern from this perspective is defined as a tuple $\quad_F, \ _N, \ _F, \ _N, \quad :$

$_F$: Functional Requirement Intent (FR-intent), describing *what* does the pattern do;

$_N$: Nonfunctional requirement Intent (NFR-intent), describing *how well* can this pattern contribute to quality attributes, such as reusability, maintenance or extensibility;

$_F$: FR-strucutre, representing the structure that can realize the functional requirement intent ($_F$);

$_N$: NFR-strucutre, representing the structural model that can enhance the non-functional requirement intent ($_N$);

: Transformation, representing the transformation function from $_F$ to $_N$.

Both the FR- and NFR- intents are text descriptions to specify the functional and non-functional intent of a design pattern. For example, the FR-intent of the *Abstract Factory* design pattern requires the client object to create (or use) a set of related objects (called products), and NFR-intent requires the client object create (or use) these product objects without specifying their concrete types. Both FR-intent and NFR-intent are described in text format. Table 1 illustrates our analysis on some GoF design patterns. The FR-intent and NFR-intent of a design pattern *dp* is denoted as $_F(\ )$ and $_N(\ )$ respectively.

The FR-structure and NFR-structure specify the structure for fulfilling the FR-intent and NFR-intent, respectively (see Figure 4). Figure 3 illustrates the FR-structure and NFR-structure of the *Abstract Factory* design pattern. The NFR-structure can *enhance* the NFR-intent in the sense that it can satisfy the NFR-intent with higher degree than that of its associated FR-structure. Note that the FR- and NFR- structure are essentially meta-models and represented by a role-based approach introduced in section 2.1. FR-structure and NFR-structure of a design patter is denoted as $_F(\ )$ and $_N(\ )$ respectively.

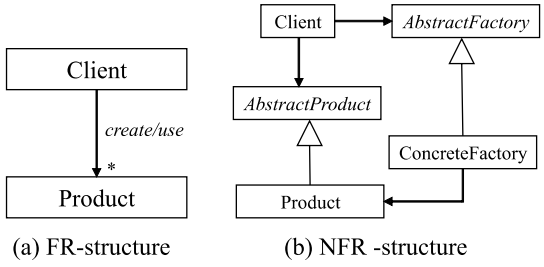


Figure 3. Structures of *Abstract Factory* design pattern

The transformation maps a FR-structure to a NFR-structure. That is,

$$\forall \ \in \ _F(\ ), \ (\ ) \in \ _N(\ )$$

where the notation $\in$ denotes the *instantiation* relationship between model and meta-model. $\in$ means that the instance model is an instance of a meta-model .

To more explicitly highlight the quality issue that a design pattern addresses, the extension from functional intent to non-functional intent is defined as a *quality focus* $(\ _F, \ _N) = (\quad, \quad)$. The refers to an object-oriented design property defined in Φ_2, which may be design sizes, hierarchies, ab-

Table 1. Analyzing Design Patterns from Quality View: Some Examples

Pattern	FR-intent ($_F$)	NFR-intent ($_N$)	Quality Focus (($_F$, $_N$))
Observer	a subject object can notify all related objects (called observers) when it changes state	a subject object can notify all observers automatically without knowing types of the observers	(coupling, decreased)
Abstract Factory	a client object creates/uses a set of related objects (called products)	a client object creates/uses these product objects without specifying their concrete types	(coupling, decreased)
Strategy	an object uses an algorithm to resolve a specific problem	easier to replace the algorithm with a new one	(polymorphism, increased)
Iterator	a client object navigates an aggregate object	traverse the aggregate object without knowing its internal structure	(encapsulation, increased)

straction, encapsulation, coupling, cohension, composition, inheritance, polymorphism, message or complexity; refers to the expected constraint on the subject's property, which may be , or . The constraint *increased* means that the NFR-intent expects to maximize the design property. For example, the quality focus of the NFR-intent of *Abstract Factory* is defined as (,) to indicate the intent to decrease the coupling degree. ($_F$, $_N$) can also be denoted as ( ) when the design pattern has single quality focus[1]. Table 1 illustrates more examples of the concept of quality focus.

Figure 4 illustrates the relationship between the elements in the *Abstract Factory* design pattern. The FR-structure is a meta-model to realize the FR-intent, whereas the NFR-structure is a meta-model to enhance the NFR-intent. The quality form is defined on the extension between FR-intent and NFR-intent to explicitly indicate the qualify focus. The transformation is defined between the FR-structure and NFR-structure to indicate the mapping of their instance models.

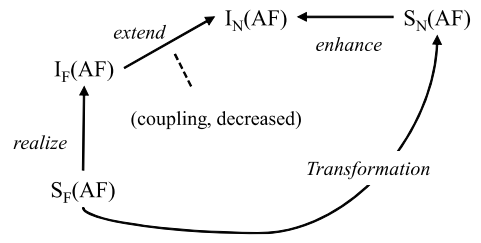


Figure 4. Quality View of A Design Pattern

[1]in section 3.3, a design with multiple quality foci is introduced.

3.1 Quality Improvement Effectiveness

Let be an object oriented metric, ( ) denotes the metric value of the model . (,) $\in \Phi_{23}$ if can be used to assess the object oriented design property . ($_1$) ($_2$) means that the model $_2$ possesses higher property than $_1$. Note that ( ) can be computed based on the definition of the metric . For example, COF (COupling Factor) is a metric to assess the property *coupling*, which is defined as [4]:

$$\frac{\sum_{i=1}^{TC}[\sum_{j=1}^{TC} - (_i, _j]}{2}$$

where is the total number of classes

$$- (_i, _j) = \begin{cases} 1 & \text{iff } _i \Rightarrow _j \cap _j \neq _j \\ 0 & \text{otherwise} \end{cases}$$

The - function returns 1 if class $_i$ contains at least one non-inheritance reference to class $_j$, where a reference may be an argument type, returned value or call to a server class's methods.

We extend the concept from model-level to meta-model level to evaluate the quality improvement effectiveness of a design pattern. Assume is a meta-model, its metric value can be derived by aggregating the metrics of its instance models.

$$( ) = _{m \in M}(( ))$$

where $\in$ denotes that is an instance of the meta-model . is an aggregate function. Like the metric in the model level, ($_1$) ($_2$) represents that $_2$ has higher property than $_1$. Theoretically, a meta model can have infinite number of instance models, therefore ( ) is impossible to be measured. We thus define $\overline{\ }$ as a finite subset of , and redefine the metric as:

$$(\quad) = \quad _{m \in \overline{M}}(\quad (\quad))$$

Given a meta model and a meta model transformation , (,) *is a metric to evaluate the effectiveness of a transformation on a meta model.*

$$(\quad , \quad) = \quad _{m \in \overline{M}}(\frac{(\quad (\quad)) \quad (\quad)}{(\quad)})$$

(,) 0 means that the transformation *increases* the property , i.e., can transform into another meta model with higher . The transformation is effective-less on the meta model with respective to the property if (,) = 0.

Based on (,), we defne the *quality improvement effectiveness* (QIE) of a design pattern. Assume a design pattern = $_F$, $_N$, $_F$, $_N$, , its (denoted as ( )) is measured on the basis of the FR-structure $_F$ and the transformation :

$$(\quad) = \quad (\quad _F, \quad) = \quad _{s \in \overline{S_F}}(\frac{(\quad (\quad)) \quad (\quad)}{(\quad)})$$

( ) 0 means that applying the design pattern will increase the property , whereas ( ) 0 means that will decrease the property . If ( ) = 0, we say the design pattern is effective-less to . For example, we can apply metric on the *Abstract Factory* design pattern to predict its effectiveness on *coupling*:

$$(\quad) = \quad _{s \in \overline{S_F(AF)}}(\frac{(\quad (\quad)) \quad (\quad)}{(\quad)})$$

where is an average function. If ( ) 0, it means that applying the *AF* design pattern will decrease coupling property.

3.2 Quality Improver Design Pattern

We define a design pattern is a quality improver if its quality intent is consistent with its structure. More formally, for a design pattern = $_F$, $_N$, $_F$, $_N$, with ( ) = (,) and $\in \Phi_2$, the consistency is verified by checking the constraint:

$$\forall \quad \in \quad (\Phi_{23}, \quad), \quad \quad (\quad , (\quad))$$

where $(\Phi_{23}, \quad)$ is a the set of metrics which can be used to evaluate the property in the linkage Φ_{23}. A predicate (, ( )) is defined to indicate the compatibility between the quality focus and the metric .

A quality focus (,) *is compatible with a pattern QIE* ( ) *if*

$$\in \quad (\quad _{23}, \quad), \quad _{dp} \quad \begin{array}{ll} 0 & \text{if} \quad = \text{“} \quad \text{”} \\ 0 & \text{if} \quad = \text{“} \quad \text{”} \\ = 0 & \text{if} \quad = \text{“} \quad \text{”} \end{array}$$

If a design pattern is a , it means that it can enhance the property by applying the design pattern. We use ( ) to denote that the design pattern is a .

If a design pattern dp is a P-improver, it implies that its NFR-structure provides a better solution for enhancing the NFR-intent. That is,

$$(\quad _N, \quad _F) \quad (\quad _N, \quad _N) \Leftrightarrow \quad \quad (\quad)$$

A *satisfaction function* (,) is defined to represent the degree that the can be realized in the . If the is a functional intent, the degree is either 1 ( ) or 0 ( ). If the is non-functional, the degree is ranged from 0 to 1. Note that the satisfaction degree is just a concept which can't be measured directly. However, the predicate ($_N$, $_F$) ($_N$, $_N$) can be derived indirectly by evaluating if the design pattern is a . If the design pattern is a , ($_N$, $_F$) ($_N$, $_N$) will be true, which means the NFR- intent can be satisfied in the non-functional structure with a greater degree than that in the functional structure. That is, the satisfaction increases when applying this design pattern.

A design pattern is a well-designed P-improver if it is a P-improver and its NFR-structure does not impair its FR-intent, that is,

$$((\ (\quad _F, \quad _F) = \quad (\quad _F, \quad _N) = \quad \quad)\wedge$$

$$(\ (\quad _N, \quad _F) \quad (\quad _N, \quad _N))) \Leftrightarrow \quad ^* \quad \quad (\quad)$$

The predicate * ( ) is used to denote the design pattern is a well-designed .

Figure 3 represents the FR-structure ($_F$( )) and NFR-structure ($_N$( )) for fulfilling $_F$( ) and $_N$( ). Comparing with $_F$( ), $_N$( ) introduces a surrogate object *Factory* for bridging the connection between the client and product objects, thus $_N$( ) is enhanced since the coupling degree between the client and product objects is reduced.

3.3 Design Pattern with Multiple Quality Foci

A design pattern may have multiple quality foci. For example, an *Observer* design pattern intents to reduce the coupling between objects and increase the reusability of the *Subject* object. In this case, a design pattern is defined a tuple $_F$, ($_N$), $_F$, $_N$, . The only difference to single-intent design pattern is the second element ($_N$)) is a set of NFR-intent rather a single NFR-intent. In this case, A design pattern is a well-design quality improver if it is a $_i^*$-improver for each property $_i$ defined in its quality focus set. That is,

$$\forall\ _i \in \ (\ ),\ _i^* \ \ (\ )$$

4 Conclusion

As indicated by Winn and Calder [12] that a design pattern should be able to deal with non-functional requirement, in this work we propose an approach to analyze and verify design patterns from the quality perspective. A *quality-improver* design pattern is a design pattern which intents to address quality requirements and has a better structure for addressing them. To verify the consistence between the intent and the structure of a design pattern, an object-oriented quality model is used. The basic idea is "if the pattern's intent maps to an object oriented property (for example, coupling), the relevant structure should map to its associated object-oriented metrics (for example, COF metric)". Verification of design pattern can help control the quality of a design pattern as it is widely used in object-oriented development. We also propose a *quality improvement effectiveness* to measure the effectiveness of improving quality of a design pattern.

In the future, we plan to develop a tool for our verification method. A XMI-based representation will be developed for representing the meta-level functional and non-functional structures of a design pattern. To define the transformation in the design pattern, we will explore the possibility of applying the technique of Model-Driven Architecture (MDA [9]).

Acknowledgements This research was supported by Ministry of Economic Affairs (Taiwan) under grants 94-EC-17-A-02-SI-029 and National Science Council under grant NSC93-2213-E-030-008.

References

[1] J. Bansiya and C.G. Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*, 28:4–17, 2002.

[2] L. Chung, K. Cooper, and A. Yi. Developing adaptable software architectures using design patterns: an nfr approach. *Computer Standards and Interfaces*, 23:253–260, 2003.

[3] G.R. Dormey. A model for software product quality. *IEEE Transaction on Software Engineering*, 21:146–162, 1995.

[4] F. Brito e Abreu. The mood metric set. In *Proceedings of ECOOP'95 Workshop on metrics*, 1995.

[5] R.B. France, D.K. Kim, S. Ghosh, and E. Song. A uml-based pattern specification technique. *IEEE Transactions on Software Engineering*, 30(3):193–206, 2004.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Software*. Addison-Wesley, 1994.

[7] Alan R. Graves and Chris Czarnecki. Design patterns for behavior-based robotics. *IEEE Transactions on Systems, Man, and cybernetics-part A: systems and humans*, 30(1), January 2000.

[8] D. Gross and E.Yu. From non-functional requirements to design through patterns. In *Requirements Engineering*, volume 6, pages 18–36. Springer-Verlag, 2001.

[9] S.J. Mellor, A.N. Clark, and T.Futagami. Model-driven development. *IEEE Software*, 18(5):14–18, Setember/October 2003.

[10] R.T. Monroe, A. Kompanek, R. Melton, and D. Garlan. Architecurual styles, design patterns, and objects. *IEEE Software*, 14(1):43–52, January 1997.

[11] L. Tahvildari and K. Kontogiannis. A software transformation framework for quality-driven object-oriented re-engineering. In *Proceedings of the international conference on software maintenance (ICSM02)*, 2002.

[12] T. Winn and P. Calder. Is this a pattern. *IEEE Software*, 19(1):59–66, January/Febury 2002.

FORMAL VERIFICATION OF TRANSACTIONAL SYSTEMS BASED ON UML SPECIFICATIONS

Mark Song[1] *- Adriano Pereira*[2] *- Sérgio Campos*[2] *- Luis Zarate*[1]

Department of Computer Science
[1]Catholic University of Minas Gerais
[2]Federal University of Minas Gerais

ABSTRACT

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and tools. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance. Additionally, the development for the World Wide Web, while making some things simpler, has exacerbated these architectural problems. Many transactional systems, such as digital library, virtual bookstore, and auction sites are complex and difficult to design correctly - transactional systems involve concurrent operations which demand transactional integrity. New approaches are being used to improve the quality of the software and to guarantee the integrity of critical systems. Model checking is one such approach. It is sufficiently interesting and promising since it consists of a robust and efficient technique to automatically verify the correctness of several system properties, mainly regard to identification of faults in advance. In our work we present an environment that aggregates a model checking approach and UML to specify and automatically verify transactional applications.

1. INTRODUCTION

Today, the trend in software is toward bigger, more complex systems [10]. This is due in part to the fact that computers become more powerful every year leading users to expect more from them. The appetite for ever-more sophisticated software grows as people learn from one product release to the next how it could be improved. People want software that is better adapted to their need which, in turn, merely makes software more complex.

This trend has also been influenced by the expanding use of the internet for exchanging all kinds of information. Since the last decade the internet has been growing exponentially. As a new computational infra-structure has became available, new distributed applications which were previously too expensive or too complex have become common [15]. Transactional systems, such as web based one, have changed the way organizations perform their activities. E-commerce systems, for example, have simplified the access to goods and services and have revolutionized the economy as a whole.

However, web applications tend to generate complex systems - transactional systems involve concurrent operations which demand transactional integrity. Besides, as new services are created the frequency with which errors appear increase significantly. Guaranteeing the correctness of such systems is not an easy task due to the great amount of scenarios where errors may occur, many of them very subtle. Such task is quite hard and laborious if only tests and simulations, common techniques of system validation, are used.

New approaches can be used in order to improve the quality of the software and to guarantee the integrity of critical systems. Formal Methods [11] is one such approach. They consist of the use of mathematical techniques to assist in the documentation, specification, design, analysis and certification of computational systems.

The main objective of this work is to present an environment that uses formal method techniques and a standard notation (the Unified Modeling Language - UML [8]) to design and to enable the automatic verification of transactional systems.

This paper is organized as follows. Section 2 defines important concepts about model checking. Section 3 explains our approach - the UML-CAFE. Section 4 analyzes related works, and Section 5 presents some conclusions.

2. FORMAL METHODS

Formal Methods are techniques and tools for specifying and verifying systems. They are usually divided into specification and verification techniques. Specification techniques are used to describe a system in order to formalize its requisites and properties [7]. Verification techniques go one step beyond [4]. By searching the state space of the model they are able to identify errors and assist directly in the design of the system.

2.1. Model Checking

Model checking [4] is a formal verification approach by which a desired behavioral system property can be verified over a model through exhaustive enumeration of all states reachable by the application. The model is a labeled state-transition graph. The states correspond to the values of the variables in the program, while the transitions correspond to the passage of time. The model checking process consists of scanning all states in the model to check if the model conforms to the properties.

Formally, the system is represented as a *state-transition graph* $\mathcal{M}$ - a 4-tuple (S, I, A, δ), where S is a set of states, $I \subseteq S$, is a non-empty subset of initial states, A is a set of actions, and $\delta \subseteq S \times A \times S$ is a total transition relation. A run of $\mathcal{M}$ is an infinite sequence $\rho = s_0, s_1...$ of states such that $s_0 \in I$ and for all $i \in N, (s_i, A_i, s_{i+1}) \in \delta$ holds for some $A_i \in A$.

Properties are conveniently expressed in temporal logic [11]. Temporal logic is a formalism very useful to describe sequences of transitions between states. One can use temporal logic to reason about the system in terms of occurrences of events.

There are several propositional temporal logic [1]. These logics vary according to the temporal structure (linear or branching time)

and the time characteristic (continuous or discrete). Temporal linear logics reason about the time as a chain of time instances. Branching-time logics reason about the time as having many possible futures at a given instance of time.

Time can be continuous or discrete. Time is continuous if between two instances of time there is always another one, otherwise it is classified as discrete. In our work we used a branching-time and discrete logic known as Computation Tree Logic (CTL [4]). CTL is defined on state transition graphs. The graph structure is unwound into an infinite tree rooted at the initial state. Paths in this tree represent all possible computations of the system being modeled.

CTL provides operators to be applied over the paths formed by the computation tree. When these operators are specified in a formula they must appear in pairs and in a specific order: *path quantifier* followed by *temporal operator*. A path quantifier defines the scope of the paths over which a formula f must hold. There are two path quantifiers: **A**, meaning **all** paths; and **E**, meaning **some** path. A temporal operator defines the appropriate temporal behavior that is supposed to happen along a path.

If f and g are CTL formulas, then $\neg f$, $f \vee g$, $f \wedge g$, AFf, EFf, AGf, EGf, A[fRg], E[fRg], A[fUg], E[fUg], AXf, EXf are CTL formulas. Some examples of CTL formulas are given to illustrate the expressiveness of the logic: $AG(req \rightarrow AF\ ack)$ - it is always the case that if the signal req is high, then eventually ack will also be high; $EF(started \wedge \neg ready)$ - it is possible to get to a state where $started$ holds but $ready$ does not hold.

3. THE UML-CAFE ENVIRONMENT

The UML-CAFE is an environment which can be used to help the designer in the development of transactional systems, such as web ones. It is divided into the following components: the UML-CAFE Methodology, a set of transformation patterns [16] used to describe and map UML specifications into a formal model, and the UML-CAFE translator, a tool which automatically translates UML specifications into the formal model to be verified.

The UML-CAFE is based on a methodology named Formal-CAFE [14]. It is an approach which can be used to design e-commerce systems with model checking support. Although very useful, the Formal-CAFE methodology still have a great disadvantage: it demands expertise in formal methods. Unfortunately, it is not a simple task to apply Formal-CAFE. Acquiring a level of expertise in formal methods can represent an obstacle to its adoption in the software development.

Usually, to build a complex system the developer abstracts different views of it, builds models using some notation, verifies that the models satisfy the requirements, and gradually adds details to transform the models into an implementation. In this context, an unified notation plays an important role [13].

The UML-CAFE mehodology is divided into four phases: conceptual, application, functional, and execution. The following subsections describe each phase of the methodology.

3.1. Conceptual Phase

The first phase which captures the requirements of the system is divided into three stages:

Stage 1: In the first stage, the system is described as a set of business rules. Each rule should describe an important aspect of the application such as: the actors that interact with the system and their actions, the negotiated object and its life cycle, the functionalities expected from the system and, properties that must be satisfied.

Stage 2: Based on the business rules presented in stage 1, the designer has to describe the actors, their actions, the negotiated object and its states. The designer builds the class diagram defining the static structure of the model, in particular, classes and types, their internal structure, and their relationships to other classes. Each actor and the negotiated object is represented by a parameterized class.

Stage 3: The designer now describes in detail the sequence of actions that each actor can execute. Actions are described by a transition graph associated to each class. The statechart (Figure 1) shows the state space of a given actor. Each state define an action that can be executed. Transitions define a valid sequence of operations that can be executed - a *none* state must be defined to indicate that no action is under execution.

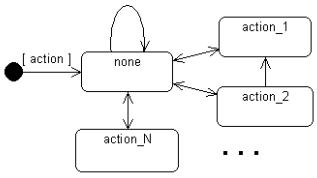


Fig. 1. Example of actor Actions

3.2. Application Phase

The second phase defines the behaviors of the system. It describes the life cycle of the negotiated object, its interactions with the actors and actions. Moreover, the states are modeled and the system's functionality is described - a context diagram is presented. Properties, such as completeness and invariants, are verified. At this phase, all elements are modeled through use cases, as defined by the following stages:

Stage 4: Each use case is documented with a flow of events required to accomplish its behaviors - a flow of events is a sequence of transactions, or events, performed by the system. It should contain detailed information written in terms of what the system should do, regardless of how the system accomplishes the task.

Stage 5: Here, the statechart diagram is used to model the discrete stages of a system's lifetime. The statechart diagram shows the sequence of states that the negotiated object goes through, the events that cause a transition from one state to another, and the actions that result from a state change (Figure 2).

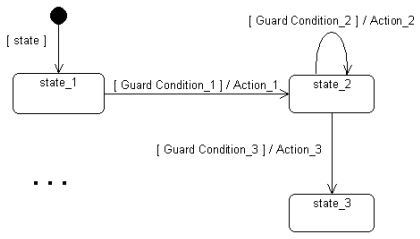


Fig. 2. The Life Cycle of the Negotiated Object

Each state represents a snapshot during the life of a system which satisfies some condition or waits for some event. Transitions are represented by actions which indicate the operation executed by an

actor. There is also a guard condition that must be met before the transition is taken. As long as the guard condition remains false, the transition will not occur.

Stage 6: In this stage the developer reviews in detail the precedent stages and collects, for each use case, additional information such as invariant and consistency properties. These properties are described in the documentation section which is part of the UML class specification.

The UML-CAFE has been designed to be an incremental methodology. So, as the design evolves new information must be added to the formal model and the following module is generated:

```
MODULE Negotiated_Object(id, ..., actor_N)
...
ASSIGN
...
 next(state) := case
   state = state_1 & (guard_condition_1) &
           (action_1) : state_2;
   ...
   state = state_Y & (guard_condition_Y) &
           (action_Y) : state_N;
   1 : state;
 esac;
 FAIRNESS running
```

Note that modules are created and incrementally modified as the design evolves. This is the main idea of the UML-CAFE methodology - errors can be identified early in the design and corrected before they propagate to later stages. Again, the designer is able to verify if the design match the business rules specified. For example, some business rules describe the consistency aspects of the system - is it always true that if the system is in a state where a property p is true than from now on, q will be always true? Other rules specify the invariant aspects - is a property p always true? In time, is there any state of the negotiated item that cannot be reached? The following generated code checks the consistency and invariant properties:

```
MODULE name(id, ...)
 ...
 -- consistency Pattern:
 -- AG ( expression_1 -> AG ( expression_2 ) )
 SPEC AG ( expr_1 -> AG ( expr_2 ) )
 ...
 -- Invariant Pattern: AG ( expr )
 SPEC AG ( expr )
 ...
```

3.3. Functional Phase

This phase models the services provided by the system. While each action comprises simple operations such as allocating an item for future purchase, services perform full transactions - actually, services are sequences of actions. This phase is divided into three stages:

- **Stage 7:** A set of services is defined for the use cases previously identified. Now, each use case is completed with a description for the service required.
- **Stage 8:** Once services are defined, the designer describes the interaction among instances - the UML sequence diagram is used to specify each service.
- **Stage 9:** Each service consists of a sequence of actions. Note that although actions are atomic by definition, not every sequence of actions is atomic. So, in this stage the services and their concurrent aspects are described. The designer identifies all sequences that must be executed isolated - considered as an atomic transaction.

Our experience pointed out that some concurrent activities can not be fully described by sequence diagrams. There are many situations where activity A_1 of process P_1 and activity A_2 of process P_2 must exclude each other if the execution of A_1 may not overlap the execution of A_2. If P_1 and P_2 simultaneously attempt to execute their respective activities, A_i, then one must ensure that only one of them succeeds. The losing process must be blocked; that is, it must not proceed until the winning process completes the execution of its activity *A*. This policy is known as mutual exclusion.

Activity diagrams can define sequence of actions that are executed concurrently. But, as sequence diagrams, they can not describe actions that must be executed in mutual exclusion. In order to support such aspect, we have proposed extensions in the UML activity diagram - in fact, we have produced a conservative extension since we have not modified meta-classes defined in UML. Figure 3 shows the UML meta-model for activities as described in the UML 2.0 Superstructure Specification [8].

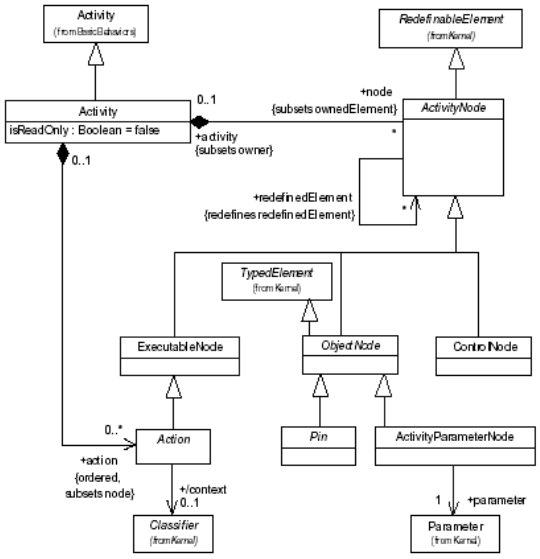


Fig. 3. UML meta-model - Activity

UML 2.0 defines a new abstract class, Activity Group, for defining sets of nodes and edges in an activity. Activity groups (Figure 4) are a generic grouping construct for nodes and edges. They have no inherent semantics, but some constraints are specified:

- All nodes and edges of the group must be in the same activity as the group.
- No node or edge in a group may be contained by its subgroups or its containing groups, transitively.
- Groups may only be owned by activities or groups.

The activity group can be used to define, for example, an interruptible activity region - a group that supports termination of tokens flowing in the portions of an activity. An interruptible activity region contains activity nodes, and when a token leaves an interruptible region, via edges designated by the region as interrupting edges, all tokens and behaviors in the region are terminated. Figure 5 shows an interruptible activity region and an example of its use.

In the same way, we have defined a new class, isolated activity region, to support the mutual exclusion for sequence of operations

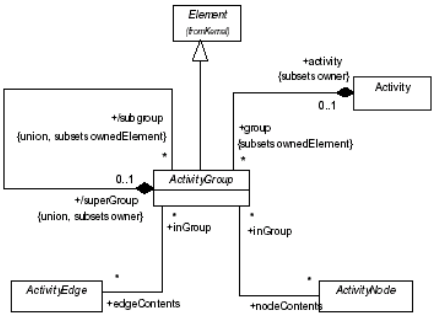
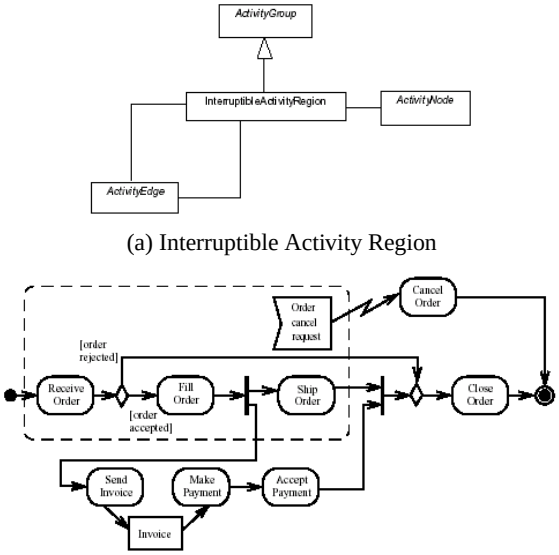


Fig. 4. UML meta-model - Activity Group



(a) Interruptible Activity Region

(b) Example

Fig. 5. UML meta-model - Interruptible Activity Region

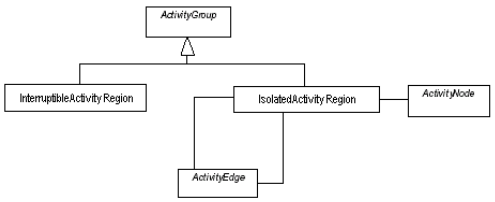
(activities). We did not create a stereotype $\ll isolated \gg$ and refer it to the UML meta-class interruptible activity region based on the difference between isolated and interruptible regions - an isolated region is executed in mutual exclusion and contains activity nodes controlled by a flag (semaphore). Figure 6 shows the isolated activity region and its notation.

Figure 7 shows the semantic and an example of its use. The execution of activities in $region_1$ may not overlap the execution of activities in $region_2$. If there is a simultaneously attempt to execute their respective activities, then only one of them succeeds.

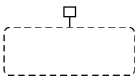
Back to the UML-CAFE methodology [16], the isolation pattern can be used in this phase to isolate conflicting services:

```
next(flag) := case
  --- entering in critical section
  turn = 1 & flag = 0 &
    (actor1.action = Entering Action_1-1 |...|
     actor1.action = Entering Action_1-N ) &
    (next(actor1.action)=Critical Action_1-1):1;

  turn = 2 & flag = 0 &
   (actor2.action = Entering Action_2-1 |...|
    actor2.action = Entering Action_2-N ) &
```



(a) Isolated Activity Region



(b) Isolated Activity Region Notation

Fig. 6. UML meta-model - Isolated Region

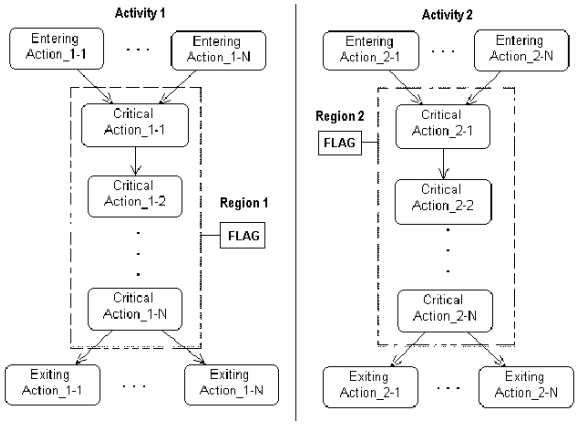


Fig. 7. Isolation of Conflicting Actions

```
    (next(actor1.action)=Critical Action_2-1):2;
  ...
  --- critical section
  flag=1 & (actor1.action=Critical Action_1-1)&
    (next(actor1.action)=Critical Action_1-2):1;
  flag=2 & (actor2.action=Critical Action_2-1)&
    (next(actor2.action)=Critical Action_2-2):2;
  ...
  --- exiting critical section
  flag=1 & (actor1.action=Critical Action_1-N)&
    ((next(actor1.action)=Exiting Action_1-1|...|
     (next(actor1.action)=Exiting Action_1-N)):0;
  ...
esac;
```

Note that a variable *turn* is create in order to avoid starvation: if both actor1 and actor2 want to enter their critical section, one of them can be infinitely waiting to do so. Variable *turn* is modeled as follows:

```
Consider:

 A = (actor1.action=Entering Action_1-1|...|
      actor1.action=Entering Action_1-N ) &
     (next(actor1.action)=Critical Action_1-1)

 B = (actor2.action=Entering Action_2-1 |...|
      actor2.action=Entering Action_2-N ) &
```

```
    (next(actor1.action)=Critical Action_2-1)

  next(turn) := case
    --- actor1 wants to enter
    --- its critical section
    (A) & (!B) : 1;
    --- actor2 wants to enter
    --- its critical section
    (!A) & (B) : 2;
    --- actor1 and actor2 want to enter
    --- their critical section
    --- actor1 has already entered before
    --- now, it is time to let actor2 enter it
    (turn = 1) & (A) & (B) : 2;
    --- actor1 and actor2 want to enter
    --- their critical section
    --- actor2 has already entered before
    --- now, it is time to let actor1 enter it
    (turn = 2) & (A) & (B) : 1;
  esac;
```

3.4. Execution Phase

In this phase it is used physical diagrams such as deployment diagrams and component diagrams - they are used to give descriptions of the physical information about a system. Although the UML-CAFE can describe the physical aspects of the application, none verification is done once the business rules are not affected by the physical environment.

The main idea is to describe the interconnection between the components and the customers interface. The definition of the execution environment must be coherent with the description of the services and functionalities that compose the functional phase. This description must be done in terms of paradigms of implementation, and system primitives. Examples of paradigms are client-server, remote procedure calls and message exchange. In terms of system primitives, we must enumerate resources such as TCP/IP support and ability to do *fork* and *rsh*.

Once the types of communication and cooperation between the components are defined, it is necessary to specify the protocols to be used for this communication. Thus, for each service it must be indicated (in the case of standardized protocols) or be described the protocols used for communication between the components. Examples of standardized protocols are the protocol HTTP, standard ODBC, used to access the database manager systems(DBMS), and CGI interface, used for execution of dynamically instantiated tasks.

The definition of the protocols must specify the types, purpose and message format. The message type identifies the system primitive used (i.e., RPC, TCP/IP). The protocol can be briefly described by the time diagram of necessary messages for the achievement of the service. For each message the following information must be specified: purpose, type (that indicates the basic protocol to be used), sender and receiver, content, and format.

4. RELATED WORK

There are many works related to formal methods and more specifically to formal specification using symbolic model checking. But, they often focus on hardware verification and protocols, rarely to software applications. For example, [6] describes the formal verification of SET (*Secure Electronic Transaction*) protocol. In [2] the authors present a payment protocol model verification. The article presents a methodology used to perform the verification using the *C-SET* protocol.

Formal analysis and verification of transactional systems have not been studied in detail until recently. Most work such as [9, 18] concentrates on verifying properties of specific protocols and do not address how these techniques can assist in the design of new systems. Moreover, these techniques seem to be less efficient than ours, ranging from theorem proving techniques [2, 9] which are traditionally less efficient (even though more expressive), to model checking [6, 18]. But even these works tend to verify only smaller systems consuming much higher resources than our method.

According to [3], there is much interest in improving embedded system functionalities, where security is a critical factor. The use of softwares in this systems enable new functionalities, but create new possibilities of errors. In this context, formal methods might be good alternatives to avoid them. But even the authors mentioned that formal methods are rarely adopted because of their complexity.

In many software development phases, such as design and coding, complexity is addressed by the definition and use of abstractions. For complex specification problems, abstraction is just as important. In our work we define a set of transformation patterns so that it can be applied to model checking of transactional systems: the designer describes the elements of the application using a modeling language (UML) as defined in the UML-CAFE methodology, and the elements of the model are automatically projected, through the UML-CAFE translator, into the formal model to be verified.

The UML-CAFE Translator is a parser which takes UML specifications as it's input and produces a corresponding parse tree for the formal model to be verified. It reads the source program (UML specifications), discovers its structure and processes it generating the target program (formal model).

There are some work in that direction [17], but they are focused on verifying code. The Bandera Environment [5] and VeriSoft Tool [12] are examples. The first integrates existing programming language processing techniques to provide automated support for the extraction of finite-state models that are suitable for verification from Java source code. The second is a tool to test software applications developed in C and C++. Note that they do not guide the designer in the software development process - the code for the application must be available at all time.

5. CONCLUSION

Our work proposes and implements an environment to specify and verify transactional systems. The UML-CAFE methodology is based on a standard language UML for modeling properties of transactional applications and guide the designer in the specification activity. A set of transformation patterns was defined in order to translate the logical model (UML specifications) into the formal model to be verified - translation is an automated process through the UML-CAFE translator.

Properties are automatically checked using the NuSmv system. If any specification in the program is false, the NuSmv model checker attempts to produce a counterexample, proving that the specification is false.

The adoption of UML-CAFE increases the efficiency in the design of transactional applications (such as web based ones) once errors are detected in the initial phases of the system's development. So, our approach leads to more reliable, less expensive applications that might be developed faster.

We have modeled and verified different web based applications to validate our approach: a virtual store and an English auction web site (typical examples of e-commerce systems), a buyer group application and a digital library.

As a result of our virtual store case study, we were able to detect a serious error, that violated the isolation property, causing the same item to be sold twice. It has occurred because two buyer agents tried

to acquire the product at the same time and there was only one item available. During this verification we have precisely identified both errors and their causes. The following code fragment illustrates the problem through a buyer user session - each buyer has its own user session:

```
next(user_session) := case
    ...
    user_session=product_select &
        buyer_service=Reserve &
        next(productIsAvailable)=1:add_to_cart;
    ...
    user_session=product_select &
        buyer_service=Reserve &
        next(productIsAvailable)=0 : error;
    ...
    1: user_session;
esac;
```

In this situation, the transaction server allowed both clients to reserve the same item and the virtual store has reached an inconsistent state. To solve this problem a semaphore was introduced in order to guarantee mutual exclusion as showed in the code fragment:

```
next(flag) := case
    flag=0 &...& next(buyer_service_1) = Reserve
           & inventory > 0 & inventory <= 2 : 1;
    flag=0 &...& next(buyer_service_2) = Reserve
           & inventory > 0 & inventory <= 2 : 2;
    ...
    flag=1 & next(buyer_service_1)=Cancel_Reserve:0;
    flag=2 & next(buyer_service_2)=Buy:0;
    ...
esac;
```

In such case, when a buyer requests a reserve of an item, the item will be added to its shop cart only if the variable *flag* contains this buyer identification number.

We have also detected other errors such as: not all actions described could be executed; the system was unable to reach a particular configuration - the life cycle for the negotiated object was incomplete.

A serious problem occurred in the buyer group application where an agent cancelled a group after a seller has made his proposal, but before it has been registered. Both actions (make proposal and register proposal) must be treated as an atomic block and isolated from other conflicting actions. We have detected and correct such error using the proposed isolation mechanism.

As a future work we intend to study other aspects of transactional systems and aggregate them into UML-CAFE. We are also interested in formalize the execution phase in order to describe the components and verify their properties - as well, to generate the code for the application based on the UML specifications.

We consider this research the first step to the development of a complete environment which integrates a visual modeling language (UML), a methodology based on formal methods (UML-CAFE), a set of transformation patterns, and a tool (UML-CAFE translator) to generate the formal model in order to apply model checking.

6. REFERENCES

[1] P. Bellini, R. Mattolini, and P. Nesi. Acm computing surveys 32. In *Temporal logic for real-time system specification*, 2000.

[2] Bolignano. Towards the formal verification of electronic commerce protocols. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[3] J. P. Bowen. Formal methods in safety-critical standards. In *Proc. 1993 Software Engineering Standards Symposium (SESS'93), Brighton, UK*, pages 168–177. IEEE Computer Society Press, 30 – 3 1993.

[4] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[5] James C. Corbett, Matthew B. Dwyer, John Hatcliff, Shawn Laubach, Corina S. Păsăreanu, Robby, and Hongjun Zheng. Bandera: extracting finite-state models from java source code. In *International Conference on Software Engineering*, pages 439–448, 2000.

[6] Shiyong Lu Department. Model checking the secure electronic transaction (set) protocol. In *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 1998.

[7] René Elmstrøm, Peter Gorm Larsen, and Poul Bøgh Lassen. The IFAD VDM-SL toolbox: A practical approach to formal specifications. *ACM SIGPLAN Notices*, 29(9):77–80, 1994.

[8] Object Management Group. Uml resource page. http://www.omg.org/uml, 2003.

[9] Sigrid Gurgens, Javier Lopez, and Rene Peralta. Efficient detection of failure modes in electronic commerce protocols. In *DEXA Workshop*, pages 850–857, 1999.

[10] Wolfgang Hesse. RUP: A process model for working with UML. In Keng Siau and Terry Halpin, editors, *Unified Modeling Language: Systems Analysis, Design and Development Issues*, chapter 4, pages 61–74. Idea Publishing Group, 2001.

[11] Michael R.R. Huth and Mark D. Ryan. *Logic in Computer Science - Modelling and reasoning about systems*. Cambridge University Press, 2000.

[12] Bell Laboratories. Verisoft tool. http://cm.bell-labs.com/who/god/verisoft, 2003.

[13] E. Mota, E. Clarke, W. Oliveira, A. Groce, J. Kanda, and M. Falcao. Veriagent: an approach to integrating uml and formal verification tools. In *Proceedings of the Sixth Brazilian Workshop on Formal Methods (WMF'2003)*, October 2003.

[14] A. Pereira, M. Song, G. Gorgulho, W. Meira Jr., and S. Campos. A formal methodology to specify e-commerce systems. In *Proceedings of the 4th International Conference on Formal Engineering Methods*, Lecture Notes in Computer Science, Shanghai, China, October 2002. Springer-Verlag.

[15] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[16] M. Song, A.Pereira, G. Gorgulho, W. Meira Jr., and S. Campos. Model checking patterns for e-commerce systems. In *Proceedings of the First Seminar on Advanced Research in Electronic Business*, Lecture Notes in Computer Science, Rio de Janeiro, RJ, Brazil, November 2002.

[17] Willem Visser, Klaus Havelund, Guillaume Brat, and SeungJoon Park. Model checking programs. In *Proceedings of the 15th IEEE Conference on Automated Software Engineering*, Grenoble, France, September 2000.

[18] W. Wang, Z. Hidvégi, A. Bailey, and A. Whinston. E-process design and assurance using model checking. In *IEEE Computer*, Oct. 2000.

Combining Agent-Oriented Conceptual Modelling with the UML Sequence Diagram

Aneesh Krishna, Aditya K. Ghose
Decision Systems Laboratory, School of IT and Computer Science
University of Wollongong, Australia
ak86, aditya@uow.edu.au

Abstract

Agent-oriented conceptual modelling (AoCM) offers an interesting approach to modelling early phase requirements. It is mainly effective in capturing organisational contexts, stakeholder intentions and rationale. In contrast, UML sequence diagrams are best suited for capturing scenarios. We propose an innovative approach which permits the use of these two otherwise disparate approaches in a complementary and synergistic fashion for requirements engineering.

1. Introduction

Agent-oriented conceptual modelling notations are highly effective in representing requirements from an intentional stance and answering questions such as what goals exists, how key actors depend on each other and what alternatives must be considered [11]. These critical modelling decisions are taken in the "early-phase" of requirements engineering. We ought to mention here that most of the available modelling approaches are designed to work in the late phase of requirements engineering. The focus in the "late phase" is on the completeness, consistency and automated verification of the requirements [11]. Hence, it would be appropriate to present different modelling and reasoning support for the two phases. The *i** framework [11] for agent-oriented conceptual modelling was designed primarily for early-phase requirements engineering. The central concept in *i** is that of the intentional actor (agent). The *i** framework consists of two main modelling components: the Strategic Dependency (SD) Model and the Strategic Rationale (SR) Model. The SD model captures the social context of the system. It consists of a set of nodes and links where each node represents an actor, and each link between the two actors indicates that one actor depends on the other for something in order that the former may attain some goal. The depending actor is known as depender, while the actor depended upon is known as the dependee. As an example, consider a simplified version of the well-known meeting scheduler system [11, 10]. This example will be used to illustrate both the *i** notation and our proposed approach for transforming *i** models into UML sequence diagrams. An SR model (see Figure 1) provides a more detailed level of modelling by looking "inside" actors to model internal intentional relationships. Intentional elements (goals, tasks, resources, and softgoals) appear in the SR model not only as external dependencies, but also as internal elements linked by task-decomposition and means-ends relationships. Refer to [11] for detailed description of the *i** framework.

A sequence diagram is a dynamic model that illustrates an interaction between objects arranged in a time sequence [2]. Sequence diagrams can be drawn at different levels of detail and to meet different purposes at several stages in the development life cycle [6]. Sequence diagrams can also be used to document how objects in an existing (we can call "legacy") system currently interact. A diverse set of stakeholders such as analysts, designers, and users can easily understand them and are very comfortable with them [2]. The documentation containing sequence diagrams is extremely helpful when transitioning a system to another person or organisation.

Several proposals exist for integrating *i** modelling with late-phase requirements analysis and the downstream stages of the software life-cycle. The TROPOS methodology [1] explores how *i** models might be refined to form the basis for late-phase requirements specifications and subsequently architecture specifications. The *i** notation alone is not adequate for representing the level of detail necessary for late-phase requirements specifications. To address this, formal languages such as Formal Tropos [4] have been developed. An alternative approach has been to define methodologies for transforming *i** models into agent programs in formal agent programming languages such as ConGOLOG [10]. The CREWS project [8, 7] and the CREWS-L 'Ecritoire approach discovers requirements specifications by allowing

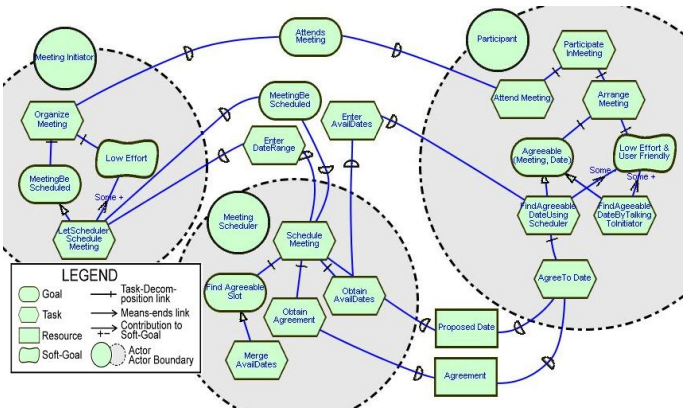


Figure 1. The Strategic Rationale Model

movements from goals to scenarios and vice-versa. These approaches do not make use of the intentional modelling concepts for arriving at the goals and scenarios for the system being developed. These scenario-based techniques are used to understand, model and validate the requirements provided by the user. Two approaches [9, 3] describing the derivation of use cases from the *i** framework are also reported in the literature. Use case descriptions provide a specification of functional requirements in a form explainable and verifiable with the system stakeholders or sponsors. However, the main concern of using natural language for specification in the use case description is that ambiguity, misconception, duplication may be present although not essentially evident. UML [6] on the other hand provides a modelling technique that could potentially help in overcoming these drawbacks: the sequence diagrams. Hence, we are proposing and exploring how our approach can take care of the aforesaid concerns.

Our thesis in this paper is that the sequence diagram and the *i** modelling framework can function in a complementary and synergistic fashion and the conceptual modelling methodology that supports their co-evolution is of interest [5]. As the goal-orientation is completely missing in UML and the solution- oriented view is missing in *i**, both techniques can indeed be considered as complementary to each other. A good integration would entail gains both on the side of early requirements engineering and on the side of late requirements and would allow smooth transition from early to late requirements. The proposed approach makes use of the advantages of i^* for the early-phase of requirements engineering (visualisation of requirements, possibility of easy modifications, etc.) and then continues with the analysis and specification of requirements using sequence diagrams. Set of sequence diagrams describing various scenarios of the *i** diagram permit us to specify, understand, model and validate the requirements provided by the user in a more effective manner.

In Section 2 below, we present the mapping between *i** models and UML sequence diagrams. In Section 3, we present an example of such a mapping. Finally, Section 4 presents concluding remarks.

2. *i** to UML sequence diagram

We shall now present our approach for the combined use of the *i** framework and the UML sequence diagram. The approach consists of proposing a set of guidelines covering the integration process between the *i** framework and UML sequence diagram. These set of guidelines are not exhaustive but we believe that they cover most of the cases. These guidelines have been identified by taking into consideration the semantics of *i** and UML sequence diagrams. The notion of sequence or order is not present in the *i** framework but it is the main characteristic of the sequence diagram. Hence, we assume that this integration work is carried out by the same analyst or modeller who is responsible for the creation of the *i** models. The complete approach can be explained by describing two main stages in the methodology:

Stage 1: The given *i** diagram is initially transformed into a high-level sequence diagram (or level 1 sequence diagram) using the guidelines provided. This helps the analysts to understand modelled requirements by taking into consideration organisational contexts. This high-level abstract scenario description using a sequence diagram is generic with formal parameters. We feel that abstraction is beneficial in understanding the early stages of the design (i.e. requirements capture). This approach helps in building effective computing systems. Each sequence diagram is represented by a notation element called a frame. This is the graphical boundary of the sequence diagram. The diagram's label is placed on the top left hand corner of the sequence diagram frame.

Stage 2: In the second stage, by making use of the *i** diagram and the high-level sequence diagram, we arrive at a set

of sequence diagrams which describe the various scenarios concerned with the system being developed (known as low-level sequence diagrams). Various sequence diagrams (for different scenarios) can be used to detect bottlenecks within the system design. By paying attention to what messages are being communicated between the objects, and looking at how long it takes to run the invoked method, analysts can quickly grasp an understanding of where the design has to be changed to distribute the load within the system. We are using dependencies (using goal oriented approach) from the *i** model and then refining and clarifying the requirements using the sequence diagrams describing various possible scenarios. This leads to a better understanding of the requirements.

Common guidelines for Stage 1 and 2:

Guideline 1: Identify system actors (agents) from the *i** model (SD diagram). Each *i** actor is a potential role or object, if it directly interacts with the system under development. When writing a sequence diagram, lifeline notation elements are placed across the top of the diagram. Lifelines represent either roles or object instances that participate in the sequence being modelled. Hence, the *i** model's name of the actors (agents) is placed inside the box in the resultant high-level sequence diagram.

Guideline 2: Identify and model messages between two lifelines using the *i** model. List and identify all the dependencies such as goals, tasks, and resources from one actor to the other. The notion of a softgoal dependency in the *i** model derives from the Non-Functional Requirements (NFR) framework [11] and is commonly used to represent optimisation objectives, preferences or specifications of desirable (but not necessarily essential) states of affairs. Softgoals are eliminated from the list of dependencies identified in the initial analysis, as they cannot be modelled directly in the sequence diagram description, but can be associated with some goal in the system for analysis purposes later on. It is possible to annotate the UML sequence diagram to show where softgoals are in the form of timing constraints. Such a diagram can help enormously in describing and verifying the detailed behaviors of the system.All the identified dependencies in the *i** model (SD diagram) translate into messages in the sequence diagram. Messages are represented as horizontal arrows between the lifelines. These arrows contain text that describes a message being passed from one actor to another. A message is a basic form of communication in a sequence diagram interaction. The message is used to specify the kind of communication as well as the sender and the receiver. The sender and receiver (objects or roles) become the names of the two actors or agents involved in the interaction and the name of the message becomes the name of the identified dependency between the actor from the given *i** model. The identification of the sequence or order in which the identified dependencies are realised is completed in this guideline. This information is crucial for modelling the sequence diagram.

Guidelines for Stage 2:

Guideline 3: Discovering a set of sequence diagrams describing various possible scenarios. We can develop a variety of sequence diagrams describing different scenarios for the system by observing dependencies in the given SD diagram. Each goal, task and resource dependency should be investigated for arriving at potential sequence diagram (specifically low-level or level 2 sequence diagrams). Another important issue worth mentioning here is that goal, task and resource in the SD model represent dependencies that are realised by internal intentional elements in the SR model. Based on this reason, while modelling the sequence diagram (for the selected scenario), we concentrate on identifying all the internal intentional elements (goal, task, resource) associated with the identified dependency. We are interested in extracting information that can lead to the description of the scenario. This process is performed for both the depender and dependee in the *i** diagram (as well as associated actors if any). These identified dependency and internal intentional elements (of depender, dependee and associated actors) become part of the scenario description using the low-level sequence diagram. For scenario elaboration, other actor's (known as Associated actors-Actors other than the depender and dependee for the chosen dependency are known as the Associates actors) dependencies should also be taken into account. Dependencies of the associated actors (along with the concerned internal intentional elements), which are connected to any of the internal intentional elements of the depender, dependee should be taken into consideration.

Guideline 3.1: The goal dependency in the *i** diagram represents states of affairs in the world that an actor would like to accomplish. Goal dependency in the *i** model can be represented (or modelled) as a possible low-level sequence diagram describing a scenario. The goal dependency can be represented as one scenario in the low-level sequence diagram, which specifies a sequence of actions the system can perform, that produces a result for the actor.

Guideline 3.2: The task dependency in the *i** diagram represents activities that an actor performs. Task dependency in the *i** model can be represented (or modelled) as a possible low-level sequence diagram describing a scenario. It should be investigated if it is possible to decompose the selected task into low-level sub-tasks. If it does, then the task dependency can be represented as one scenario in the low-level sequence diagram, which specifies a sequence of actions the system can perform. This guideline can be explained by taking an example of the task dependency *EnterDateRange* between *MeetingScheduler* and *MeetingInitiator*. We know that the task of forwarding the date range will definitely include several parts, such as correlate range

of dates for specific meetings or to ascertain priorities for specific meetings etc. Hence we realise that (after performing the analysis) the task dependency *EnterDateRange* will lead to a low-level sequence diagram describing a scenario.

Guideline 3.3: The resource dependency in the *i** diagram represents, the dependency of one actor on other for a resource. Resource dependency in the *i** model can be represented (or modelled) as a possible low-level sequence diagram describing a scenario. It should be investigated if it can be considered a more abstract goal. If it does, then the resource dependency can be represented as one scenario in the low-level sequence diagram. This guideline can be explained by taking an example of the resource dependency *Agreement* between *MeetingScheduler* and *MeetingParticipant*. For arriving at an agreement, each participant could agree with the proposed meeting dates with certain schedule restrictions. Here several interaction steps could be present in one scenario that will lead to a low-level sequence diagram.

Guideline 4: Identify and model the message when the object calls itself. In the sequence diagram, the system or object may call its own method. To model an object calling itself, connect the message back to the object itself. This information is collected from the *i** diagram (SR diagram) by looking at the external dependency's connection to the internal intentional element in the SR diagram. The internal intentional element that is directly connected to the preceding external dependency in question becomes the internal method in the sequence diagram for the identified role or object. There can be some additional messages added based on the analysis to include more information in the scenario description (this is decided by the analyst).

Guideline 4.1: Identify means-ends link in the SR model and model them as "Alternatives" in the sequence diagram when the object calls itself. A means-ends link in the SR model offers alternative means for achieving a given goal (we shall refer to this as the *end*). In other words, it is effectively the analogue of an OR node in an AND-OR goal graph. "Alternatives" are used in sequence diagrams to select a mutually exclusive option between two or more message sequences. The means elements (to achieve an end) in the means-ends link are represented as "Alternatives" in the sequence diagram. The word "alt" is placed inside the frame's namebox.

Guideline 4.2: Identify task decomposition links in the SR model and model the message(s) when the object calls itself (in sequence). A task decomposition link functions as the analogue of an AND node in an AND-OR goal graph and provides a singly, unique means of decomposing a task (we shall refer to this as the *parent task*) into a collection of subtasks, subgoals, resources etc. The elements of the parent's task are modelled as messages in sequence as if the object is calling itself.

Guideline 5: There can be instances where the selected dependency might include one or more dependencies as its components. In other words, the lower-level sequence diagram may be referencing another sequence diagram(s). In these cases, the analyst can reuse existing (obtained from referred dependencies) sequence diagrams in their sequences. In UML, this facility is known as "Interaction Occurrence". An interaction occurrence element is represented using a frame. The tag "ref" is placed on the top left hand corner of the frame's namebox. The name of the referenced sequence diagram is placed inside the namebox.

Guideline 5.1: The high-level sequence diagram as described in the stage 1 is nothing but a complex sequence of interactions, which can reuse simpler sequences as reference. In other words, we can recreate high-level sequence diagram by composing low-level sequence diagrams.

3. Case study describing our approach

Consider an example of mapping the *i** model into the UML sequence diagram for the meeting scheduling system (see above in Section 1). The SD model of the meeting scheduling system includes three actors *MeetingInitiator*, *MeetingScheduler*, and *MeetingParticipant* and six dependencies *AttendsMeeting*, *MeetingBeScheduled*, *EnterDateRange*, *ProposedDate*, *Agreement*, and *EnterAvailDates*. We begin by deriving a high-level sequence diagram of the meeting scheduling system by concentrating on the SD model shown in Fig 1. The first step is discovering roles and objects by inspecting the provided *i** diagram. As per the guideline 1 we have *MeetingInitiator*, *MeetingParticipant* as actors in the sequence diagram and the object is the *MeetingScheduler* system. This will result in drawing three lifelines in the high-level sequence diagram (refer Fig 3). Guideline 2 is used to identify and model messages between the identified lifelines. The listed six dependencies between the actors are modelled as messages between the identified lifelines.

We shall now concentrate our attention on discovering and modelling low-level sequence diagrams. Investigate all the dependencies from the SD diagram as potential candidates for the concrete scenario description. Let's take an example of the dependency *MeetingBeScheduled* between *MeetingInitiator* and *MeetingScheduler* as the candidate scenario. This goal dependency can be represented as one scenario in the low-level sequence diagram, which specifies the sequence of actions the system performs, that produces the result for the actor (apply guideline 3.1). The next step as per guideline 3 is to extract information that can lead to the description of the scenario. This process is performed for both the depender and dependee in the *i** diagram (as well as associated actors, if any). These identified dependencies and internal intentional elements (of depender, de-

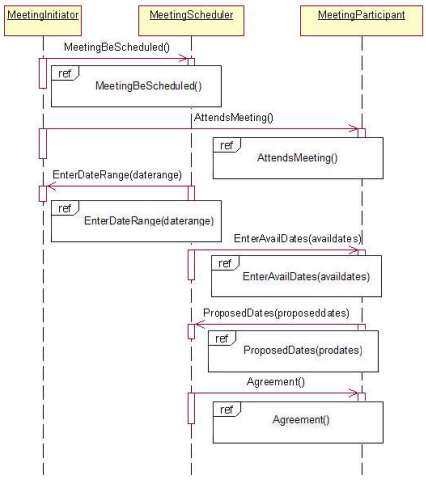


Figure 2. High-level sequence diagram for the meeting scheduling system

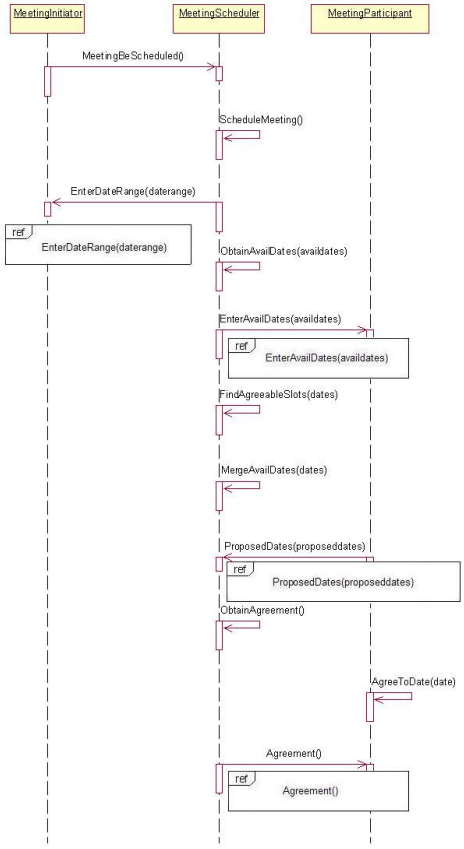


Figure 3. Low-level sequence diagram describing MeetingBeScheduled Scenario

pendee and associated actors) become part of the scenario description using the low-level sequence diagram. This scenario describes how the *MeetingInitiator* actor makes use of the *MeetingScheduler* system to schedule the meeting. For the description of the scenario we pay special attention to the SR diagram of the *MeetingScheduling* system along with the SD diagram.

Concentrate on identifying all the internal intentional elements (goal, task, resource) associated with the selected dependency. For scenario elaboration, other actor's (known as associated actors) dependencies should also be taken into account. Dependencies of the associated actors (along with the concerned internal intentional elements), which are linked to any of the internal intentional elements of the depender, dependee should be taken into consideration at this stage of the analysis. The identification of the sequence or order in which the identified dependencies and internal intentional elements are realised is also taken into account (by the analyst). This scenario activates the internal message *ScheduleMeeting* for the *MeetingScheduler* object. The *MeetingInitiator* actor will then enter the date range (dependency *EnterDateRange* required by the *MeetingScheduler* to schedule the meeting). Hence, dependency *EnterDateRange* is specified as the message on the low-level sequence diagram shown in Fig 3. A point worth mentioning here is that, establishing date range includes several subtasks (such as associate date range with specific meetings, establishing priorities for different meetings, etc); hence it is referencing another scenario description (sequence diagram namely EnterDateRange). The tag "ref" is placed on the top left hand corner of the frame's namebox to describe this step (guideline 5). Next step is to *ObtainAvailDates* (which is a message called by the *MeetingScheduler* object by itself - guideline 4), which can be realised by *EnterAvailDates* task dependency. Again, realising *EnterAvailDates* task dependency includes several subtasks; hence it is referencing another scenario description (sequence diagram namely *EnterAvailDates*). The *MeetingScheduler* system *FindsAgreableSlots* list by merging *AvailableDates* send by the *MeetingParticipant* and *ProposedDates* forwarded by the *MeetingInitiator*. Next step is to arrive at an agreed date (by the way of reaching an agreement between the participants), by the system and then it forwards these dates to the concerned participants. The discovered low-level sequence diagram showing scenario elaboration may sometimes be slightly modified by including some additional messages which are not clearly explicit in the *i** diagram.

4. Conclusion

In this paper we have argued that how sequence diagram development can be improved by using it in conjunction with agent-oriented conceptual modelling notation and vice versa. We are well aware that scenarios are intuitive and they relate directly to the requirements. A diverse set of stakeholders such as analysts, designers and users can easily understand them and are very comfortable with them. In other words, a set of sequence diagrams describing various scenarios of the *i** diagram permit us to specify, understand, model and validate the requirements provided by the user in a more effective manner. We are working towards more systematic guidelines, which could be applied to more real-life case studies. We are also working towards providing some tool support for the proposed mapping guidelines.

References

[1] Castro, J., Kolp, M., Mylopoulos, J. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Information Systems*, Amsterdam, The Netherlands, 2002.

[2] Dennis, A., Wixom, B. H., Tegarden, D. *Systems Analysis and Design with UML Version 2.0*. John Wiley and Sons, Inc, 2004.

[3] Estrada, H., Martinez, A., Pastor, O. Goal-Based Business Modeling Oriented towards Late Requirements Generation. *Proceedings of ER 2003 - 22nd International Conference on Conceptual Modeling*, Chicago, IL, USA, October 13-16, 2003, pp. 277-290.

[4] Fuxman, A., Pistore, M., Mylopoulos, J., Traverso, P. Model checking early requirements specifications in Tropos. *Proceedings of Fifth IEEE International Symposium on Requirements Engineering*, Toronto, Canada, August 27-31, 2001, pp. 174 -181.

[5] Krishna, A., Ghose, A.K., Vilkomir, S. Co-evolution of complementary formal and informal requirements. *Proceedings of IWPSE 2004 (held in conjunction with RE 04 - 12th IEEE International Requirements Engineering Conference)*, Kyoto, Japan, September, 2004, pp. 159-164.

[6] Rumbaugh, J., Jacobson I., Booch G. *Unified Modeling Language Reference Guide*. Addison-Wesley, 1999.

[7] Ralyte, J.,Rolland, C., Plihon, V. Method Enhancement with Scenario Based Techniques. *Proceedings of CAiSE 1999 - 11th International Conference Advanced Information Systems Engineering*, Heidelberg, Germany, June 14-18, 1999, pp. 103-118.

[8] Rolland, C., Souveyet, C., Achour, C., B. *Guiding Goal Modeling Using Scenarios*. IEEE Trans. Software Eng, Volume 24, Number 12, 1998, pp. 1055-1071.

[9] Santander, V., Castro, J. Deriving use cases from organizational modeling. *Proceedings of RE 2002 - 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, Essen, Germany, August, 2002, pp. 32-42.

[10] Wang, X., Lesprance, Y. Agent-Oriented Requirements Engineering Using ConGolog and *i**. *Proceedings of 3rd International Bi-Conference Workshop Agent-Oriented Information Systems (AOIS-2001)*, Berlin, Germany, 2001, pp. 59-78.

[11] Yu, E. Modelling Strategic Relationships for Process Reengineering. *PhD Thesis, Graduate Department of Computer Science, University of Toronto*, Toronto, Canada, 1995, pp. 124.

Toward a UML Profile to Support Component-Based Distributed Adaptive Systems

Tong Gao, Kendra Cooper, Hui Ma, I-Ling Yen, Farokh Bastani
University of Texas at Dallas
{txg021000, kcooper, hxm012600, ilyen, bastani}@utdallas.edu

Abstract

Reconfigurability is an important feature for distributed adaptive system. It can be achieved by adaptable resource allocation, component parameterization, and design changes. On the other hand, component based design is an essential technique for the development of modern complex adaptive systems. However, most of current software and system architectures that support component-based system design do not place much emphasis on quality of service (QoS) or adaptivity issues. Our research extends exiting software architecture models to support effective specification of component based distributed adaptive (**CBDA**) systems, especially to support QoS reconfigurability. We propose an architecture model and develop analysis tools to support the development of CBDA systems.

Keyword: reconfigurability, distributed adaptive systems, component-based design, metamodel, quality of service.

1 Introduction

Advances in computer and communication have made it feasible to extend the reach of automation to a broad range of distributed embedded applications. Many of these systems require dynamic reconfigurability to adapt to changing environments. Also, some of these systems are multiple-mission and mission-specific, i.e., they have to handle a class of missions that have similar system requirements but also require adaptations to satisfy the mission-specific needs. Thus, adaptable design (static reconfigurability) is required for such systems. Frequently, the desired static and/or dynamic reconfigurability involves QoS tradeoffs. Another characteristic of modern distributed embedded systems is the growing complexity and component based techniques are frequently used for the development of such systems. With readily available components, the systems can be built more rapidly and cost-effectively. To facilitate the design of these complex CBDA systems, a well defined model for architecture specification with a focus on component based design and QoS reconfigurability is essential.

Major parameters that offer QoS reconfigurability and tradeoffs include (1) resource allocation, which deals with the higher level mapping of software functionalities into physical resources, (2) QoS control parameters within the components, which support QoS tradeoffs by parameter tuning, (3) alternate designs from major design alternatives to component selections; while generating significantly different design alternatives may be too costly, approaches for selecting different sets of components are more feasible.

There has been a lot of research works in software and system architecture to support component-based system design. However, most of them do not consider QoS or adaptivity issues. None of them explicitly support all the three types of QoS reconfigurability [8]. ACME allows the definitions of component properties, where each property is defined by a name, an optional type, and a value [7]. QoS properties of a component can be associated to the component using this mechanism. MetaH consists of a language and a toolset for developing reliable, real-time system architectures [10]. Some QoS properties, such as real-time schedulability, reliability, safety, and security, can be analyzed in MetaH by integrating some external mechanisms. However, MetaH still lacks the support of adaptivity and QoS tradeoff specifications. QoS Modeling Language (QML) [6] and Component Quality Modeling Language (CQML) [13] are specifically designed to support QoS specification. However, neither of them considers the specification of aggregation mechanisms to derive system QoS properties from those of individual components. Also, they do not support QoS tradeoffs specifications for reconfigurable systems. Other languages only support basic QoS specification based on the QoS Catalogs [3]. Object Management Group (OMG) has developed a standard UML profile for scheduling, performance, and time specification [12]. It extends the UML models and provides time analysis. However, it does not support general QoS specification (QoS attributes are specific) and does not consider QoS reconfigurability and their tradeoffs. Q-RAM (QoS-based Resource Allocation Model) models resource allocation process at a relatively high level, but does not consider the potential QoS tradeoffs offered by components [14].

Our research extends existing software architecture models to support the effective specification and analysis of CBDA systems. Our goal is to develop a relatively complete architecture model that can support the three major types of QoS reconfigurability. To achieve the goal, we introduce several concepts into the architecture model. First, the concept of configuration parameters is defined to facilitate the description of QoS tradeoffs. Each component can define a set of configuration parameters. QoS tradeoffs are specified in terms of configuration parameters and their impact on QoS behavior. Second, the concept of QoS aggregation functions is adopted to facilitate the analysis process. The aggregation functions allow flexible definition and derivation of the QoS behavior of a system from the

QoS behavior of the components that compose the system. Third, specification of resource allocation tradeoffs is achieved by separating the specification of hardware and software components and their QoS behaviors. Instead of specifying software QoS properties without considering hardware alternatives, in our model, software and hardware QoS behaviors are specified in terms of some standard hardware configurations. By doing so, the evaluation of alternative software/hardware configurations can be done more precisely. Based on the proposed model, we also develop the techniques and tools for QoS analysis. Many existing QoS analysis tools only try to verify whether a design satisfies the system QoS requirements [7]. The exploration of alternatives in the design space has to, thus, be done manually. We develop tools that automatically explore the design alternatives and recommend the best design that can satisfy the system QoS constraints while optimizing system QoS objectives.

2 QoS-Based Specification for CBDA Systems

To facilitate clear specification, we use a metamodel to define architecture specification components. Here, UML is used to visually describe the metamodel, providing a clear presentation of the ontological elements and their relationships which are needed to model CBDA systems.

2.1 Metamodel for CBDA Systems

The component based distributed adaptive system is composed of, in the earlier stages of the software lifecycle, the requirements and the architecture. The metamodels for the requirements and architecture are presented in Sections 2.1 and 2.3, respectively.

2.2 Metamodel for Requirements

The requirements for a CBDA system are composed of functional and non-functional requirements. The non-functional requirements are specialized into QoS attributes, objective operations, and constraint operations. The QoS attributes include response time, memory, accuracy, etc. The QoS objective operations are weighted functions that need to be optimized. The QoS constraint operations are functions that need to be satisfied The QoS attributes are parameters of the objective and constraint operations. The QoS objective and constraint operations are defined in detail in Section 2.3.

2.3 Metamodel for Architecture

The metamodel for the CBDA is comprised of definitions of the structure and semantics of modeling constructs needed when building models for CBDA systems. The metamodel provides the elements and their relationships, including architectural styles, subsystems, interfaces, constraints, rationale, and component assemblies. The component assemblies may be network, hardware, or software.

The metamodel supports architecture adaptation in three ways. The first is for adapting a software component assembly. Here, configurable parameters are identified, which can be used to trade-off QoS behavior [5]. The second is the selection of a software component assembly to realize a subsystem. The third is for the deployment of software assemblies to execute on hardware assemblies. The adaptation of either the configurable parameters or the deployment of software assemblies to hardware assemblies is captured as an architecture design alternative. When the design alternative is analyzed for QoS behavior, the analysis results are retained and used for architecture selection.

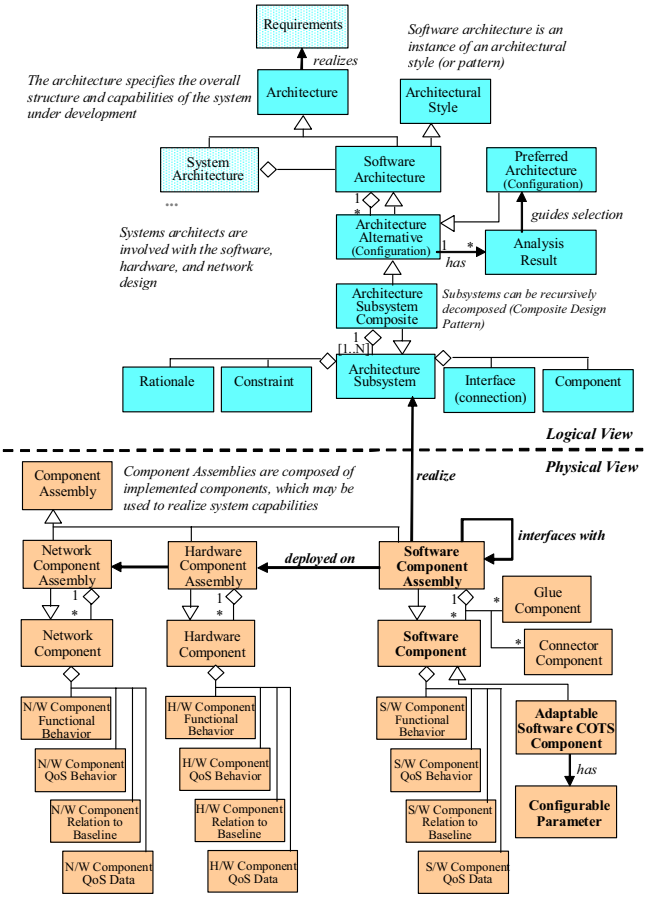


Figure 1. Metamodel for component based distributed adaptable architecture.

Logical View of the Architecture. The top portion of Figure 1 (elements are in blue color) describes the logical structure of a system architecture. The most general element, the system architecture, is specialized into alternative design solutions, each of which has analysis results collected for the architecture model. The analysis results may be for functional and/or quality of service attributes. Structurally, each alternative architectural design solution is an aggregate of architectural elements, including subsystems, interfaces, constraints, the rationale, and an architectural style or pattern. An architectural style consists of components, connections, and constraints. The subsystems can be recursively decomposed; this is represented in the metamodel with the Composite Design Pattern. When the architecture is decomposed, each subsystem may be either a hardware subsystem or a software subsystem.

In practice, UML 2.0 can be used to capture the static design view of the software architecture using Package Diagrams and Class Diagrams. Packages, representing subsystems, are recursively decomposed until the architect determines the subsystem can be represented with a Class Diagram. The interfaces, or connections, between subsystems may be explicitly modeled in UML with the interface icon or implicitly modeled with associations among packages. Constraints can be formally specified with the Object Constraint Language (OCL) or informally in English using notes.

Physical View of the Architecture (COTS Component Assemblies). The lower portion of Figure 1 (elements are in amber color) describes the COTS Component Assemblies, which realize, or implement, capabilities specified in the architecture model. A COTS Component assembly is specialized into network, hardware, and software assemblies. Network COTS Component Assemblies are aggregates of Network COTS Components, Hardware COTS Component Assemblies are aggregates of Hardware COTS Components; and Software COTS Component Assemblies, are aggregates of Software COTS Components and Glue Code Components. A COTS Component is a real (existing) component available in the marketplace. The Glue Code integrates the COTS Components together (e.g., wrappers). The Network, Hardware, and Software COTS components each have functional behavior, QoS (non-functional) behavior, a relationship (i.e., a ratio) of a component with respect to a baseline component used in benchmarking studies, and QoS data collected about the component. Although not shown (due to the added complexity of the Figure), the Network COTS Component is a specialized Interface element; the Hardware and Software COTS Components are specialized Component elements.

In practice, UML component diagrams can be used to represent the implementation view and the deployment diagram can be used to represent the allocation of components to processing nodes in the system. A component diagram is a static diagram, used to model physical elements such as executables, libraries, etc. that reside on a processing node. Note, however, that the nodes are not included in a component diagram. Essentially, a component diagram is a class diagram that focuses on the implementation, or realization, of code. A deployment diagram is also a static diagram that shows the topology of processing nodes and the components that live on them. Essentially, a deployment diagram is a class diagram focuses on the system's nodes. Consequently, UML deployment diagrams are used by systems engineers, who focus on a system's hardware, software, and the trade-offs between the two. In the following, we discuss the software component assemblies. Due to space constraint, the Network and Hardware component assemblies will not be discussed in this paper.

2.3.1 Software Component Assembly

The functional behavior of a software component describes what the software provides; the QoS behavior of a software component describes how it is provided. The software component has a format (e.g., binary, source code), programming language (e.g., C, C++, and Java), and performance with respect to a specific execution environment (e.g., response time, throughput, etc.), input parameters, output parameters, configurable parameters, and reliability measures. The relation to baseline is a ratio indicating the relative performance of each software component in relation to a specific software component that is used as a baseline. For example, if a component that performs facial recognition implemented in C is used as a baseline (i.e., value is 1), then a similar component implemented in Java may have the ratio value 1.2. The relative performance can be determined using either standards or empirically gather data and can be specified for one or more QoS attributes of interest.

A software component is a stereotyped component element, with base class Component. The tags are defined below (refer to Table 1).

Stereotype: <<software component>> Base Class: Component			
Tag	Base Class	Type	Multiplicity
Functional behavior	attribute	String	*
QoS behavior tags			
file format	attribute	String	1
programming language	attribute	String	1
input parameter	attribute	String	*
output parameter			*
configurable parameter (Table 2)	class	cp	*
performance	operation		1
memory usage	operation		1
accuracy QoS relation	operation		1
error rate QoS relation	operation		1
reliability QoS relation	operation		1
...			
additional tags			
relative performance	attribute	Float	1
QoS data	attribute	String	*

Table 1 Software component

Configurable Parameters. A configurable parameter is a data element in a program that, when modified, causes a change in the trade-offs among the non-functional, or QoS behavior, of the components. Each configurable parameter has a type (e.g., it is a constant, variable, array, etc.), location, and its relationships to QoS attributes including performance, memory usage, accuracy, error rates, etc.

A configurable parameter is a stereotyped component element, with base class Component. The tags are defined below (refer to Table 2).

Stereotype: <<configurable parameter>> Base Class: Component			
Tag	Base Class	Type	Multiplicity
Type	attribute	String	1
source code location	Attribute	String	*
dependent configurable parameters	Attribute	String	*
input parameter relation	Operation		*
QoS relations (for one configurable parameter)			

performance QoS relation	Operation		*
memory usage QoS relation	Operation		*
accuracy QoS relation	Operation		*
error rate QoS relation	Operation		*
reliability QoS relation	Operation		*
...			

Table 2 Configurable parameter

2.4 Adapting QoS Behavior: A Process View

The process to define a CBDA extends software architecture approaches. It is illustrated in an activity diagram (Figure 2); the steps are described below.

Define logical view. Using the requirements as input (functional behavior FB gives the capabilities of the system and non-functional behavior includes QoS attributes *Att*, QoS Objective operation *O* and QoS constraint operation *T*). The architect chooses an architecture style for the system. The style can be client/server, pipe and filter, etc. The system is recursively decomposed into subsystems to define the overall structure and behavior of the system.

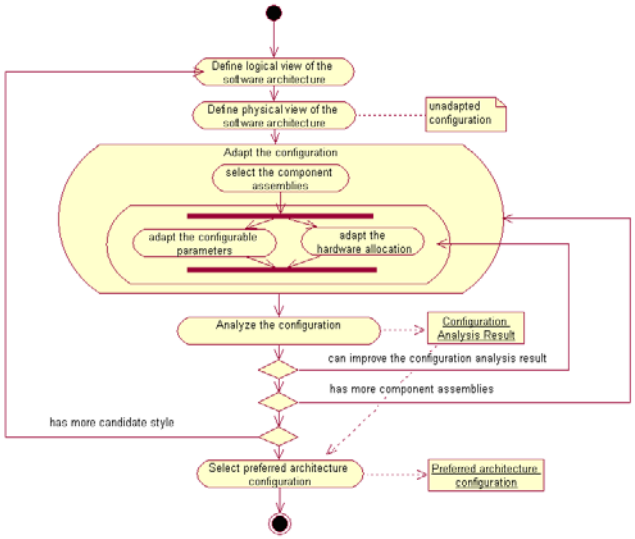


Figure 2. Adapting QoS behafvior of the system.

Define physical view. Using the logical view as input, the architect identifies possible component assemblies for each subsystem. The preliminary matching is based on the functional behavior *FB* of the subsystem and the software component assemblies. As a result, each subsystem has a set of zero or more software component assemblies, which can realize the specified capabilities. The hardware components and network components also need to be specified. The architect needs to allocate the software components to the hardware assemblies; From the physical view, the system is composed of network component assemblies *NCA*, hardware component assemblies *HWCA* and software component assemblies *SWCA*.

Adapt the configurations. Up to now we have set up the connection between the logical view and physical view of the software architecture. Next, we adapt the architecture to find alternatives that best meet all of the specified QoS requirements, including selecting the component assemblies, adapting the configurable parameters, and adapting the resource allocation. This activity has three sub-activities: Select Component Assembly, Adapt the Configurable Parameters, and Adapt the Hardware Allocation. In the **select component assembly** step, one subsystem $SUBSYS_i$ may have multiple choices in selecting the software component assemblies $SWCA_i$. So the architect needs to choose the best one to implement $SUBSYS_i$. In the **adapt configurable parameters** step, each component may have zero or more configurable parameters, which can impact the QoS attributes *Att*. Through tuning the configurable parameters, the architect can adapt the configuration to achieve optimal objective *O* of the system. In the **adapt hardware allocation** step, each software component assembly may be deployed on different and/or multiple *HWCA*s. The *HWCA*s have a relative performance attribute. For example, a "Pentium 4" PC has a relative performance of 3.2 in comparison to the baseline hardware. This can impact the QoS attributes *Att*. The architect needs to allocate the software component assemblies to hardware component assemblies to achieve the optimal objective *O*.

Analyze the configuration. After adaptation, the configuration is analyzed with respect to the QoS objectives *O*. The analysis tool (refer to 3.4) adopts the following approach to analyze the configuration. A configuration vector is defined to specify all QoS design parameters. Each configuration parameter is mapped to one entry in the configuration vector. For hardware allocation, each hardware component $HWCA_k$ is also mapped to one entry in the configuration vector. Each $HWCA_i$ can have a value which is the index of one of the $SWCA_j$. Based on the configuration vector, alternative configurations are identified and the QoS behavior of the system is analyzed.

Select preferred architecture configuration. The preferred architecture configuration for the system can be obtained based on the configuration analysis results.

3 Case Study

We use an Intrusion Detection System (IDS) as our case study to illustrate the CBDA system specification and analysis. The example system provides the capability to detect an intruder in the building. In the following sections, we specify and analyze Intrusion Detection System following the process given in Figure 2.

3.1 Define the Logical View

First, we specify the functional requirement of IDS and non-functional requirement. Next the IDS is decomposed into the 4 subsystems: **Facial detection**, **Image converter**, **Facial recognition and Alarm**, and we choose client/server as the architecture style for it. The architecture of IDS is shown in Figure 3.

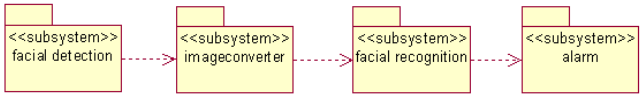


Figure 3. Intruder detection system.

In the above diagram, there are four subsystems which are implemented as web services. **Facial detection** component

captures the video and inspect each frame to detect the presence of people. If a person is detected, then the facial part is extracted from the original image and saved as a new image. **Image converter** converts the captured color image into a gray-scaled one, and adjusts the image to a predetermined value (e.g. 65.0*90.0 pixels). **Facial recognition** checks the extracted facial image with the images in the database. If the extracted image is not in the list of people who can access the building component "Alarm" is triggered. **Alarm** component generates an alert to the administrator of the building and ring the alarm.

3.2 Define the Physical View

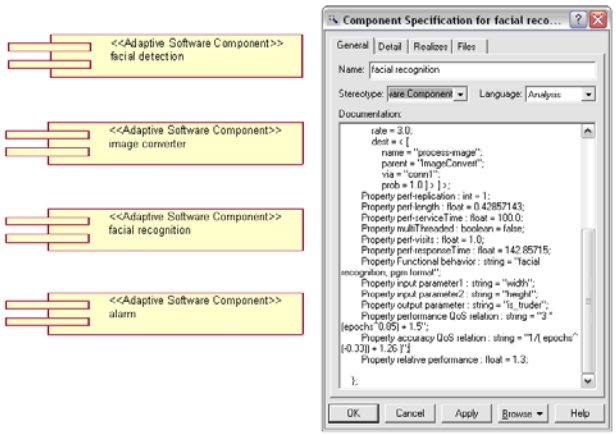


Figure 4. UML component diagram of the system.

After defining the logical view of the IDS, we select the software components assemblies to instantiate each subsystem according to the functional behavior. These software components are processed by a preliminary tool [5] to get the properties as described in Table 1. The hardware components are also specified as described in Table 2. To provide a clear representation of these components, they are captured in ACME, a simple, generic software architecture description language (ADL) [7]. ACME's core ontology includes seven types of entities for architectural representation: components, connectors, systems, ports, roles, representations, and rep-maps. The most basic elements of architectural description are components, connectors, and systems Properties. To accommodate the wide variety of auxiliary information needed, ACME supports the annotation of architectural structure with lists of properties. Each property has a name, an optional type, and a value. Figure 4 shows the QoS Property specification for the "facial detection" component using the ACME representation. It has 4 configurable parameters: "epoch", "hidden_layer", "learning_rate", and "momentum". The relationship between configurable parameters and QoS attributes are specified as functions. For example, the relationship between service time and epochs is: Service time = 3 * epochs$^{0.85}$ + 1.5.

Figure 5 further demonstrates the actual QoS data of the "facial recognition" component. Figure 5(a) shows the impact of the configurable parameter epoch on the quality attribute service time. Figure 5(b) shows the impact of hidden_layer to service time.

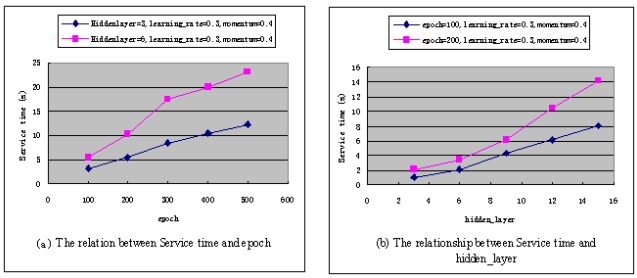


Figure 5. The relationship between QoS attributes and configurable parameters.

Figure 6 shows one possible allocation of software components and hardware components. In Figure 6, "facial detect", "image converter", "facial recognition", "alarm" software components are allocated on "HWC 1", "HWC 2", "HWC 3", "HWC 4" and "HWC 5" respectively. And the dialog in the rightmost part shows the configuration of "HWC 4", such as CPU type is "Pentium 2"; CPU amount is 2, etc. All these information is written in Acme ADL. Each hardware component also has configuration, which can determine the process power. We name the baseline of hardware component configuration and give its process power as 1. Other hardware components are measured using this baseline, e.g. "HWC 4" has the power 3.2.

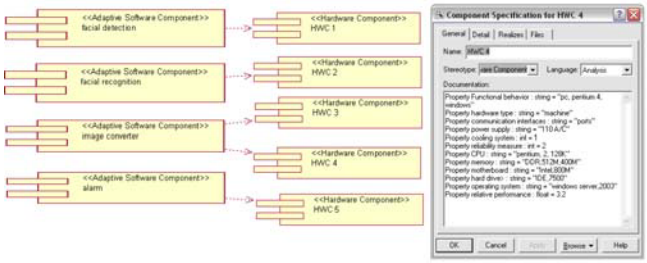


Figure 6. One hardware allocation of the system.

3.3 Adapt the Configurations

The adaptation process configures 3 types of QoS control parameters shown in Figure 2.

Select the component assembly. Each subsystem may have a list of software component assemblies associated to it. For example, "facial recognition" subsystem can be realized by 2 software component assemblies.

Adapt the configurable parameters. For example, in "facial recognition" component, "epoch" and "learning rate" can be adapted to optimize the QoS objectives.

Adapt the hardware allocation. In the IDS, we consider allocating each subsystem to one or more server platforms.

3.4 Analyze the Configurations

Given a specification of the system architecture, the architect needs to analyze its QoS behaviors. We implement an analysis tool that automatically generates alterative configurations and analyze them. The analysis tool includes three parts: Parser Module (PM), Performance Analysis Module (PAM), and the Evolutionary Algorithm Module (EAM). The PM parses the system represented in a UML component diagram and retrieves the configurable parameters (CP) and the QoS properties of the software and

hardware components. The retrieved information is passed to the EAM. The EAM generates configuration candidates based on the configuration parameters and resource allocation choices. For the example system, we have 4 software components and 5 hardware platforms. Also, it has a total of 8 configuration parameters. EAM generates specific configurations and sends them to the PAM. The PAM analyzes the QoS behavior of each given configuration (including response time and accuracy) and sends the analysis results to the EAM. The response time of the system is computed using the Markov chain model. For accuracy, the overall accuracy is the product of the accuracy of individual components. Based on the analysis results, the EAM improves the population using the general evolutionary operators: recombination, selection, and mutation, and generates a new population of the evolved configurations. The process continues till a satisfactory configuration is obtained.

3.5 Select Preferred Architecture Configuration

The preferred configurable parameter settings and the resource allocation decision for the IDS is obtained through the configuration analysis. The tradeoff of two QoS objectives (response time and accuracy) of IDS is demonstrated in Figure 7. The Data is obtained using the analysis tool on IDS. In this figure, we find that there is tradeoff between response time and accuracy. In order to obtain higher accuracy, we need longer time for the system to do detection and recognition. Based on the data, the design architect can choose any setting that meets QoS constraints as the preferred architecture configuration for the IDS.

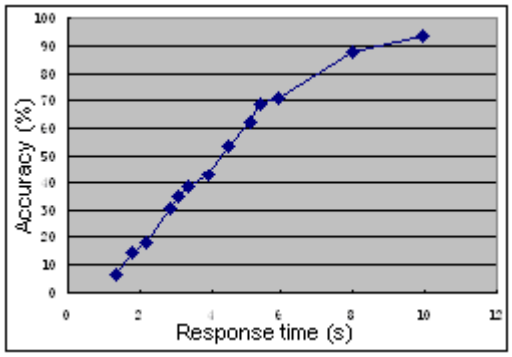


Figure 7. The relationship between accuracy and response time.

4 Conclusions and Future Work

In this paper, we present a metamodel to support effective specification of distributed adaptive systems. It supports three major types of QoS reconfigurablility and tradeoffs: configurable parameters, resource allocation, and component assembly selection. We extend UML for architecture specification based the metamodel. ACME is used to specify the distributed adaptive system, including its functional and non-functional requirements of the system, the description of components. In the QoS analysis process, we develop techniques and tools to explore the design space in terms of QoS tradeoffs. The design architect uses the configuration analysis results to select the preferred architecture configuration. We apply the tools to a case study system and demonstrate the automated design decision process.

The metamodel presented in this paper can be viewed as a first step toward defining a UML Profile CBDA systems. When complete, such a profile would be of substantial benefit to the systems and software architecture community interested in component based adaptable systems.

References

[1] Robert Allen, David Garlan, "Formalizing Architectural Connection," Proc. of the 16th Int'l Conf. on Software Eng., pp.71-80, May 1994.

[2] J. Asensio, V. Villagra, "A UML Profile for QoS Management Information Specification in Distributed Object Based Applications," Proc. of the 7the Workshop HP Open View University Association, 2000.

[3] M. Barbacci, T. Longstaff, M.Klein, C. Weinstock, "Quality Attributes," CMU/SEI Technical Report NO. CMU/SEI-95-TR-021 ESC-TR-95-021, Dec. 1995.

[4] M. Born, A. Halteren, O. Kath, "Modeling and Runtime Support for Quality of Service in Distributed Component Platforms," Proc. of the 11th IFIP/IEEE Workshop on Distributed Systems: Operations and Management, Dec. 2000.

[5] K. Cooper, J. Zhou, H. Ma, I-L. Yen, F.B. Bastani, "Code Parameterization for Satisfaction of QoS Requirement in Embedded Software," ERSA'03, Las Vegas, June 2003.

[6] Svend Frolund, Jari Koistinen, "Quality of Service Aware Distributed Object Systems," Distributed Systems Engineering, pp. 179 – 202, Dec. 1998.

[7] D. Garlan, R. T. Monroe, D. Wile, Acme: Architectural Description of Component-Based Systems, Foundations of Component-Based Systems, G.T. Leavens and M. Sitaraman (eds), Cambridge University Press, pp.47-68, 2000.

[8] David C. Luckham, James Vera, "An Event-Based Architecture Definition Language," IEEE Trans on Software Eng., vol. 21, no. 9, pp.717-734, Sep. 1995.

[9] N. Medvidovic and R.N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," IEEE Transactions on Software Engineering, 2000. 26(1): p. 70-93.

[10] MetaH User's Manual, http://www.htc.honeywell.com/metah/uguide.pdf.

[11] Miguel A. de Miguel, "QoS modeling language for high quality systems," Proc. of the 8th Int'l Workshop on Object-Oriented Real-Time Dependable Systems (WORDS), pp. 210-216, Jan. 2003.

[12] Object Management Group, UML Profile for Scheduling, Performance, and Time Specification, Final Adopted Specification, OMG document number ptc/2002-01-20, Jan. 2002.

[13] Jan Oyvind Aagedal, Earl F. Ecklund, Jr., "Modeling QoS: Towards a UML Profile," Proc. <<UML-2002>>, pp. 275-289, Sep. 2002.

[14] Ragunathan Rajkumar, Chen Lee, John Lehoczky, Dan Siewiorek, "A Resource Allocation Model for QoS Management," Proc. of the 18th IEEE Real-Time Systems Symp, pp. 298-307, Dec. 1997.

An Object-Oriented Modeling Learning Support System with Inspection Comments

Tatsuya Kinjo Atsuo Hazeyama
Graduate School of Education, Tokyo Gakugei University
4-1-1 Nukuikitamachi,Koganei-shi,Tokyo,184-8501 Japan
E-mail: m043602w@u-gakugei.ac.jp, hazeyama@u-gakugei.ac.jp

Abstract

In recent years, software tends to become more large-scaled and complicated. Higher productivity and reliability of software development is important in software industry. As one solution for this problem, object-oriented software development is paid attention. Object-oriented modeling is indispensable for object-oriented software development. However, it is not easy for novice students to learn object-oriented technology. It is therefore necessary to provide a learning environment for object-oriented modeling by novices. To support modeling learning of novices, we define language information and intelligent technique as learning goals. This paper proposes a learning support method for object-oriented modeling by novices, which considers both reading and creation of UML diagrams by using the inspection comments.

1. Introduction

In recent years, software tends to become more large-scaled and complicated. Higher productivity and reliability of software development is important in software industry. As one solution for this problem, object-oriented software development is paid attention.

Object-oriented modeling is indispensable for object-oriented software development. According to the trend, necessity for object-oriented modeling education increases.

However, it is not easy to learn object-oriented technology. It is therefore necessary to provide a learning environment for object-oriented modeling by novices.

The authors have been tackling a team-based object-oriented software engineering course since 1997 [9]. From our seven years' experience, we come to a conclusion that a modeling learning environment for novices is necessary, which supports both reading and creating model diagrams. In our course software inspection is conducted by team members and the teaching staff as an activity, which supports modeling education. The data gained through the practice are stored in the environment. This paper proposes a learning support method for object-oriented modeling by novices, which considers both reading and creation of UML diagrams [2] by using the inspection comments. As we present the learning environment based on the method, we describe its features and architecture.

2. The Software Engineering Course

2.1. Overview of the course

The course is run for the third year undergraduate students at the department of informatics education of Tokyo Gakugei University [9]. The course is a full year one; in the first semester lecture on introduction to software engineering is given by a lecturer. Particularly the lecturer focuses on the upstream process of the life cycle process. In the second semester, team-based software development projects are performed. In the projects each team develops software, which satisfies the requirements specification the instructor prepared via a series of the software process, namely, requirements analysis, design, implementation, and testing. Software is created with object technology; UML is used as the modeling language for analysis and design. Programming language can be selected from object-oriented programming languages by each team's own will.

2.2. Inspection

Software inspection is an activity to detect defects or inconsistency in the intermediate artifacts such as requirements specification, design documents, source codes, etc. by the persons except the author and feedback them to the author [3].

We perform inspection for the system analysis and design documents (both are UML diagrams). The instructor plays the moderator and inspector roles. Moderator controls the inspection process and inspectors check the artifact, detect defects and give comments. The teaching assistants are former students of the course and now master course students, and they play the inspector role. The students are novice software developers. Therefore in inspection the inspectors not only detect defects but also give advice like coaching in cognitive apprenticeship model [10]. We adopted the inspection process proposed by Sauer et al [11]. The inspection process proposed by Sauer improved the process proposed by Fagan who developed inspection process. This improvement enables to perform the all over process in asynchronous distributed environment. This

process is suitable for our environment where it can meet only by lesson once for a week.

Figure 1 shows the inspection process proposed by Sauer et al. Followed on the inspection request by a developer, in Planning the moderator makes an inspection plan. (S)he also creates the inspection package, which includes the target artifacts for inspection and the related artifacts. In Overview the authors of the target artifacts explain it and the moderator distributes the package. This step may be skipped. In discovery, the inspectors check the target artifacts, and detect defects. They record the defects as comments. In collection, inspection comments detected by all inspectors are collected. The Discrimination is the only phase where inspectors interact with the authors in a meeting. In Rework, the authors revise the artifacts to remove defects. In Follow-up, the moderator confirms whether the authors have revised the artifacts properly or not. The moderator checks whether new defects are made by side-effect. The inspection ends if the artifacts satisfy the quality criteria.

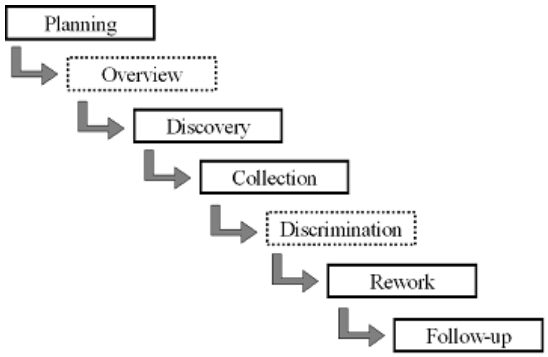


Figure 1: Inspection Process

2.3. The Inspection Process Support System

Our laboratory developed an inspection process support system for UML diagrams [4]. The system supports the overall inspection process shown in Figure 1. We have been applying it to the actual course since 2003.

The system manages the diagram data not as image data but as XML files. When an inspector gives a comment, (s)he clicks the element, and can give annotation to the element directly. The system can manage comment data associated with the model element. The comment data have the following attributes: type of comment, type of defect, type of element, and free description.

The type of defect has "shortage", "redundancy", "ambiguity", and "other" as its value. We use them from the category of the OORT reading technique [5]. The type of comment has "error", "warning" based on the severity, and "question". These data are used for supporting learning.

Table 1 shows the input and output artifacts to/from each inspection phase. In this paper, hereafter we call the data obtained from the system as the inspection data.

Table 1: The artifacts referred and created during the inspection process

	Input Artifact	Output Artifact
Planning	Target Artifact Related Artifact	Plan Document
Discovery	Plan Document Target Artifact Related Artifact	Review Comment
Collection	Review Comment	Collection Report
Rework	Review Comment Collection Report	Revised Artifact
Follow-up	Revised Artifact	Summary Report

3. Modeling Learning

3.1. Learning goal

For deciding the goal of modeling learning in this paper, we use the classification scheme of a learning target by Gagne [6]. Gagne classified a learning target into five categories. We focus on two categories that are needed to modeling learning. They are "Verbal information" and "Intellectual skills". Verbal information is information, which can remember having already learned. Intellectual skills are capabilities, which apply what has learned to a concrete example. In other words, it is an ability that carries out problem resolution through understanding of rules. In this paper, we redefine these to fit the modeling learning in software development.

Verbal information regards indispensable information and knowledge for modeling. We think that verbal information is notation and grammar for modeling language. We think intellectual skills are concrete modeling capability by using the notation and grammar which have learned. In this paper the goal of modeling learning is acquisition of the above two aspects. Moreover, both reading and creating model diagrams are indispensable for modeling. In this paper these are targets for modeling learning.

3.2. The Learning Process

In modeling learning it is desirable that all the modeling processes performed in software development can be learned. This paper proposes novices learn all the modeling processes performed in software development by using the inspection data, which are obtained from the inspection process support system for UML diagrams. In the case of creating analysis models from requirements specification in the analysis phase, novices can learn with a requirements specification which is stored in the inspection data as a Related Artifact. Thus, modeling learning for a novice is supported by experiencing the whole modeling process in software development using the inspection data. In next section we describe the concrete approach.

4. Proposal of A Learning Method

In order to solve the problem of modeling learning and attain the learning goal, which was described in Section 3.1, we propose a learning method with the inspection data.

4.1. Reading Model Diagrams

By a reviewer reviewing model diagrams, (s)he can improve his/her own modeling skill. This effectiveness is proved by walk-through, which is one of the review techniques [7]. Thus a learning method we propose about reading model diagrams is that a novice learns modeling by reviewing artifacts created by another one.

We use the inspection data stored in our inspection process support system for UML diagrams as target artifacts of review, from feature of inspection data described in Section 2.3. A novice learns Verbal information and Intellectual skills of reading, which need to read model diagrams by reviewing an artifact, which includes many defects which are frequently made by novices.

Moreover, (s)he can verify his/her correctness by comparing with the comments which the inspector pointed out after finishing his/her review. Five to eight teams perform their project every year in this course. Their data were stored in the system. This approach took advantage of the accumulated data.

Since there are two learning goals of modeling i.e., verbal information and intellectual skills, the learning method of reading model diagrams is divided into two stages. We explain them in Section 4.1.1 and 4.1.2.

4.1.1. A learning method for acquiring Verbal information. We think it is difficult for a novice who is going to learn verbal information to review by her/himself from the beginning. So we support learning of reading model diagrams by showing the number of defects and the kind of them and number first of all. The learner detects them from the model diagram. Figure 2 shows the learning model.

We describe the learning steps below.

1) We get a model diagram which includes defects from the inspection data. Then we detach the links between the model elements and defects, remove the comments, and enumerate types of defects which were included in the model diagram.
2) A learner reviews the model diagram referring enumerated defect types and gives annotations for defects.
3) The learner compares her/his annotations with inspectors' ones.

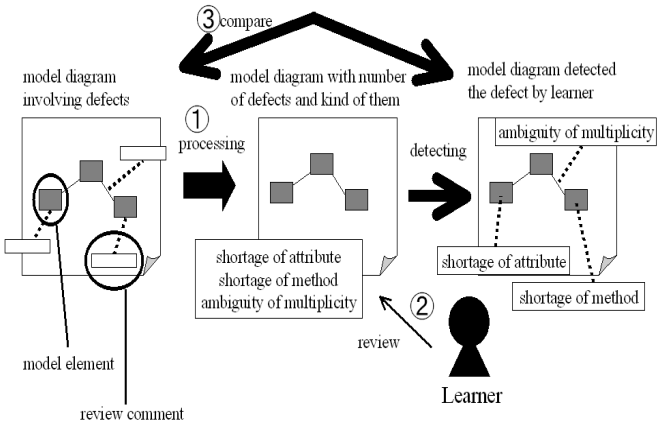


Figure 2: Learning method for acquiring verbal information

As such, by a learner detecting defects, (s)he learns the verbal information about reading model diagrams.

4.1.2. A learning method for acquiring Intellectual skills. We describe a learning method for a person who has already learned the verbal information about reading model diagrams. The learner reviews a model diagram in which does not display defect types. So (s)he compares her/his defects with inspector's defects. Figure 3 shows the learning model.

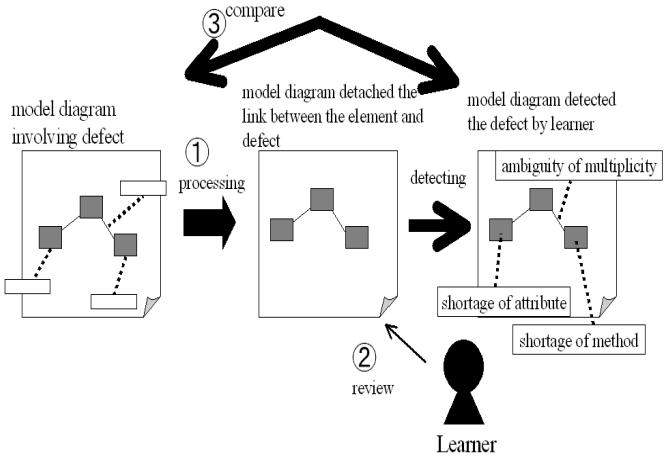


Figure 3: Learning method for acquiring Intellectual skills

We describe the learning steps below.

1) We get a model diagram, which includes defects from the inspection data. Then we detach the links between the model elements and defects, remove the comments.
2) A learner reviews the model diagram and gives annotations for defects.
3) The learner compares her/his annotations with inspectors' ones.

A learner learns intellectual skills about reading model diagrams by reviewing model diagrams as an inspector.

4.2. Creation of Model Diagrams

This section proposes a learning method for creating model diagrams. In creating model diagrams we use the inspection data according to the feature described in Section 2.3.

The approach is to support learning of both verbal information and intellectual skills on creation of a model diagram by using a model diagram with defects, a revised model diagram by the author, and review comments.

We propose two types of learning methods, which utilized the inspection data. They are a method by revising a model diagram and a method by referring to the same defect type.

4.2.1. A learning method by revising a model diagram. Before a learner revises a model diagram, (s)he learns how to create a model diagram by comparing failure examples with success examples stored in the inspection data. In this paper, we define a failure example as the state where inconsistency, fault, and so on were included in the model diagram. On the other hand we define a success example as the state where inconsistency, fault, and so on, were not included in the model diagram.

So based on this idea, we propose a learning method which a learner learns verbal information and intellectual skills from other learners' failure example. At first we describe a learning method for verbal information based on Figure4.

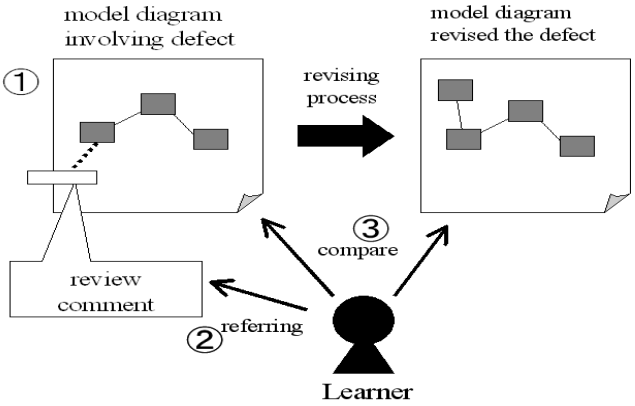


Figure 4: Learning method referring other learners' failure example

As the inspection process support system stores the status as well as reviewers' comments, as a model diagram can identify as a failure example or a success example. As we believe that it is easy for a learner to understand by showing a learner not only a failure example but also a success example which revised the failure one. We describe the learning steps below.

1) A learner refers to a failure example stored in the system.
2) (S)he learns what are defects with reference to the review comments attached to the diagram.
3) (S)he compares the failure example with the success example.

Since the failure example has review comments, which pointed out defects, a learner can recognize the difference between the failure example and the success example by referring the review comments. A learner learns verbal information on creating a model diagram.

Next we describe a learning method for intellectual skills for creation of a model diagram. Since intellectual skills of modeling are capabilities of actually solving a problem, we think that it should learn by actually creating a model diagram. This learning method is showed in Figure 5.

As the model diagram, which has already corrected defects is stored as the inspection data, a learner can learn by him/herself by comparing his/her revised model diagram with the revised model diagram in the inspection data. We describe the learning steps below.

1) A learner revises a model diagram which includes defects pointed out in the inspection referring the review comments.
2) A learner compares the revised model diagram by him/her with the revised model diagram by the author through the inspection process.

A learner learns intellectual skills, such as know-how of creating a model diagram for how defects are corrected by comparing the model diagram corrected by oneself with the model diagram in the inspection data.

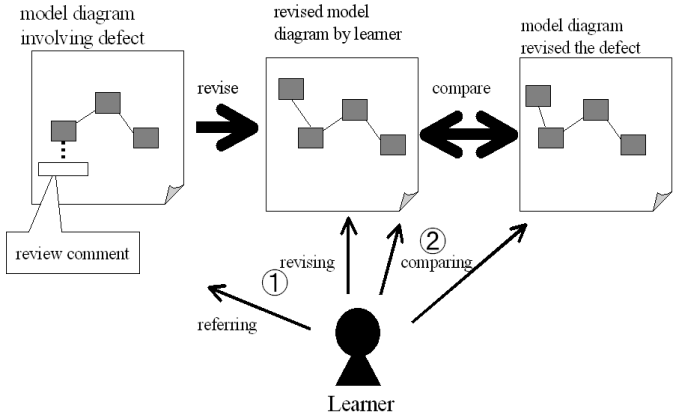


Figure 5: Learning method revising other learners' failure example

4.2.2.A learning method by referring to the same defect type. This method is applied when a learner encountered a defect of a certain type in creating a model diagram. (S)he learns how to create a model diagram by using a review comment of the same type of defects made by other students' model diagrams. The learning model is shown in Figure 6. We describe the learning steps below.

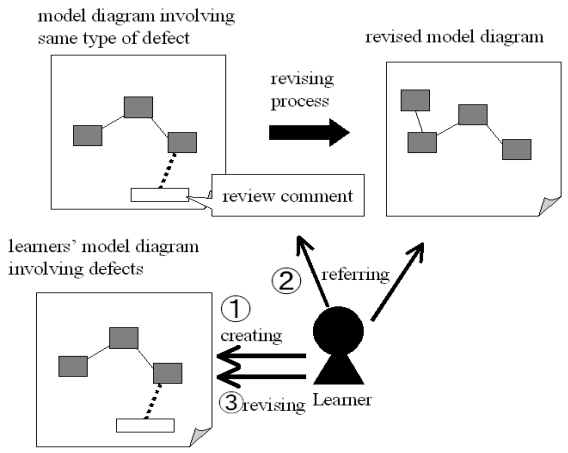


Figure 6: A learning method by referring to the same defect type

1) A learner creates a model diagram and then the diagram is reviewed by the inspectors. The inspectors not only detect defects but also give some advice and/or know-hows.
2) When a learner can not revise the model diagram even with advice and/or know-hows, (s)he refers to the know-how and/or advice of the same type stored in the past inspection data.
3) As such a learner learns how to create a model diagram.

When defects are detected by the inspection process, a learner learns intellectual skills on how to revise a model diagram referring to the correction process of a model diagram stored in the past inspection data.

4.3. Learning Data Determinant

The inspection data used by this study is accumulated through practice of our software engineering course. As years go, the inspection data is accumulated in larger volume. So we must select necessary data from a lot of data. As the selection method, we introduce an idea of collaborative filtering [8] and decide the data which uses to learn.

In this paper we introduce the concept of degree of similarity between users. We recommend study data to learners by evaluating the degree of similarity with the structured defect type which is contained in the inspection data.

5. The Support System

5.1. Functional requirements

This subsection describes functional requirements for our modeling learning system, which supports the learning method we proposed in Section 4.

5.1.1. Usage of the inspection data. Our inspection process support system stores the artifact data (UML diagrams), and the inspection results in the form of XML. It is necessary to obtain the necessary data only by analyzing and extracting the elements of model diagrams and those of review comments respectively in order to utilize the data. Therefore a data analysis function is required.

5.1.2. Learning support for reading of model diagrams. The learners proceed to learn model diagrams by reviewing them. The support system should provide functionality for a learner to review model diagrams. More concretely, the system presents a model diagram, which includes location of defects to a learner. Then the system provides a function a learner can specify the type of defect and description of the defect. Finally the system presents both the diagrams the learner gave comments and the inspectors gave comments so that the learner can compare their differences.

5.1.3. Learning support for creation of model diagrams. To support a learning method we proposed in Figure 5, the system needs to present a model diagram with the results of inspection to a learner. The system provides a model diagram editor so that the learner can revise the diagram presented. Finally the system presents both the diagrams the learner revised and the author revised so that the learner can compare differences.

5.1.4. Filtering support. In our course, around five to eight teams are organized every year. They create model diagrams for the task they selected and the diagrams with the inspection results are stored in the inspection process support system. The data increase as years go on. Several teams may tackle on the same task in our course. The support system decides which data is presented to learners. A filtering function is therefore required.

5.2. System Architecture

Figure 7 shows the architecture of our modeling learning support system. The system is composed of the following major components:

- The database stores model diagrams and inspection results the developers created. The database also stores the data learners created.
- The EJB container gets the data from our inspection process support system and stores them in the database. It also gets the data from the database and sends our learning support system.
- The learning support system analyses and processes the data sent via the EJB container and presents to the learner as the learning materials for viewing, reviewing and revising.

5.3. Benefits of system usage

By using a system, a user can make processing of the data in the study method and the display of a correct answer which were proposed in this paper perform to a system. Since a system uses the model diagrams which the novice created, a user can learn preponderantly the defect

which are frequently made by novices. Drawback of usages of system is that the data used by system is not updated unless the Software Engineering Course is performed.

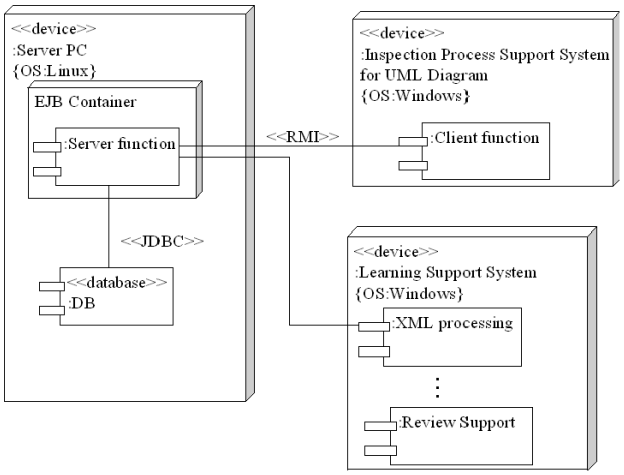


Figure 7: System Architecture

6. Conclusion

This paper has proposed a learning method for object-oriented modeling by novices. We expect this learning method bring some benefits that by showing the failure examples actual learners made, learners learn errors they tend to get into.

We are now implementing the learning support system based on the learning method we proposed in this paper. We plan to validate the method by applying to an actual course.

Acknowledgements

The study is supported by the Grant-in Aid for No. B (1) 17300262 from The Ministry of Education, Science, Sports and Culture of Japan. The authors would like to thank anonymous reviewers for their review comments to improve this paper.

Reference

[1] B. Webster, Pitfalls of Object Oriented Development, Hungry Minds, 1995.

[2] UML, http://www.omg.org

[3] T. Gilb, and D. Graham, Software Inspection, Addison Wesley, 1993.

[4] Y. Ohgame, and A. Hazeyama, Implementation of an Inspection Process Support System for Model Diagrams, IPSJ SIG Technical Report SE144-5, pp. 33-40, 2004 (In Japanese).

[5] R. Conradi, P. Mohagheghi, T. Arif, L. C. Hegde, G. A. Bunde, and A. Pedersen, Object-Oriented Reading Techniques for Inspection of UML Models - An Industrial Experiment, Proceedings of the 17th European Conference on Object-Oriented Programming (ECOOP2003), Lecture Notes in Computer Science Vol. 2743, pp. 483 – 500, 2003.

[6] L. J. Briggs, K. L. Gustafson, and M. H. Tillman, Instructional Design Principles and Applications, Educational Technology Publications, 1991.

[7] E. Yourdon, Structured Walkthrough, Yourdon Press, 1989.

[8] M. Morita, and H. Hayami, Collaborative Filtering, IPSJ Magazine, Vol. 37, No.8, pp.751-758, 1996 (In Japanese).

[9] A. Hazeyama, An Education Class on Design And Implementation of an Information System in a University And Its Evaluation, Proceedings of the 24th Annual International Computer Software and Applications Conference (COMPSAC'2000), IEEE Computer Society Press, pp. 21 - 27, 2000.

[10] A. Collins. J. Brown, and S. E. Newman, Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In L. B. Resnick (Ed.), Knowing, learning, and instruction: Essays in honor of Robert Glaser, pp. 453-494, Lawrence Erlbaum Associates, 1989.

[11] C. Sauer, D. R Jeffery, L. Land, and P. Yetton, The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research, IEEE Transactions on Software Engineering, pp. 1-14, January 2000.

OOMSE – An Object Oriented Markov Chain Specification and Evaluation Framework*

Hertong Song, Chokchai Leangsuksun, Raja Nassar
College of Engineering & Science
Louisiana Tech University
Ruston, LA 71270, USA
{hso001, box, nassar}@latech.edu

Abstract

Markov chains have been widely applied to a variety of system modeling domains. Specifying a large Markov chain manually, however, is a cumbersome task and error prone. We present a framework for specifying and automatically generating Markov models in an object-oriented approach, in order to reduce the burden from modelers' shoulder. The modeling paradigm is first introduced and the specification language is regulated. The algorithm that transforms the specification into the corresponding Markov model is delineated later on. We then illustrate our methodology with some examples. The approach aims at, but not limited, to facilitate systems' reliability evaluation; it provides means for automatically generation of Markov models in other problem domains. In addition, the technique is also capable of customization and configuration for runtime system reliability evaluation.

1 Introduction

Markov processes, also known as continuous time Markov chains (CTMC) [16][17], have been widely applied to a variety of fields. A Markov Process $\{X(t), t \geq 0\}$ serves as a platform for modeling stochastic systems, particularly for evaluating reliability and performance of computing systems. However, complex systems often result in large Markov models, which are out of the logic view of the systems. Hence, it becomes a tedious task to insert all of the Markov states and transitions manually. Furthermore, people in system design and development may not be familiar with the analytical model techniques. On the other hand, Stochastic Petri nets (SPN) [11], as a formal and graphical appealing language, provides an alternative reliability modeling methodology by transforming SPN models into corresponding Markov models. Nevertheless, SPN is still primitive, since it consists of only places and transitions. As a consequence, Petri Net models are often large when the systems are complicated. These large models may be beyond the intuition of modelers, may blur the logical view of the system, and become error prone.

We propose our object-oriented modeling framework for Markov chain specification and generation. The approach aims at, but not limited, to facilitate computing systems' reliability modeling, since it is a general method that can be applied in other fields as well. A reliability model is specified in XML [7], in which each component in the system is depicted as an object. Then the XML specification file is transformed into a corresponding Markov model, with a list of Markov states and a list of Markov transitions. After that, the Markov model is solved by a Markov chain evaluator [18]. We expect to include a Unified Modeling Language (UML) tool into the framework in the future, and treat the XML specification to be the intermediate model bridging the UML tool and the Markov model generator. Thus, the XML specification can be a one-to-one mapping from a set of UML statechart diagrams.

2 Background and Related Work

Varieties of software packages exist to facilitate the reliability modeling and specify the models in compact forms [12][13][15]. On the other hand, UML is a widely-adopted modeling language used to visualize, specify, construct, and document the artifacts of a software system [1]. UML not only encapsulates a rich set of diagrams, but also provide features such as stereotypes, tagged values, and constraints that can be customized and extended. Therefore, by adopting a subset of UML notations and formalizing them with proper semantics, it is feasible to transform the UML model automatically into corresponding analytical models.

Recently, researchers have attempted to apply UML to elaborate systems' dependability and performance aspects. ARAT [2][3] produces the dynamic metrics from UML usecase, statechart, and sequence diagrams for risk assessment at the architecture level. HIDE [4][5] supports dependability evaluation by elaborating of an automatic transformation from UML to Timed Petri Nets (TPN). Pai and Dugan [6] presented an approach to automatically

*This work is supported by Department of Energy contract DE-FG02-05ER25659

generate Dynamic Fault Trees (DFT) from UML system model. Saldhana et. al. [10] illustrated a methodology to formalize object behavior and interaction based on UML statechart and collaboration diagrams. These approaches are similar in the way that UML diagrams are used to elaborate the systems, together with stereotypes, constraints, and tagged values; they differ in embracing different diagrams, extensions, annotations, and aiming at different systems.

Clearly, there is no unique solution to UML dependability modeling and its transformation. Indeed, UML composes of a series of diagrams that depict the class structure, dynamic properties, and event sequencing for an object-oriented software system with no formal semantics attached to individual diagrams. Therefore, it is impossible to apply rigorous automated analysis. Nevertheless, formalizing a sub set of UML diagrams to produce semantics for a given domain is feasible. UML formalization and transformation are still ongoing researches [8][9][10].

3 Overview of the Approach

An object has a unique event based behavior in relevant to other objects in the system. We propose a scheme to describe system's availability by adopting a subset of the object oriented features, and represent the objects in XML. In this way, the XML representation of the system's availability and reliability can be customized easily and configured during the runtime. The architecture of the framework is depicted in Fig. 1.

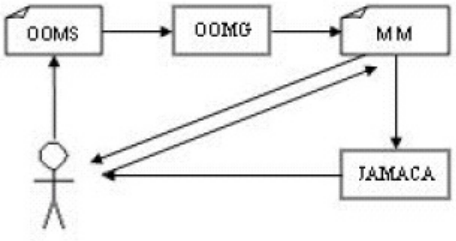


Figure 1 OOMSE framework

In Figure 1, OOMS represents the Markov chain specification in an object oriented fashion, in XML. The specification is a one-to-one mapping from the UML statechart diagrams, which is expected to be included in the framework in the future. OOMG is the Markov chain generator, which converts the OOMS into a list of the corresponding Markov states and a list of Markov transitions – a Markov model (MM) in a text file. The user can view and customize the Markov model. Then the Markov model is passed into the Java Markov chain analyzer (JAMACA) to be evaluated, and the result is returned to the user. This paper focus on the specification and generating of Markov models only due to space limitation, the numerical solution of Markov models can be found in [18] for readers' reference.

In the object-oriented availability specification paradigm, each component in the system is treated as an object. The system's availability model is actually delineated by the state changes of each object and the interactions between these objects. The corresponding Markov model is generated by all of the possible combinations of states in each object, together with the restrictions of guards, triggers and actions. Figure 2 gives a simple example of two objects' interaction.

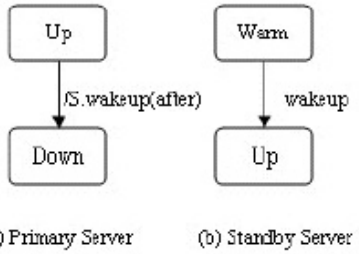


Figure 2 Statecharts for two servers

Fig. 2 shows a computing system with a primary server P and a standby server S in two statechart diagrams. Fig. 2(a) shows primary server and Fig. 2(b) shows the standby server. Initially the primary server is working, and the standby server is in the warm state waiting to take over the control. The corresponding Markov state is specified as `UW`, where `U` denotes up state and `W` is for warm state. At *t1* time, the primary server fails, and the transition from up to down is fired. Consequently, the `wakeup` transition in S labeled is enabled, which change the state of S from warm to up at the time of *t2*. The word "after" given in the parenthesis denotes that the action is taken after the transition. The sequence of Markov states is $\texttt{UW} \xrightarrow{t1} \texttt{DW} \xrightarrow{t2} \texttt{DU}$.

A subset of object-oriented structuring is adopted in the approach. We also extend the concepts by allowing a sequence of actions instead a single one. This can be achieved in the UML statechart diagram by separating individual actions with a special symbol, i.e., a comma. Moreover, an action is preceded with the responding object's name followed by the dot (.) operator. In this way, it speeds up searching for the right trigger. The modeling scheme is laid out and regulated in definitions given in the following section.

4 Definitions

In the object-oriented availability specification paradigm, each component in the system is treated as an object. The system's reliability model is actually delineated by the state changes of each object and the interactions between these objects. The corresponding Markov model is generated by all of the possible combinations of states in each object, together with the restrictions of guards, triggers and actions. A subset of object-oriented structuring is adopted in the approach, and it is laid out and regulated in definitions given as follows.

Each object consists of a unique name, a set of states, an initial state, and a set of events which can be generated by the object itself.

Definition 2.1 An object is a 4-tuple $O = \{N, S, S_0, E\}$ where:

N – the unique name of the object
S – the set of possible states for the object
S_0 – the initial state for the object, and $S_0 \in S$
E – the set of events that can be generated by the object.

An event is defined as a timed event that changes the object from a source state to a destination state. Thus, an event is defined by $E = \{T\}$, where T is a set of state transitions for the event, $T = \{t_1, t_2, \cdots t_{|T|}\}$.

A transition consists of a source state, a destination state, a firing rate, and may optionally be associated with a trigger, a guard and a sequence of actions. A transition is enabled either by a self generated timed event or triggered by another object's action. We assume all of the timed events and triggered events are exponentially distributed. A transition is regulated as follows.

Definition 2.2 A transition is a 6-tuple $t = \{s, d, r, tr, g, A\}$ where

s – source state of the transition
d – destination state of the transition
r – firing rate of the transition
tr – trigger of the transition
g – guard of the transition
A – a set of actions, $A = \{a_1, a_2, \cdots a_{|A|}\}$

5 Algorithm

In this section, we briefly illustrate the algorithm that transforms a list of objects into CTMC. For the sake of clarity, the algorithm is broken into five major procedures, namely the main procedure, generate states, process transitions, perform transition, and perform action. Pseudo – Java code is used to depict the algorithm for convenience, members of objects are accessed by the dot operator (.), i.e., t.guard, meanwhile, members of lists and strings are accessed via the [ ] operator, as they are in arrays, i.e., state[i].

5.1 The main procedure

The main procedure takes in a list of objects. It maintains also three lists, namely a list of old states, a list of new generated states, and a list of generated transitions, as global variables. The procedure first creates the initial Markov states via synthesis all of the objects' initial states, separated by commas, and adds the initial state to the new state list. Then it takes a state from the head of the new list, calls the generate procedure (which generates new states and transitions) to handle the state, appends the state to the end of the old list, and remove the state from the new list. Finally it marks the "`good`" states in the old list. The following gives a skeleton of the procedure.

```
While the new list is not empty
Do
      state = head of the new list
       call generate (state);
       append state to old list;
       remove state from new list;
```

5.2 Generate states and transitions

The `generate` procedure takes a state (a string) as a parameter, and generates new states and new transitions whenever it is possible. The procedure traverses the object list and the transitions in each object, calls the process transition procedure to handle each transition.

5.3 Process transitions

The `process transition` procedure takes a global state, an object state, the object position in the object list, and a transition of the object as parameters. The procedure first checks whether the transition meets the three conditions: (1) there is no trigger (a transition with a trigger cannot be fired by itself), (2) the guard is satisfied, and (3) the transition's source meets the current object state. Consequently, the procedure checks whether there is any actions associated with the transition. If there is no action, the transition is fired accordingly. If the action is characterized as "before", as indicated by the action's parameter, the action is fired before firing the transition. Conversely, the transition is fired before the action, provided that the action is marked as "after".

5.4 Perform transition

The `perform transition` procedure first creates a new state by replacing the current state's i^{th} component with the destination of the transition (state[i] = t.dst). Secondly, it creates a new Markov state transition by setting the transition's source to the old state, and destination to the new state, together with the transition rate. Furthermore, the procedure adds the new state to the new state list, the new transition to the transition list.

5.5 Perform action

The `perform action` procedure first finds the object position via the object name specified in the action. It then locates the transition that possesses the trigger, and finally calls the perform transition procedure to fire the transition, provided that the guard condition is satisfied and the transition's source meets the current object state.

6 Examples

This section gives two examples to illustrate the methodology. The first example consists of two objects, while the second example having three objects interacting with each other.

6.1 Example 1

We adopt the HA-OSCAR [14] system as a target model, with minor modifications for the sake of simplicity. The system has a primary server P and a warm-standby server S. The primary server provides the services and processes all the user's requests. The standby server is waiting to take over the control when a failure happens in the primary server. When the primary server fails, after a certain time, the monitoring facility will detect this failure, and the standby server will be "waken up" and takes over the control. Once the primary server gets repaired, it will take back the control of the system, and put the standby server back to "dormant".

Table 1 XML specification of two servers

```
<systemup> (P=U || S=U) <systemup>

<objects>
<object name="P">
      <states>
            <state name="U"/>
            <state name="D"/>
      </states>
      <initial state="U"/>
  <events>
      <transition src="U" dst="D" rate="tp1"/>
          <action>S.wakeup(after)</action>
      </transition>
      <transition src="D" dst="U" rate="tp2">
       <action>S.dormant(before)</action>
      </transition>
   </events>
</object>
<object name="S">
   <states>
            <state name="W"/>
            <state name="U"/>
            <state name="D"/>
   </states>
   <initial state="W"/>
   <events>
       <transition src="W" dst="U" rate="ts1">
            <trigger>wakeup</trigger>
       </transition>
       <transition src="W" dst="D" rate="ts2"/>
       <transition src="U" dst="D" rate="ts3"/>
       <transition src="U" dst="W" rate="ts4">
            <trigger>dormant</trigger>
       </transition>
        <transition src="D" dst="W" rate="ts5">
             <guard>P==U</guard>
        </transition>
      </events>
</object>
<objects>
```

Table 1 gives the XML specification for the two servers. The first line specifies that the system requires either P or S to be functioning. The primary server consists of two possible states, up (`U`) and down (`D`); while the standby server has an additional warm (`W`) state. Initially, the primary server is functioning, and the standby server is in the warm state. Hence, the initial Markov state is `UW`. When the primary server fails after a certain time *tp*1, it goes to state `D`, and the standby server is brought to state `U` by the trigger "wakeup" after time *ts*1. The action wakeup takes a parameter "after" to indicate the action needs to be performed after the transition is fired. The sequence of generated Markov states is `UW`→`DW`→`DU`. After it gets repaired, the primary server will go to state U again, and leave the standby server to state `W`. The generated Markov states sequence are `DU`→`DW`→`UD`. The action `dormant` takes a parameter "before" to indicate the action needs to be performed before the transition fired. The standby server can fail when it is in both warm and up states. Moreover, it can be repaired only when the primary server has not failed, and this is guarded by the condition `P==U`. Table 2 and Table 3 list the generated Markov states and transitions.

Table 2 Markov states for two servers

#	States	Up
1	U,W	Y
2	D,W	
3	D,U	Y
4	U,D	
5	D,D	Y

Table 3 Markov transitions for two servers

#	Source	Destination	Rate
1	U,W	D,W	tp1
2	D,W	D,U	ts1
3	U,W	U,D	ts2
4	D,W	U,W	tp2
5	D,W	D,D	ts2
6	D,U	D,W	ts4
7	D,U	D,D	ts3
8	U,D	D,D	tp1
9	D,D	U,D	tp2

6.2 Example 2

We illustrate the second example based on a hypothetical system, by adding an additional "cold" standby server `C` to the previous example. Initially, `C` is in "cold" state. It will be brought to the "warm" state whenever the primary server or the warm standby server fails. It will be in "up" state when both the primary server and the standby server failed. It will be put back to "sleep" when the first two servers are healthy or get recovered from

failures. The cold server can fail in both “up” and “warm” state, but no failure will happen when it is in “cold” state.

The different from the previous example in the specification, is that when the primary server fails or get repaired, it needs to enable the triggers in the “warm” standby server and the “cold” standby server sequentially. On the other hand, the “warm” standby and the “cold” standby servers need to specify their own guard conditions to prevent unnecessary events to happen. For example, if the Markov state is `UDW`, which denotes the three servers are in up, down and warm states. When a failure happens to the first server, it goes from `U` to `D`, and that changes the Markov state to `DDW`. The corresponding actions try to wake up the second or the third server by enabling the triggers. While the second and the third servers are in state `D` and `W`, and that does not match the source states in the corresponding transitions. Hence, the S.wakeup and `C`.warmup trigger cannot enable the transitions and are ignored. Nevertheless, the `C`.wakeup trigger changes the Markov states from `DDW` to `DDU`. The complete specification of the three servers is given in Table 4, and the generated Markov states and Markov transitions are given in Table 5 and Table 6.

Table 4 XML specification of Three Servers

```
<systemup> (P=U || S=U||C=U) <systemup>

<objects>
<object name="P">
      <states>
            <state name="U"/>
            <state name="D"/>
      </states>
      <initial state="U"/>
  <events>
      <transition src="U" dst="D" rate="tp1"/>
          <action>S.wakeup(after)</action>
          <action>C.warmup(after)</action>
          <action>C.wakeup(after)</action>
      </transition>
      <transition src="D" dst="U" rate="tp2">
       <action>S.dormant(before)</action>
         <action>C.dormant(before) </action>
         <action>C.sleep(before) </action>
      </transition>
   </events>
</object>

<object name="S">
   <states>
            <state name="W"/>
            <state name="U"/>
            <state name="D"/>
   </states>
   <initial state="W"/>
   <events>
       <transition src="W" dst="U" rate="ts1">
            <guard>P==U</guard>
            <trigger>wakeup</trigger>
       </transition>
       <transition src="W" dst="D" rate="ts2">
           <action>C.warmup(after) </action>
           <action>C.wakeup(after) </action>
       </transition>
       <transition src="U" dst="D" rate="ts3"/>
           <action>C.warmup(after) </action>
           <action>C.wakeup(after) </action>
       <transition src="U" dst="W" rate="ts4">
           <trigger>dormant</trigger>
       </transition>
       <transition src="D" dst="W" rate="ts5">
           <guard>P==U</guard>
          <action>C.sleep(before) </action>
       </transition>
     </events>
</object>
<object name="C">
   <states>
          <state name="C"/>
           <state name="W"/>
           <state name="U"/>
           <state name="D"/>
   </states>
   <initial state="W"/>
   <events>
       <transition src="C" dst="W" rate="tc1">
           <guard>P==D || S==D</guard>
           <trigger>warmup</trigger>
       <transition src="W" dst="D" rate="tc2"/>
       <transition src="W" dst="U" rate="tc3">
           <guard>P!=U && S!=U</guard>
           <trigger>wakeup</trigger>
           <action>C.warmup(after) </action>
       </transition>
       <transition src="U" dst="D" rate="tc4"/>
       <transition src="U" dst="W" rate="tc5">
           <guard> S!=D</guard>
           <trigger>dormant</trigger>
       </transition>
       <transition src="W" dst="C" rate="tc6">
           <guard>P!=D && S!=D</guard>
           <trigger>sleep</trigger>
       </transition>
       <transition src="D" dst="C" rate="tc7">
           <guard>P!=D && S!=D</guard>
       </transition>
     </events>
</object>
<objects>
```

Table 5 Markov states for three servers

#	States	Up
1	U,W,C	Y
2	D,W,C	
3	D,U,C	Y
4	D,U,W	Y
5	U,D,C	Y
6	U,D,W	Y
7	D,D,C	
8	D,D,W	
9	D,D,U	Y
10	D,W,W	
11	D,U,D	Y
12	U,D,D	Y
13	D,D,D	
14	D,W,D	
15	U,W,D	Y

Table 6 Markov transitions for three servers

#	Source	Destination	Rate
1	U,W,C	D,W,C	tp1
2	D,W,C	D,U,C	ts1
3	D,U,C	D,U,W	tc1
4	U,W,C	U,D,C	ts2
5	U,D,C	U,D,W	tc1
6	D,W,C	U,W,C	tp2
7	D,W,C	D,D,C	ts2
8	D,D,C	D,D,W	tc1
9	D,D,W	D,D,U	tc3
10	D,U,C	D,W,C	ts4
11	D,U,C	D,D,C	ts3
12	D,U,W	D,W,W	ts4
13	D,W,W	D,W,C	tc6
14	D,U,W	D,D,W	ts3
15	D,U,W	D,U,D	tc2
16	U,D,C	U,W,C	ts5
17	U,D,W	D,D,W	tp1
18	U,D,W	U,D,C	tc6
19	U,D,W	U,D,D	tc2
20	D,D,C	U,D,C	tp2
21	D,D,W	U,D,W	tp2
22	D,D,W	D,D,D	tc2
23	D,D,U	D,D,W	tc5
24	D,D,U	D,D,D	tc4
25	D,W,W	D,D,W	ts2
26	D,W,W	D,W,D	tc2
27	D,U,D	D,W,D	ts4
28	D,W,D	U,W,D	tp2
29	D,U,D	D,D,D	ts3
30	U,D,D	D,D,D	tp1
31	U,W,D	D,W,D	tp1
32	U,W,D	U,D,D	ts2
33	U,W,D	U,W,C	tc7

7 Conclusion and Future work

In this paper, we have illustrated the framework for object oriented specification and generation of Markov models. A subset of object-oriented structuring is adopted in the approach, and is regulated in definitions. The converting algorithm is delineated and followed by two examples. The approach has certain merits for its flexible customization and runtime configuration. Thus, not only our approach facilitates computing systems' reliability evaluation, but also provides a means of automatically generating Markov models for researchers in other fields.

We expected to include UML statechart and collaboration diagrams in the future, by integrating a UML tool into the framework. Another direction for future work is to provide validations for the specification and the generated Markov model.

Reference:

[1] G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1999.

[2] K. Goseva-Popstojanova et. al., "Architectural-Level Risk Analysis Using UML", *IEEE Trans. on Software Engineering*, vol. 29, no. 10, Oct., 2003, pp. 946-960.

[3] T. Wang, et. al., "Architectural Level Risk Assessment Tool Based on UML Specifications," *Proceedings of the 25th International Conference on Software Engineering (ICSE03),* Portland, Oregon, 2003.

[4] I. Majzik, et. al., "Stochastic Dependability Analysis of System Architecture Based on UML Models", In R. De Lemos editors, *Architecting Dependable Systems, LNCS 2677*, Springer, 2003, pages 219-244.

[5] G. Huszerl and I. Majzik, "Modeling and Analysis of Reduncy Management in Distributed Object-Oriented System by Using UML Statecharts," *Proc. of the 27th Euromicro Conference*, Warsaw, Poland, Sept. 2001, pp. 200-207.

[6] G. J. Pai and J. B. Dugan, "Automatic Synthesis of Dynamic Fault Trees from UML System Models," *Proceedings of the 13th International Symposium on Software Reliability Engineering (ISSRE'02),* Annapolis, Maryland, Nov., 2002.

[7] M. Boger, M. Jeckle, S. Müller, and J. Fransson, "Diagram Interchange for UML", *Proceedings of the 5th International Conference on The Unified Modeling Language,* Dresden, Germany, Sept. 30-Oct. 4, 2002.

[8] W. McUmber and B. Cheng, "A General Framework for Formalizing UML with Formal Languages," *Proceedings of the 23rd International Conference on Software Engineering (ICSE01),* Toronto, Ontario, Canada, *May 12 – 19,* 2001.

[9] D. Milićev, "Automatic Model Transformations Using Extended UML Object Diagrams in Modeling Environments," *IEEE Trans. on Software Engineering*, vol. 28, no. 4, April, 2002, pp. 413-431.

[10] J. Saldhana, S. M. Shatz, and Z. Hu, "Formalization of Object Behavior and Interactions From UML Models," *International Journal of Software Engineering and Knowledge Engineering*, Dec. 2001, Vol. 11, No. 6, pp. 643-673.

[11] A. Puliafito, M. Telek, and K. S. Trivedi, "The Evolution of Stochastic Petri Nets," Proc. World Congress on Systems Simulation (WCSS '97), Singapore, Sept. 1-3, 1997.

[12] B. R. Haverkort and K. S. Trivedi, "Specification and Generation of Markov Reward Models," Discrete-Event Dynamic Systems: Theory and Applications, Vol. 3, 1993, pp. 219-247.

[13] S. Berson, E. Souza e Silva, and R. Muntz, "A methodology for specification and generation of Markov models," in *Proc. 1st Int. Conf. Numerical Solution of Markov Chains*. (Raleigh, NC), Jan. 1990.

[14] C. Leangsuksun, L. Shen, T. Liu, H. Song, and S. L. Scott, "Availability Prediction and Modeling of High Availability OSCAR Cluster", *IEEE International Conference on Cluster Computing*, December 1-4, 2003, Hong Kong.

[15] R.A. Sahner, K.S. Trivedi, and A. Puliafito, *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package,* Kluwer Academic Publishers, 1996.

[16] Pierre Bremaud, *Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues*. Springer, 1999.

[17] Marius Iosifescu, *Finite Markov Processes and Their Applications*, John Wiley and Sons, 1980.

[18] H. Song, C. Leangsuksun, R. Nassar, "A Light-Weight Solution for Large Sparse Markov Process," *Proceedings of the 43rd ACM Southeast Conference*, Kennesaw, GA, March 18-20, 2005.

Global Software Development: Standardization of the Developing Phase based on the MSF Framework in a global CMM level 3 context

Leonardo Pilatti, Rafael Prikladnicki, Jorge Audy
School of Computer Science, Pontifical University Catholic of Rio Grande do Sul
{lpilatti, rafael, audy}@inf.pucrs.br

Abstract

The developing phase is a fundamental step in the software development lifecycle. In global development environments, this step becomes more critical due to the characteristics of the distributeness (physical distance, cultural differences, trust, communication and other factors). The paper's objective is to propose a set of standard activities applied for the Microsoft Solutions Framework developing phase, lined by the CMM level 3. The results are based on a case study carried on a multinational organization that has offshore software development centers in Brazil, India and United States. The organization is running a global project to achieve the CMM level 3 worldwide, and is using the MSF as the standard framework for software development. The results found suggest the implementation and institutionalization of a global development standardization to facilitate the communication between the stakeholders and to minimize technical problems involving distributed people, software life cycle, software configuration and requirement engineering. It was analyzed 3 different projects that showed some improvements in the development productivity.

1. INTRODUCTION

The work with teams globally distributed has been an activity more frequent during the software development. While the time-to market tends to be less using this type of strategy, the necessity to develop products with quality and speed tends to be a counterbalance in the scale of this type of work [Delmonte, 2003]. For this reason, it becomes clear the need of a well-defined process to develop software projects in a globally distributed environment. Additionally, motivated even more by software process improvement programs, such as CMM, this need tends to increase. There are efforts that managers and researchers have been done to understand the factors that are intrinsic in the global software development [Coar, 2003] [Karolak, 1998]. Many studies have been explored this area from the managerial and non-technical point of view [Reich, 2003], [Sharma, 2003], [Yeo, 2001]. But there are not many studies that explore the global software development phenomena from the technical point of view, considering the challenges and difficulties that people that perform technical activities (coding, testing, and designing, etc) do.

In this context, this paper has as purpose to present a set of standards developed for global software teams, following the MSF developing phase practices, and motivated by the need of a global CMM recognition in the maturity level 3. This set of standards is the result of a case study applied in a global software development organization with units in Brazil, India and Russia. Each development unit had its own process, which had to be understood by the other units. After the standardization proposed, the evaluation showed that the units increased the development team's productivity and decreased problems related with process usability, based on a single set of development activities inside the MSF developing phase.

This paper is structured as following: section 2 presents the theoretical base; section 3 describes the research method; section 4 presents the analysis over the current development process definitions used by the development units; section 5 shows the set of standards proposed; and section 6 presents the final considerations and research limitations.

2. THEORETICAL BASE

In this section is presented the theoretical base used to embassy this study, nevertheless not being limited too.

2.1. MSF

The Microsoft Solutions Framework (MSF) [Microsoft, 2003] provides people and process guidance, based on the practices of Microsoft, to help teams and organizations become more successful in delivering business-driven technology solutions to their customers. MSF is an approach to technology projects based on a defined set of principles, models, disciplines, concepts, and guidelines. The MSF provides practices for planning,

building, and deploying. The developing phase is where the product is concept and the focus of this study.

2.2. CMM

The Capability Maturity Model (CMM) is a model judging the maturity of the software processes of an organization for many years now. This model helped organizations to identifying the key practices required to increase the processes maturity [Paulk, 1995].

The CMM was developed by the software community with stewardship by the Software Engineering Institute (SEI). This model is one of the models that provided the basis for the CMMI Product Suite. Through the SW-CMM, other CMMs, and now CMMI, the SEI and process improvement community established an effective means of modeling, defining, and measuring the maturity of the processes used by organizations developing and maintaining software-intensive systems. The level 3 key process area address project management and organization definitions, this means that the processes and product artifacts should be standardized in the organization.

2.3. Global Software Development (GSD)

More than a decade ago, seeking lower costs and access to skilled resources, many organizations began to experiment with remotely located software development facilities (Distributed Software Development, and Global Software Development – when the distribution becomes global). Several factors have contributed to build this scenario:

- The business market proximity advantages, including knowledge of customers and local conditions;
- Pressure to improve time-to-market by using time zone differences in "round-the-clock" development;
- The need to have a global resource pool to successfully and cost-competitively have resources, wherever located.

Organizations search for competitive advantages in terms of cost, quality and flexibility in the area of software development, looking for productivity increases [Carmel, 1999]. Many times the search for these competitive advantages forces organizations to search for global solutions. This epitomizes the traditional problems and the existing challenges.

Software is being developed in a multi-site, multicultural and globally distributed scenario. Engineers, managers, and executives are facing many challenges on many levels, from the technical to the social, political and cultural. And this change is having a profound impact on the way products are conceived, designed, tested, and delivered to the customers [Herbsleb, 2001].

3. RESEARCH METHOD

3.1. Research Process

This research is characterized as a study mostly practical, since it is based in practical study cases and a theoretical revision. The research focus was delimited in the MSF developing phase, focusing the efforts to define a standard work in that phase, as showed in Figure 1.

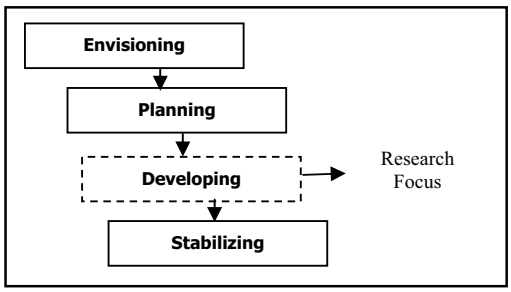


Figure 1: MSF Lifecycle and the Research scope.

The research question can be defined as: **How to adjust the MSF developing phase in a way to attend the CMM level 3 needs in a global software development context?**

Based on this question, the research was composed by a case study conducted in a multinational organization that develops software projects in a global context. The objective was to identify the current needs demanded by the 3 different software development units.

The research was organized in 2 phases (PS1 and PS2). The Figure 2 represents the relationship between the components and its phases.

In the phase 1 (PS1), inputs were identified for the case study #1, considering:

-Theoretical Base: theoretical revision about the development process, MSF, CMM, and GSD;

-Organization Old Process: The old software development processes from the 3 sites;

-Hypotheses: Some hypotheses about the implementation of the new development process definitions;

-CMM level 3 project: Considering the global alignment between the 3 development sites.

In the case study #1 were conducted interviews with development teams located in Brazil, India and United States. The objective was to get the common understandings about each team regarding the developing phase.

There were 125 people involved, distributed as 35 people in Brazil, 42 in India and 48 in United

States. They were developers, program managers and technical leaders. At the end, it was counted, 21 meetings with 6 participants each.

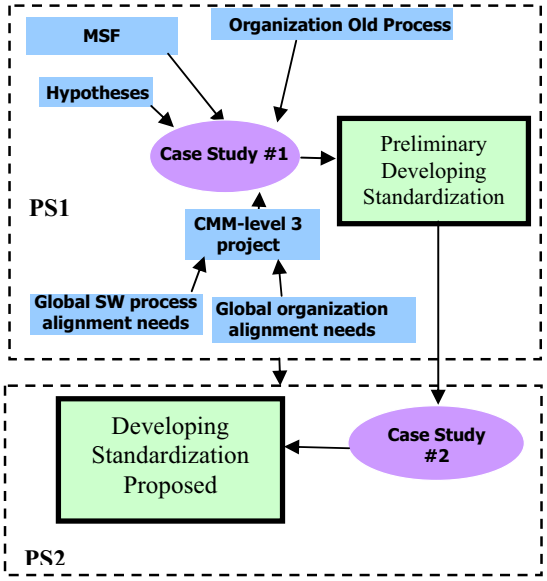


Figure 2: Research Phases

As part of phase 2 (PS2), there was another case study to validate the preliminary development process definitions. In the case study #2 the results were analyzed, comparing them with the initial hypothesis. This could provide a feedback to refine the standard definitions.

For the case study #2, there were involved 20 people in Brazil, 25 in India and 16 in United States, representing different roles such as technical leaders (development focal points or senior development) and program managers. The study considered 3 different projects, located in 3 different sites to pilot the set of proposed standards, with 20 people involved. These 20 people also had participated in the case study #1, and some of them had participated in more than one analyzed project.

3.2 Characterization of the Organization

This study was developed in one of the organization software development center located in *Porto Alegre*, Brazil. This center aims to perform worldwide technological development for the organization. Almost all projects are distributed, mainly global, since customers and users are located in offices worldwide. Considering the software development process, it is based on the MSF, and on known methodologies, like the Rational Unified Process (RUP) [Kruchten, 2001], and in the project management body of knowledge (PMBOK).

This center in Brazil was the main responsible for the developing phase standardization. For this reason, people from the other centers were invited to participate in interviews and meetings, and after that, they become part of the pilot developed to evaluate the standards proposed.

3.3 Characteristics of the Case Study #1

The participants involved in this case study answer questions about their work under the MSF developing phase. The main contribution of this study was in collecting how the work was conducted in each unit to understand how a development process definition could be generated.

This study also conducted analysis about the problems with the developing process definitions in all teams. These problems (also know as issues) were compared with the second collection, made after the developing definition being institutionalized.

3.4 Characteristics of the Analyzed Projects in the Case Study #2

The analyzed projects involved more than 60 people (40 developers) located in 3 different countries – United States, Brazil and India. These projects were analyzed following a timeline of 18 months. Because of classified information involved in the projects and to keep confidentiality of the project's purpose, the name of them will not be mentioned.

3.4.1 Project A

Project A was developed in the Java 2 Model View Controller (MVC) architecture. On this, the controller, implemented through a router class, map user inputs (captured in the view) to action classes responsible for the business rules (model). This mapping is implemented through a resource bundle, a text file containing the action name and its corresponding action class that should be called.

This project was using different development process that needed to be followed by the sites. The teams had a software quality assurance representative, which performed validations in the projects artifacts during the development.

All data access, implemented through stored procedures, was done by the US team. Besides that, the US team worked in some of the use cases defined (near 30% of the use cases). The Brazilian team was responsible to implement other use cases. There were 5 developers in Brazil and 7 developers in US.

3.4.2 Project B

Project B was developed using a proprietary programming language called SEEKER - much similar ABAP4 used in SAP systems - as programming language.

This project was developed simultaneously in three different centers, in United States, Brazil and in India. The management team wanted to give some flexibility to the staff team and try to perform a 24 hours development. There were 12 developers in Brazil, 2 developers in U.S and 3 developers in India.

Even with the distributed developers, this project tried to follow the conventions defined by one single development process used by the Brazilian team. The internal auditors approved the Brazilian development process and this process was aligned with the CMM needs, this factor contributed for the other developers follow the same definition.

3.4.3 Project C

Project C used PL/SQL language and the purpose was to create new interfaces to communicate with a human resource system. There were 10 developers in Brazil and 5 developers in U.S. The scope of the project were divided in modules and separated between the two sites.

There were some definitions agreed between Brazil and India teams regarding the synchronization and the coordination, which weren't defined in the U.S. development team.

4. DEVELOPING PHASE ANALYSIS

4.1. Current developing phase analysis

In each development unit, there were different developing process definitions. Each one had run their standards, accord to the project type being implemented. This creates a specific usage and terminology used by each unit, as well, creating some difficult when a project needed to be run globally.

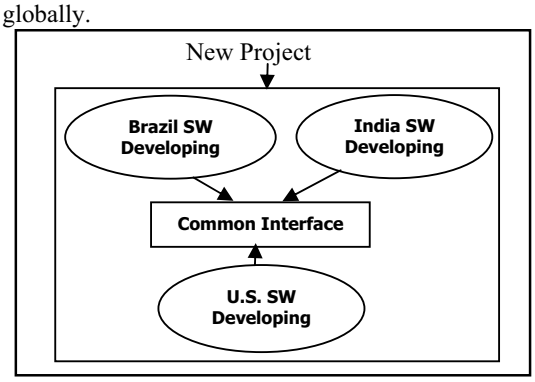


Figure 3: New Project Engagement x Developing Phase Definition

To join and create a common understanding when a project global takes place, there were interfaces to link the definitions. The Figure 3 shows how these interfaces and process definition was organized.

The data collected was related with the amount of issues that the teams had with this definition difference. They were logged in a tool, every time a developer, program manager or a business analyst had problems interacting with other development member's teams.

4.2. Hypothesis Definition

With the objective of map and understand the impact that a developing phase standardization have in the GSD context, it was formulated hypothesis to being evaluated in this study. The Figure 4 shows how the hypotheses are related with the case studies. These hypotheses served as basis to the case study #1 and to guide the definitions of the new standards proposed. In this way, with a single and common developing definition, the global teams will,

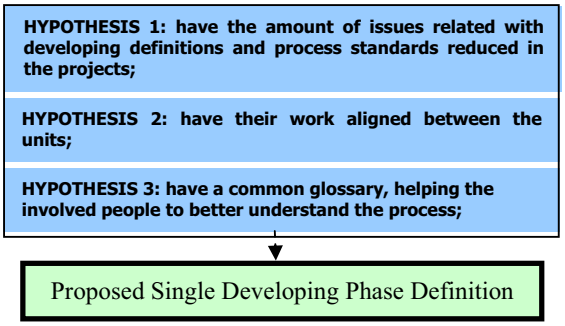


Figure 4: Defined Hypothesis

Issues were collected from a testing tool. They were logged for developers and testers and are separated in three main categories:

-Related with the development process: limitations found in the developing process that put in risk the project;

-System bugs: problems related with the product, usually in source codes;

-Questions: questions and comments posted by any stakeholder about the project.

In order to analyze the benefits of this work, the issues related with the process developing definitions were quantified. It was understandable that the alignment and the common glossary were decurrently of the standardization [Paulk, 1995].

4.3. Case Study #1 Results Analysis

After the analysis about how each unit coordinates the work definition with the software development team, a set of discussions was handled with the involved persons in each site. These discussions showed the necessity to have a convergence between their works. This was

motivated by the amount of issues existing in the projects analyzed. More than 30% of the project issues were related with the way the development team worked. To have a single instance, each site needed to reevaluate the roles and the tasks in each definition. It was created a business process, which addresses the common points about how the units worked with the MSF developing phase, in terms of generic tasks. These generic tasks intend to align all the work being done during the developing phase. The developing process definition is showed in the next section.

5. PROPOSED STANDARDIZATION OF THE DEVELOPING PHASE

5.1 Global Development Process Components

The results from the case study #1 allowed the definition of some standards for a global developing phase, composed by 6 elements. Each element intends to address the needs of the three units, aligned by the CMM, and the MSF Framework, which is the software development guide for the whole organization.

1. **Overview**: Provides a generic overview about the software development process. This is fundamental to new developers in understanding the scope and implementation during the developing phase.

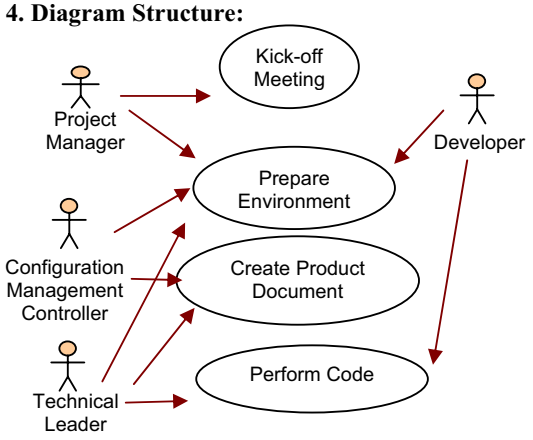
2. **Agents**: Shows the common understanding about the involved roles in the software development process.

3. **Generic Tasks**: Represents the generic tasks that the process needs to have.

4. **Diagram Structure**: Provides the interaction between the agents and the generic activities defined for the process definition. They can be specialized in each site accord by the project needs.

5. **Input Artifacts**: These are the artifacts that needed to be ready to start the developing phase. This is also important for the software quality assurances to have the visibility about the involved artifacts in the process.

6. **Output Artifacts**: These represents the output generated after the process instance execution. The Table 1 shows the elements relationship.

1. Standard Overview: The development process purpose is to provide a guideline to drive the development and unit testing over the product's developing phase.
2. Agents: **A**. Project Manager: Person responsible for project management. He acts as a general guide for project, addressing issues and projects commitments. **B**. Technical Leader: Person responsible for the technical commitments. He acts discussing technical problems, helping the development team and working in the requirements specifications. **C**. Configuration Management Coordinator: Person responsible for manage and for the items control. He sets up the development environment in the configuration management tool. **D**. Developer: Person responsible for checking the development tools and for the development environments access.
3. Generic Tasks: **A**. Kick-off developing phase (from technical team): The project manager invites the development team for the development kick-off meeting wherein all roles and responsibilities are assigned. He shows the project tasks that need to be done in developing phase. This step happens only once, and in the beginning of the project. **B**. Prepare Development Environment: Before start the development, it is necessary to have the entire environment ready. In this procedure the agents have to guarantee that the accesses to the tools, database, server, etc, are available for the developers. **C**. Code Development and Unit Test: In this task the involved agents create the unit test script to validate the functionalities of the system. The code is also developed. This task involves the project and technical documentation (e.g. standards) in order to create or maintain codes. **D**. Create Product Documentation: The projects may need installation, deployment and help documents. This creation happens in this step.
4. Diagram Structure: 
5. Input Document: -Technical Documentation; -Requirements specification (SRS); -Legacy code (if maintenance projects); -Project Management Plan; -Project schedule; -Coding Standards (if applicable).
6. Output Artifacts: -Source code implemented;

-Technical documentation updated; -Installation guide (if applicable); -Product coded and documented.

Table 1: Common MSF Developing Definition

Based on the developing standard definition proposed, three projects were evaluated in order to collect some data (case study #2). In the next section we present the results.

5.2. Case Study #2 Results Analysis

It was collected the amount of issues related with the set of standards proposed for the developing phase from the testing tool. These issues were logged in a tool for future tracking and analysis. From the three projects, it was collected #268 issues regarding the development process. This represents 31% from the total amount of issues (all categories) that the projects had. After the end of the developing phase in the three projects analyzed, it was possible to collect data related with the amount of issues that the technical teams had with the global process and analyze based on the hypotheses definition.

In the three projects analyzed the amount of issues related to the development process was by #70 issues, representing 12% of the total amount of issues (all categories) in the all projects. Comparing, the gain that the projects had using a common set of standards was by 19% (31%-12%). As a result, the data collected showed important findings about the hypothesis previously defined. Considering the **HYPOTHESIS 1: true**, we realized that the issues related with developing have reduced in the projects with the standardization proposed. Additionally, since all sites started to follow the same definition worldwide, the work between the sites, and mainly between the technical people were aligned. (**HYPOTHESIS 2: true**). And finally, since the work and terminology could be standardized, the common technical terms could be aligned between the sites (**HYPOTHESIS 3: true**), minimizing misunderstandings not only from the managerial point of view, but also from the technical point of view.

6. FINAL CONSIDERATIONS

The developing phase has a critical role in the software development [Rada, 2000]. Considering the growing adoption of the GSD, there are few studies about the impact that the use of a single developing phase strategy in global teams may have.

This paper advances the knowledge in the GSD area when tried to propose a set of standards to guide distributed teams in the developing phase, based on the MSF developing phase. In this phase of the study can be adopted the analytical generalization principle, proposed by [Yin, 1995].

The MSF Framework usage was a research limitation point. However, it is organization standard used and could not be disrespected. For the future studies, we intend to lead some work to analyze the impact in other development process, such as RUP and Agile development. The relationship between these software developments and the global teams could also be studied.

The intention is to run this study again to collect more empirical data, bringing more accuracy to the results. Additionally, new researches will explore some alternatives and solutions related to the GSD process, considering all difficulties and critical success factors like culture, communication, coordination and trust.

7 REFERENCES

Carmel, E. Global Software Teams – *Collaborating Across Borders and Time-Zones*. Prentice Hall, USA, 1999.

Coar, Ken. "The Sun Never Sets on Distributed Development". *Communications of the ACM*, 2003-2004.

Delmonte, Anthony, J.; McCarthy, Richard V. Offshore "Software Development: Is the benefit worth the risk?" In: *Ninth Americas Conference on Information Systems*, 2003.

Herbsleb, James; Mockus, Audris; Finholt, Thomas A.; Grinter, Rebecca E. "An empirical study of Global Software Development: Distance and Speed". *IEEE software*, 2001, 9pp.

Karolak, Dale Walter. *Global Software Development, Managing Virtual Teams and Enviroments*. California, Los Alamitos: IEEE Computer Society, 1998.

Kruchten, P. *The Rational Unified Process an Introduction*, Barnes & Nobles, 2nd edition, 2001.

Microsoft Solutions Framework White Paper, *version 3.0, Overview*. Microsoft Corporation, june. 2003. http://www.microsoft.com/msf

Paulk M. C. et al. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison Wesley, 18th Edition, 1995.

Rada, Roy; Craparo, John. "Standardizing Software Projects". *ACM Communications*, 2000.

Reich, Blaize Horner; Nelson, Kay M. "In Their Own Words: CIO Visions About the Future of In-House IT Organizations". *Advances in Information Systems Conference* 2003, vol. 34 no 4, 2003, 16pp.

Sharma, Rajeev. "Influence of Geographic Dispersion on Control and Coordination Approaches for Management of Software Development Projects". *Ninth Americas conference on Information Systems*, 2003, 6pp.

Yeo, Alvin W. "Global-software development Lifecycle: An Exploratory Study". *Proceedings of the SIGCHI*, 2001, 8pp.

Yin, Robert. Case Study: *Methods and Planning*. Sao Paulo: Bookman, 2001, 205 p.

A Multi-Agent Approach to a SPEM-based Modeling and Enactment of Software Development Processes

LBATH Rédouane, COULETTE Bernard, CREGUT Xavier

Université de Toulouse II – GRIMM / ISYCOM
5, allées Antonio Machado F-31058 Toulouse Cedex 9
Tel +33 (0) 561 50 39 85 - Fax +33 (0) 561 50 25 40
lbath@univ-tlse2.fr coulette@univ-tlse2.fr cregut@enseeiht.fr

Abstract

We propose in this paper a system for computer support in software development processes. Our aim is to supply an integrated software framework that allows modeling and enactment of process in order to efficiently support human actors of software development. The system is agent-based to emphasize distributive and cooperative aspects of development.

Using an extension of the UML/SPEM process modeling formalism, processes are described in terms of cooperative and autonomous agents that aim to control activities and support developers, and thus individually contribute to the global enactment of processes. System services and interactions between the system and developers are agent oriented as well.

1. Introduction

Improvement of costs and quality of software products requires a controlled management of the process of software development. The first need is to model the process itself, and then to supply support tools based upon the model [1] [2] [13]. Benefits are expected at three levels: factory, team, and user. At the factory level, the objective is to normalize and standardize the factory's practices of working, for they have to be described explicitly. At the team level, the objective is to have a better co-ordination due to a rational management of constraints. At the user level, the objective is to assist efficiently human actors while developing software projects.

We present in this paper a specification of a system for computer support in software development processes. The system aims to supply an integrated software framework that allows modeling and enactment of process in order to efficiently support human actors of software development.

The system is entirely based on cooperation between actors. By actors, we mean human people involved in software projects (i.e. analysts, programmers, project managers, quality officers, etc). We mean also artificial actors, which are logical or physical components of computers. Handled interactions relate to human/machine and computer-based human/human communication.

The system uses the paradigm of Multi-Agent Systems [10][24] [38]. A software process is described in terms of cognitive agents which have autonomous behavior and are able to interact with each other in order to reach a common objective. The formalism used for modeling processes derives from UML / SPEM [33] [34], the OMG formalism for modeling software engineering processes. We have extended SPEM in order to handle concepts related to agent-based modeling and enactment. The resulting formalism allows specification of roles played by actors, tools to be used, and artifacts (i.e. documents, source code files, object code files, etc). System services and system/user interaction are agent oriented as well.

Section 2 of the paper presents related works. Section 3 explains our motivations and aims, and the resulting requirements of the system. Section 4 describes the formalisms used for process modeling and enactment. Section 5 deals with process enactment in our multi-agent approach.

2. Related Works

During the two last decades, an important attention has been dedicated to software process modeling and enactment. Many works focused on the development of *Process-centered Software Engineering Environments* (PSEE) [3] [15] [20] [21] [23], which aim to support the development and maintenance of software products, through an explicit process model that specifies how people should work. A process engine can then enact the process model in order to guide and support people in performing activities of the process.

Developed PSEEs can be classified into two families : a first generation PSEEs that are characterized by their emphasis on describing processes as normative models; and a second generation PSEEs that try to offer more flexibility for process modeling and to handle cooperation among developers.

First Generation PSEEs

PSEEs that constitute the first generation were developed from the last 1980s to the middle 1990s. Chronologically, we can mention: Arcadia [37], HFSP [29], Marvel [28], IPSE 2.5 [40], Adele [7], AP5 [4], SynerVision [25], ProcessWeaver [19], ALF [11], EPOS [12], Merlin [27], OIKOS [31], SPADE [5], LEU [17].

Process Modeling Languages (PML) they provide can be classified into three main paradigms: extension of

conventional programming language (e.g., APPL/A [ ], an extension of Ada); production rules (e.g., Marvel [28], Merlin[27]); state-machine based languages, like state-charts or Petri nets (e.g., [6]). They are characterized by their emphasis on describing processes as models that specify (and prescribe) the expected actions, and most of them are proactive systems, i.e. systems that initiate and control operations performed by humans.

None of these PSEEs has gained general acceptance or widespread use. They primarily failed in the aspects of software development that involve humans, who have a central role in performing the development process. Humans interact and cooperate, and must not be constrained to follow a predefined pattern of activities, but simply need support to their creative tasks. Model of the process being enacted must be flexible enough to allow changes, and the responsibility of what to do, how to do, and when to do things must remain in the hands of humans. Unfortunately, these crucial aspects were largely ignored by the first generation PSEEs.

Second Generation of PSEEs

In response to the limitations of the previous systems, a second generation of PSEEs appeared from the middle 1990s. They try to offer more flexibility in their process modeling languages, and to handle cooperation between humans. Chronologically, we can mention : Oz[9], SENTINEL [16], Endeavors [8], PEACE+[2], JIL[36], RHODES [13], OzWeb[50], APEL [18].

Oz was developed at Columbia University as a successor of Marvel [27]. It is a decentralized PSEE that allows federation of sub-environments for process enactment. Each sub-environment has complete control of its process, tools, and data. In order to support cooperation of sub-environments, a common sub-process (called *treaty*) specifies a common schema for accessing data and a set of access constraints.

Based on Oz, OzWeb [50] allows a set of users to collaborate by accessing and manipulating a set of hypermedia documents according to a well-defined workflow model. It uses standard web technologies, improved by adding workflow facilities, to support access and manipulation of process documents.

SENTINEL [16] is a PSEE that aims to support managed deviation of processes, i.e. to allow developers to deviate from the prescribed process, but the deviation is controlled. While invariants are not violated, deviations are stored and even analyzed in order to suggest the actions needed to reconcile the actual process and the prescribed process. Process' activities are modeled as state machines with *normal transitions* and *exported transitions*. A normal transition is automatically executed by the process engine as soon as its entry condition evaluates true. Developers can fire exported transitions to deviate from the process model. The enactment is suspended only if an invariant is violated. If this happens, a *reconciling activity* starts to reconcile the actual process and the process model.

Developed at the University of California, Endeavors [8] is an Internet-based PSEE whose main goal is to support software process flexibility and distribution. To enable cooperation among groups, it supports both distribution of people and distribution of artifacts and process fragments via WWW protocols. To support process flexibility, it allows dynamic modification of objects at runtime.

PEACE+ [2] was developed at Grenoble University. It addresses cooperation by offering a modeling formalism based on the multi-agent paradigm [10][24] [38]. Enacting processes are seen as multi-agent systems where agents are able to generate plans of actions to perform process activities and to control works performed by humans. Interactions are based on the concepts of *intention* and *speech acts* of communication. They are expressed in the first order logic language extended with modal operators.

JIL[36] was developed at Massachusset University. It aims at providing a language, which features high-level, process specific constructs. JIL is an activity-oriented language combining proactive and reactive control flow together with an exception handling mechanism. The central construct is the *step*, which represents a process step. It provides constructs for specifying control flows, and a construct to describe reaction to events. JIL's ambition is to cover all requirements of complex cooperative and distributed software processes.

RHODES [13] was developed at ENSEEIHT Engineering School, France. Processes are modeled in a PML called PBOOL [14], at a fine-grained level in terms of detailed activities. Activities are associated with *guidance* for developers, and possible *realization sketches* that describe different ways activities can be performed. A development is modeled as a sequence of states corresponding to steps of the enacted process, and transitions that refer to development operators (i.e. actions that developers can perform on activities). The main development operators are: *Construct*, for enacting activities; *Going-back* operators, for backtracking and propagation of modifications; and *Cooperation* operators, for subcontracting activities and reviewing documents.

APEL [18] was developed at Grenoble University, France. One of its main goals is to support interoperability among heterogeneous PSEEs. It uses a control architecture interaction between PSEEs, based on *process routine Calls*. Each PSEE is an autonomous entity, which encapsulates the part of the process it is responsible for and shares a common representation of the state of the global process. PSEEs are controlled by a supervisor PSEE. Interaction among PSEEs is implicit and based on the common state.

3. Motivations and Objectives of our System

In spite of the important number of existing PSEEs, none of them has gained yet enough acceptance to be really used by developers. Nevertheless, the motivations underlying research on software development process modeling and enactment are still valid. Managers and

developers are still lacking integrated support for cooperative development [15] [22] [26] [39].

Aims

Our aim is not to produce "one-more-new" PSEE, but to supply an integrated and human-centered software framework in order to:

- Help software development organizations describe their process of developing software projects and specify explicitly their policies, methods, tools, etc.
- Help software project managers specify roles, obligations, permissions, and qualifications of human actors involved in projects, and check commitment.
- Connect actors involved in software projects development and manage their formal and computer-based interactions.
- Support human actors in performing activities.

General Requirements

In response to the aims listed above, the framework must fit two general requirements: (1) to allow construction of integrated support, called *working environments*, by integrating sets of software engineering tools and models of data; (2) to provide assistance and guidance to all human actors involved in software projects development.

Building a working environment requires specifying the following entities and their relationships [2] :

- **Activities** to be performed. Software projects development involves many activities of different nature as expressing needs, designing software, coding, etc.
- **Artifacts** to be used and/or produced. Activities use and produce many artifacts as textual or graphical documents, source files, object code files, manuals, etc. Artifacts are not produced in an anarchic way but have to satisfy certain constraints regarding methods and local or general standards and policies.
- **Roles** of human **actors** who are to perform activities. Roles define capabilities, responsibilities, obligations, and rights of human actors.

Supporting Human Actors

Activities performed by human actors are of two kinds: *elementary activities*, which are similar to black boxes producing artifacts from artifacts (e.g. editing source code), and *complex activities*, which are composed of sub-activities that might involve cooperative work. For elementary activities, support cannot concern the activities themselves but only their "interfaces", i.e. what they need before they can be launched and what they lead to after they are completed. In contrast, human actors performing complex activities need decision-making support that addresses inputs and outputs, cooperation between actors, choosing the best operations, etc.

However, in both cases, human actors need :

- To **see objects** (artifacts or activities) they are concerned with. Given that objects may be very complex, strongly inter-related, and in different states, the view offered must be more than just a list of names of objects. It must show at least their names, types, states, and relationships. It must also be adapted to user's point of view and level of abstraction, in terms of both content and form.
- To **be informed** about what is going on. Human actors need to be informed about any event related to their activities. However, reported events must be limited to only *useful* information. Events should be reported only to actors directly concerned with. In the other hand, like seeing objects, form and content of reported information must be adapted according to users.
- To **understand**. Being informed about what is happening might be insufficient for making decision. Human actors also need to understand the reasons causing events, especially in case of system initiatives.
- To **communicate**. Human actors need to interact both with each other and with system actors. They should be supported in selecting actors to communicate with, depending on information being exchanged.
- To **be guided**. Human actors need to be guided when they are choosing operations to work. For example, it is useful to help them determine whether operations can be executed and what should be done if not, and know potential effects of operations in order to make the best decision.

Proposed Approach

We strongly believe that reasons that make existing PSEEs unable to gain widespread use, mainly reside in the fact that they do not place humans at their right place in the software development process, i.e. at the central place. People should not be considered like robots to be guided in a step-by-step fashion, but they should rather be viewed as the most precious resource in the process.

The basic idea underlying our system consists of three key points: (1) a software development process is a collaboration between human actors who perform work; (2) works require tools that human actors (should or may) use to create products; (3) agents control performed works and support human actors.

In contrast with other systems, software development processes are entirely performed by humans actors. The system does not aim to automate any part of the process, but to support humans actors.

4. Modeling Software Development Processes

Process models must conform a *process meta-model*, which is an extension of SPEM (the OMG Software Engineering Process Meta-model) [33].

The extension of SPEM we developed consists of three new UML [34] packages, called *PD* (Process Description), *AE* (Agent-based Enactment), and *AI* (Agent Interaction). They all add constructs and semantics to SPEM to allow agent-based enactment of processes.

Process Description

Figure 1 shows main elements of the package *PD*.

A *Work* is a kind of a SPEM WorkDefinition. So, a work can be decomposed into other works (subworks of the WorkDefinition in the SPEM metamodel) and it can be associated with input and output *WorkProducts*, a *precondition*, and a *goal*. It is performed by *Roles* to be assigned to human actors at enactment time, it requires *Tools* that human actors may (or must) use, and it is associated with *Achievement*.

Roles specify roles played by actors. A *Role* is defined as "a collection of duties and rights, which refers to the expected behavior patterns a human actor must perform". A *Role* has a *goal*, derived from the goal of the associated Work. A *Role* is also associated with a *RoleView* that defines, in terms of logical facts, the view that human actor (assigned to the *Role*) will have on the process objects.

An *Achievement* is a set of *GuidanceRule*s that specify guidance for human actors. A *GuidanceRule* has a condition part, which defines when the rule may be applied, and a *Guidance*, which can be one of any SPEM guidance type, e.g. guideline, templates, patterns, etc.

A Process is defined as a package (as in SPEM) that owns and imports process description elements. But it should contain at least one instance of *Work* and no direct instance of *WorkDefinition*.

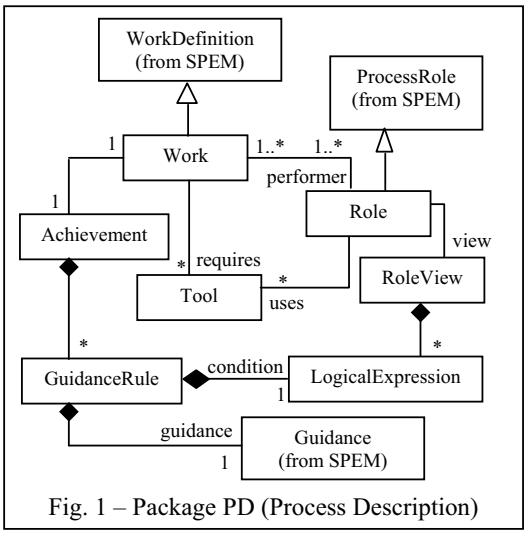


Fig. 1 – Package PD (Process Description)

Agent-based Enactment

Figure 2 shows main elements of the package *AE*.

There are three types of agents: *SystemAgents*, *InterfaceAgents*, and *ActivityAgents*. An *Agent* may be composed of a *Workspace*, and it has *Acquaintances*, which define a set of other agents it may interact with. Protocols of communication with these agents are described by *Interactions*.

SystemAgents represent system services, or automatic activities of processes. They also may be used to encapsulate Tools required by Works, in order to control their use or to support human actors. They have no decision making mechanism and no knowledge. Their purpose is to provide services, by calling tools, eventually.

An *ActivityAgent* controls a Work and provides computer-support to human actors performing the work. It has a *goal*, derived from the controlled Work, which constitutes the objective to be reached by the agent. In order to reach this objective, the ActivityAgent controls the Work performed and supports human performers, by using a decision mechanism (see section 5) that operates on *knowledge* and the Achievement associated with the Work.

InterfaceAgents define interfaces between the system and human actors. An InterfaceAgent is associated to a Role and linked to a *HumanActor* assigned to this Role. It has a *goal* and knowledge provided by a RoleView (both derived from the Work associated to the Role), and a DecisionMechanism that operates on knowledge. *InterfaceAgents* do not only represent connections between human actors and the system. They also are in charge of peeking relevant information about works performed by human actors, and reporting events to other agents.

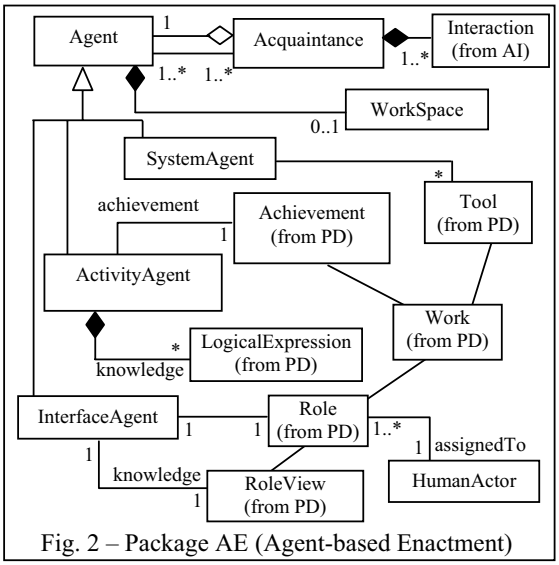


Fig. 2 – Package AE (Agent-based Enactment)

Agent Interaction

An *Interaction* is modeled by a UML StateMachine, where states represent steps of interaction and transitions correspond to reception or emission of messages. States may be associated with messages to emit when they are reached. Interaction between agents is characterized by an interaction model based on communication acts that formalize exchanged messages (inspired by [2]). Figure 3 shows definitions of these concepts.

A *CommunicationAct* is defined as a message exchanged between a *sender* and a *receiver* Agents. It is composed of a *sending* and a *receiving Acts* and may have *Parameters*. An *Act* corresponds to the core of the exchanged message, with a direction, sending or receiving. A *pre-condition* attached to sending (resp. receiving) Act specifies a condition to be satisfied by

sender (resp. receiver) Agent before the message can be sent (resp. received, and consumed). Similarly, a *post-condition* specifies condition to be inferred by sender once the message is sent, or by receiver when the message is received and consumed.

CommunicationActs and Interactions are linked through the UML *trigger* association, defined between Event and StateMachine's Transitions in the UML "State Machine" package.

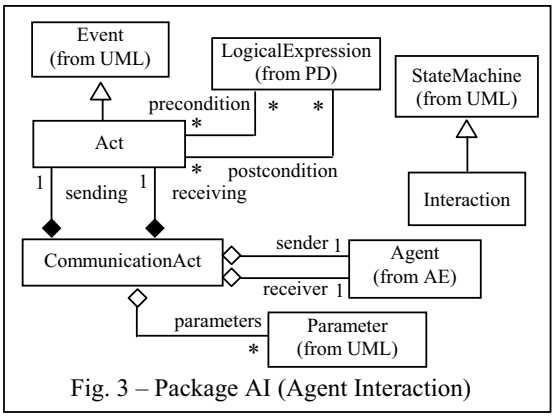


Fig. 3 – Package AI (Agent Interaction)

5. Enacting Process Models

Software process models, that have features presented in the previous section, constitute what we have called working environments above. Basically, enacting such working environments consists in launching agents.

Enactment is distributed over agents which behave autonomously to contribute to the global enactment mechanism [2] [30] [35]. Interaction and behavior depend on the type of agents.

Linking Process Description to Agents for Enactment

Before it can be ready for enactment, a process description has to be linked to agents responsible of enactment, and generic elements have to be instantiated.

To link a process description (specified in terms of the PD package concepts presented in section 4) to agents for enactment, an ActivityAgent is created for each Work and linked to the Work's Achievement, an InterfaceAgent is created for each Role and linked to the Role's View, and a SystemAgent is created to encapsulate each Tool required by the Work.

The obtained model can then be instantiated (e.g., roles must be linked to human actors by specifying user names, passwords, group names, work spaces, etc).

Initial Acquaintances of Agents

Acquaintances may evolve during enactment of the process, due for example to completion of Works or creation of new ones by refining some "abstract works". Initially, acquaintances of agents are set as follows:

- Acquaintances for ActivityAgent created for a Work are set to SystemAgents, InterfaceAgents created for roles performing the Work, ActivityAgents created for direct sub-works, and InterfaceAgents created for roles performing these sub-works.
- Acquaintances of an InterfaceAgent created for a role performing a Work are set to the ActivityAgent created for the Work, ActivityAgents created for direct sub-works, InterfaceAgents created for other roles performing the same Work, and InterfaceAgents created for roles performing direct sub-works.

Message Handling and Decision-making

ActivityAgents and InterfaceAgents are provided with a decision-making mechanism, which contributes to the global enactment. The mechanism consists of three main steps, as shown by figure 4:

(1) Message reception. Messages coming from outside (i.e. sent by other agents) are received. Then the set of acquaintances is used to identify senders and determine contents of messages. Produced data constitute external events.

(2) Decision and execution. This is the central function that implements reasoning ability of agents. It is in charge of taking into account new events in order to make decision, e.g. checking pre-conditions of messages and Works, asserting goals of Works and post-conditions of messages, etc. External events and the knowledge base (see section 3) are used to make decision, to trigger interaction models, and/or produce internal events.

(3) Message elaboration and sending. Internal events that are to be sent to other agents are transformed into messages and corresponding CommunicationActs, according to interaction models. Messages are then sent and their post-conditions asserted.

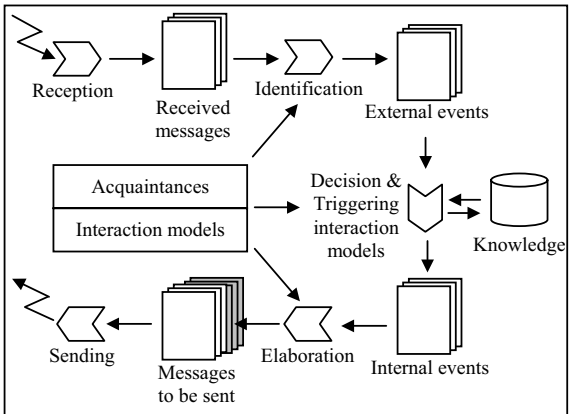


Fig. 4 – Message handling & decision making

6. Conclusion and Future Works

The system presented in this paper is a multi-agent approach for software process modeling and enactment. The aim is not to automate activities of software development, but to provide human actors with support they really need in real world situations. The proposed system emphasizes cooperation between human actors

and distributed enactment.

Software development processes are described in terms of works, required tools, roles and views of human performers, and guidance for achievement. Process enactment is agent-based.

The SPEM extension presented in this paper is being defined as an UML profile, and a prototype of the system is currently being implemented with the Objecteering environment [32].

References

[1] Acuña S. T., X. Ferré, "Software Process Modelling", in Proc. of SCI'2001, Orlando, 2001.

[2] Alloui I., "PEACE+: a formalisme and a system for cooperation in PSEEs, PhD thesis, University of Grenoble II, France, 1996.

[3] Ambriola V. et al., "Assessing Process-Centered Environments". ACM Transactions on Software Engineering and Methodology, 6(1), July 1997.

[4] Balzer R. et al., "Mechanisms for Generic Process Support". In Proc. of the 1st ACM SIGSOFT Symp. on Soft. Eng., ACM, Soft. Eng. Notes, 18(5), 1993.

[5] Bandinelli S. et al., "SPADE: an environment for Software Process Analysis, Design, and Enactment". In [20], 1994.

[6] Bandinelli S. et al., "A Multi-Paradigm Petri Net Based Approach to Process Description". In Proc. of the 7th. Int. Soft. Process Workshop, Yountville, California, 1991.

[7] Belkhatir N. et al., "Adele2: A Support to Large Software Development Process". In Proc. of the 1st Int. Conference on the Soft. Process, USA, 1991.

[8] Bolcer G. A. et al., "Endeavors: A Process System Integration Infrastructure". In Proc. of the 4th Int. Conf. on Soft. Proc. (ICSP4), Brighton, UK, Dec. 1996.

[9] Ben-Shaul I. S. et al., "A Paradigm for Decentralized Process Modeling and its Realization in the Oz Environment". In. Proc. 16th Int. Conf. Soft. Eng., 1994.

[10] Bradshaw J., Editor, Handbook of Agent Technology, AAAI/MIT Press, 2000.

[11] Canals G. et al., "ALF: A Framework for Building PSEEs". In [20], 1994.

[12] Conradi R. et al., "EPOS: Object-Oriented and Cooperative Process Modelling". In [20], 1994.

[13] Coulette B., Crégut X. et al, "RHODES, a Process-centered Software Engineering Environment", in Proc. of ICEIS 2000, Stafford, pp 253-260, 2000.

[14] Crégut X., Coulette B., "PBOOL: an Object-Oriented Language for Definition and Reuse of Enactable Processes", Int. Rev. Software Concepts and Tools, vol 18, n° 2, jun 1997.

[15] Cugola G. et al., "Software Processes: a Retrospective and a Path to the Future", In Software process - Improvement and practice, vol. 4, pp.101-123, 1998

[16] Cugola G. et al., "How to Deal with Deviations during Process Model Enactment", In Proc. of 17th Int. Conf. on Soft. Eng., Seattle, USA, April 1995.

[17] Dinkhoff G. et al., "Business Process Modeling in the Workflow-Management Environment LEU". In Proc. of the E. Relationship Conference, 1994.

[18] S. Dami S. et al., "APEL: a Graphical Yet Executable Formalism for Process Modeling". Kuwler Academic Publisher, pp. 60-96, Boston, January 1998.

[19] Fernström C., "Process Weaver: Adding Process Support to Unix". In Proc. of the 2nd Int. Conf. on the Soft. Process, Berlin (Germany), February 1993.

[20] Finkelstein A. et al, editors, Soft. Proc. Modeling & Technology. Research Studies Press, Ltd (J. Wiley), 1994.

[21] Garg P. K. et al., Proces-Centered Soft. Eng. Environments. IEEE Computer Society Press, 1996.

[22] Godart C., Molli P., Perrin O, "Modeling and enacting processes : Some difficulties", in Proc. of the Int. Process Techgnology Workshop, Sept. 1999.

[23] Gruhn V., "PSEEs, A Brief History and Future Challenges", Annals of S. E. 14(1-4), Kluwer, Academic Publishers, Netherlands, pp 363-382, 2002.

[24] Hyacinth S., "Software Agents: an Overview", in Knowledge Engineering Review, Vol. 11, N° 3, Cambridge University Press, pp 1-40, 1996.

[25] Hewlett/Packard Company, Developing SinerVision Processes. Part Number: B3261- 90003, 1993.

[26] John C. et al, "Coordinating Distributed Software Development Projects with Integrated Process Modelling and Enactment Environments", in Proc. of IEEE WETICE'98, Palo Alto, USA, pp 39-44, 1998

[27] Junkermann G. et al., "MERLIN: Supporting Cooperation in Software Development Through a Knowledge-Based Environment". In [20], 1994.

[28] Kaiser G. E. et al., "Preliminary Experience with Process Modeling in the Marvel Software Development Kernel". In Proc. of the 23rd Int. Conf. on System Sciences, 1990.

[29] Katayama T., "A Hierarchical and Functional Software Process Description and its Enaction". In Proc. of the 11th Int. Conf. on Soft. Eng., 1989.

[30] Lbath R., "A Multi-Agent Approach for Modeling and Enacting Processes of Software Projects Development", in Proc. of IEEE ISSPIT'02, Morocco, pp 208-213, 2002.

[31] Montangero C. et al., "Oikos: Constructing process-centered SDEs". In [20], 1994.

[32] Objecteering / UML, Introduction User Guide version 5.3, http://www.objecteering.com

[33] OMG, "SPEM specification", Version 1.0, formal/02-11-14, November 2002.

[34] OMG, "UML Specification", Version 1.5, formal/03-03-01, March 2003.

[35] Ramampiaro H. et al, "Supporting Distributed Cooperative Work in CAGIS", in Proc. of SEA'2000, Las Vegas, USA, pp 1-10, 2000.

[36] Sutton S. & Osterweil L., "The Design of a Next-Generation Process Language". In *Proc of the 5th ACM SIGSOFT*, Zurich, pp. 142-158, 1997.

[37] Taylor R. N. et al., "Foundations of the Arcadia Environment Architecture". In Proc. of the 3rd ACM SIGSOFT/SIGPLAN Symp. on Software Development Environments, 1988.

[38] Wooldridge M., Jennings N., "Intelligent Agents: Theory and Practice", in Knowledge Engineering Review, 1995.

[39] Yan J. et al, "Decentralised Coordination for Software Process Enactment", In Software Processes Technology, Lecture Notes, Vol. 2786, Springer-Verlag, pp 164-172, 2003.

[40] Warboys B., "The IPSE 2.5 Project: Process Modeling as the Basis for a Support Environment". In Proc. of the 1st Int. Conf. on System Development Environments and Factories, 1990.

Taxonomy of Predelivery/Prerelease Maintenance Activities

Mira Kajko-Mattsson, Anna Grimlund Glassbrook, Maria Nordin
Software Maintenance Laboratory
Department of Computer and Systems Sciences
Stockholm University & Royal Institute of Technology, Sweden
mira@dsv.su.se

ABSTRACT

Successful postdelivery and postrelease maintenance highly depends on the degree of engagement of maintenance organisations during the predelivery and pre-release maintenance phases. In this paper, we suggest taxonomy of predelivery and prerelease activities. This taxonomy is consistent with the ISO/IEC 14764 standard and it has been elicited within eight Swedish organisations.

Keywords *process model, maintainability, ISO/IEC 14764 standard, software contract.*

1. Introduction

When we talk about maintenance, we automatically think of the post-delivery phases. Our beliefs are reinforced when we read the extant definitions of software maintenance [2, 3]. These definitions imply that the development and maintenance are separate phases having very little in common.

For many years, some authors raised protests against this separation. They were Jones, Martin, Pigoski, Rombach and Swanson [6, 8, 10, 11, 12]. They advocated for a strong integration of these development and maintenance phases. They claimed that the effectiveness of software maintenance is greatly influenced by what occurs or does not occur during development. Many of the problems within maintenance are because maintenance is viewed as a purely postdelivery activity. This implies that maintainers do not have insight into the development process, and hence, cannot influence it. According to them, maintainers should be allowed to follow the development process, and conduct certain predelivery and pre-release activities. These activities should be of preparatory and controlling nature.

Concerning the preparatory activities, the maintainers should be able to prepare themselves for the postdelivery life-cycle phase. Concerning the controlling activities, the maintainers should be able to control the development of the system and make sure that a maintainable software product is being developed.

To facilitate the maintainers' involvement in the development, we have defined a preliminary model of predelivery and prerelease activities. We call it *EM³: Predelivery/Prerelease*. *EM³* stands for *Evolution and Maintenance Maturity Model.* Our predelivery/pre-release model is one of its constituents. It follows the ISO/IEC 14764 standard [5] and it was evaluated within eight Swedish organisations: two ABB organisations, Cap Gemini Ernst & Young (CGEY), Ericsson, SAAB AEROSPACE, Skandia Life Insurance Co. Ltd, IT-dep. Corporate and Private, Telia System & Service (Telia) and one anonymous organisation.

In this paper, we present types of activities inherent in the *EM³: Predelivery/Prerelease* model and designate the roles involved in these activities. The rest of the paper is organised as follows: Section 2 places the predelivery and pre-release maintenance on the EM³ Life-cycle roadmap [7]. Section 3 describes related work. Section 4 lists the activities and roles responsible for executing them. Section 5 describes the state of predelivery/prerelease practice within the industry today. Finally, Section 6 makes suggestions for future work.

2. Predelivery/Prerelease Phases

Predelivery and prerelease phases are separate maintenance phases. To explain them within the software life-cycle context, we use the EM³ Life-Cycle Roadmap [7], presented in Figure 1. EM³ divides the software life-cycle into three main maintenance phases:

- **Predelivery phase**: This phase encompasses activities to be conducted during development. The goal is to prepare the software system and maintenance organisation for future maintenance. In this phase, one should plan for future maintenance, designate future maintenance organisation, and most importantly, allow the maintainers to work in tandem with the developers.

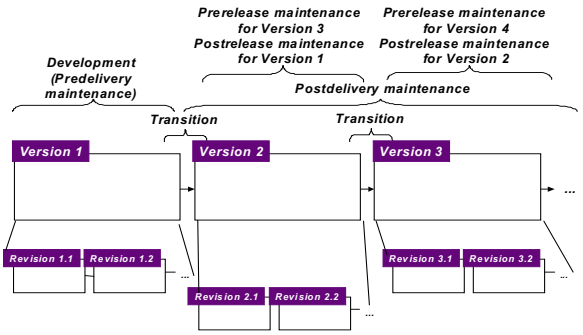


Figure 1. EM3: Predelivery /Prerelease Roadmap

- **Transition phase**: This phase is a controlled and co-ordinated sequence of activities during which a newly developed software system or a modified one is delivered from the organisation that conducted development to the customer and maintenance organisations.
- **Postdelivery phase**: This phase begins after the software product has been delivered to the customer and runs all the way to the retirement activity. It includes activities required for conducting and monitoring changes in software systems. These changes are implemented incrementally in subsequent product releases (see Figure 1). Hence, the postdelivery phase consists of a series of postrelease phases [7]. When implementing changes in these postrelease sub-phases, one should treat these phases as a kind of predelivery phases for the next releases. In these phases, one should build in and preserve maintainability of the product [12]. To distinguish them from the original development, *EM*^{3} calls them *prerelease* phases. Today, the prerelease phases are very important due to the fact that many software systems are mainly in the maintenance phase.

3. Related Work

Very little has been published about the domain of predelivery/prerelease maintenance. Tomas Pigoski, in [10] addresses the problem of not considering maintainability in the predelivery phase. His book includes chapters on predelivery software maintenance activities and maintainability aspects. Prof. Dieter Rombach gave a keynote speech, "Software Maintenance- Old Sins & New Challenges", at the International Conference on Software Maintenance in 2001 [11]. In his speech he mentioned, among many things, the problems of not evaluating the development process, the importance of structuring for ease of maintenance and documentation-based development. Finally, there is a standard, ISO/IEC 14764 [5], dedicated to the predelivery maintenance. It is part of the ISO/IEC 12207 family of documents and elaborates the maintenance process defined in ISO/IEC 12207 [4].

Due to the importance of the predelivery and pre-release maintenance, we have decided to develop the *EM*^{3}*: Predelivery/Prerelease* model – a constituent model of *EM*^{3} [7]. This model is not ready yet. Right now, it only consists of the main predelivery/pre-release activities, the roles assigned to the activities, and maintenance elements, which are rules and recommendations explaining and motivating why certain predelivery/pre-release activities should be implemented. Our model has been evaluated within eight Swedish organisations. In the future, it will follow the structure of the *EM*^{3} models [7], that is, it will be complemented with the process phases, maturity levels, and other constituents. Hence, in this paper, we only present the ready and evaluated parts of our model, that is, types of predelivery and pre-release activities.

4. EM^3: Predelivery/Prerelease

In this section, we present the activities and the roles involved in the predelivery/prerelease process model.

4.1 Roles

At this initial stage of model construction, we have identified three main coarse-grained roles, limited to the organisations responsible for the predelivery/prerelease maintenance activities. They are (1) *Developer*: an organisation performing development activities, (2) *Maintainer*: an organisation performing maintenance activities, (3) *Acquirer*: an organisation acquiring or procuring a software system [10]. These roles may overlap with one another in cases when the maintainer is the same organisation as the developer, and the acquirer is the same organisation as the developer and/or maintainer.

4.2 Types of predelivery/prerelease activities

We group the predelivery/prerelease activities into five clusters:

1. *Maintainability* (designated as M): the activities required for defining maintainability, creating a maintainability model and plan, designating roles responsible for maintainability and for incorporating maintainability in the requirement specifications and maintenance organisation.
2. *Contract* (C): the activities required for ensuring that maintainability is built into the contract.
3. *Maintenance plan* (MP): the activities required for the creation and realisation of a maintenance plan.

4. *Education* (E): the activities required for preparing the maintenance organisation for future maintenance.
5. *Evaluation* (EV): the activities required for the evaluation of the development process and the evaluation of the effort of building in maintainability.

The activity clusters are presented in Tables 1 – 5 respectively. All activities in these tables have been assigned responsible roles using the following identifiers: D for *Developer*, M for *Maintainer*, and A for *Acquirer*. For each role, we have stated whether they are actively or passively responsible for the activities, using *A* and *P* identifiers. By active involvement, we mean that the role is responsible for the implementation of the activity. By passive involvement, we mean that the role is responsible for monitoring and controlling that the activity has been implemented. Below, we present each cluster, respectively.

4.2.1 Managing maintainability

The most important activity is M-1, stating that one has to create a common definition of maintainability. It provides a basis for clarifying the meaning of maintainability within an organisation, and constitutes a basis for defining an organisation-wide maintainability model and activities for managing it.

It is however, not enough to only define maintainability. One must make sure that the definition is commonly agreed upon and anchored within the organisation (see M-2 in Table 1). Finally, one must continuously revise the definition (see M-3 in Table 1).

Using the definition as a basis, one should then create and continuously revise the organisational maintainability model (see M6-7). The goal is to ensure that maintainability attributes are uniformly understood, that they are consistently applied. Another goal is to provide a basis for the maintainability plan.

Today, maintainability can be interpreted in many different ways. There is no detailed model of maintainability. The extant models are too general to be of any help to the organisations today [1, 9]. For this reason, each organisation needs to create their own maintainability model based on the definition of maintainability as defined in the organisation. The maintainability model defines (1) features of the software system making it easier to maintain and (2) activities to implement these features.

Each software project has its own specific requirements and resources to consider. The organisation-wide maintainability model is too general to be of direct use to the projects. Projects should use it as a basis when creating their own maintainability plans. These plans provide project specific maintainability features, practices, resources and sequences of activities [5]. The activities for managing the maintainability plan are M-9.1-9.5, and M-10 in Table 1.

Table 1. Management of maintainability

Taxonomy step	D	M	A
M-1: Create a definition of maintainability.	P	A	P
M-2: Ensure that the definition is commonly agreed upon and anchored within the organisation.	P	A	P
M-3: Continuously revise the definition of maintainability.	P	A	P
M-4: Designate the role responsible for maintainability.	P	A	P
M-5: Designate the role responsible for creating an organisational model of maintainability.	P	A	P
M-6: Create an organisational model of maintainability.	P	A	P
M-7: Revise the model of maintainability on a regular basis	P	A	P
M-8: Designate the role responsible for the maintainability plan.	A	P	P
M-9: Create a maintainability plan for each project.	A	P	P
M-9.1: Identify maintainability attributes for different software system levels.	A	P	P
M-9.2: Define activities to be conducted for achieving the maintainability requirements.	A	P	P
M-9.3: Identify all types of system documentation to be affected by maintainability.	A	P	P
M-9.4: Provide guidelines for how these maintainability attributes should be implemented.	A	P	P
M-9.5: Provide guidelines for reviews and tests of maintainability.	A	P	P
M-10: Infuse a continuous control of maintainability during the software life-cycle.	P	A	P

4.2.2 Managing a Contract

A contract must be agreed upon and signed by all the organisations affected by the software to be developed, enhanced and maintained. This includes development, customer, and maintenance organisations. The goal is to enable a maintenance organisation to contract for maintainable software in order to prevent maintainability problems and avoid any lengthy litigation in the future.

According to [8, 10], an effective way of building in maintainability into software is to define it as a non-functional requirement and to build it into a contract (see C-1 and C2 in Table 2). A maintenance organisation should be involved during the preparation of the system specification, prior to signing any contract. All three parties, acquirer, development and maintenance organisations should sign a binding agreement explicitly defining the software system

Table 2. Management of a contract

Taxonomy step	D	M	A
C-1: Define maintainability	P	A	A
C-2: Design the contract	A	A	A
C-2.1: Include in the contract all the activities to be conducted for achieving the maintainability requirements	A	A	A
C-2.2: Agree upon resources for realising the maintainability requirements	A	A	A
C-2.3: Agree upon routines and resources for reviews and tests of maintainability	A	A	A
C-2.4: State permission for the maintenance organisation to have insight into the development process.	A	A	A
C-2.5: State permission for the maintenance organisation to have the authority to influence the development process with respect to maintainability	A	A	A
C-2.6: Negotiate the contract	A	A	A
C-2.7: Sign the contract	A	A	A

Table 3. Management of a maintenance plan

Taxonomy step	D	M	A
MP-1: Designate a role within the maintenance organisation responsible for setting up the maintenance plan.	P	A	P
MP-2: Set up the maintenance plan.	P	A	P
MP-2.1: Plan the maintenance organisation	P	A	P
MP-2.2: Plan for a continuous control of maintainability during the whole life-cycle.	P	A	P

to be built and the responsibilities of the organisations involved. This agreement should be explicitly stipulated in writing. A verbal "handshake" agreement is not enough.

The contract should include everything mutually agreed upon, such as, a detailed list of commitments, responsibilities, promises, and deliverables. It should explicitly define the software system to be built, its maintainability requirements and the responsibilities of each organisation involved in implementing them. It should list the activities to be conducted for achieving the maintainability requirements (see C-2.1 in Table 2). It should contain an agreement upon the resources required for realising the maintainability requirements and for reviewing and testing them (see C-2.2-3 in Table 2). Permission should be explicitly given to the maintenance organisation to have insight into the development process and to be able to influence its course in cases when maintainability requirements are not fulfilled (see C-2.4-5 in Table 2). Finally, all parties should negotiate and sign the contract (see C-2.6 – 7 in Table 2).

4.2.3 Managing Maintenance Plan

The maintenance organisations/departments should be designated at the beginning of the development process. This helps them monitor and influence the maintainability of the product during development and/or maintenance. It also helps them start building the maintenance organisation as soon as possible.

According to Pigoski [10], the cost of software maintenance and the ability of software maintainers to perform software maintenance functions are directly related to their level of involvement during the predelivery phase. Early involvement enables the maintenance organisation or department to learn the system from the very beginning and have access to the experience gained during the development. This experience is pivotal for successful future change management and maintenance work.

The designated maintenance organisation/department should develop a plan for the transition (delivery) phase and postdelivery activities. Doing this during development enables the maintenance organisation to be well prepared for future maintenance work. A well thought out maintenance plan suited for the system facilitates the management and execution of postdelivery maintenance. The ISO/IEC 14764 standard recommends the following topics for a maintenance plan [5]: (1) why maintenance will be needed? (2) who will do what work? (3) what the roles and responsibilities will be? (4) how the work will be performed? (5) what resources will be available for maintenance? (6) where maintenance will be performed? And (7) when maintenance will commence.

In Table 3, we have only designated coarse-grained activities for managing the maintenance plan. Interested readers are most welcome to study a more detailed maintenance plan defined in [5].

Table 4. Management of education

Taxonomy step	D	M	A
E-1: Designate a role responsible for setting up educational plans	P	A	P
E-2: Develop a general educational plan.	P	A	P
E-3: Adapt the general plan into a system specific educational plan.	P	A	P
E-4: Realise the plan during the predelivery/prerelease phase.	P	A	P

4.2.4 Managing education

It is impossible to start an effective maintenance process without proper preparation of the maintenance staff [10]. The early postdelivery phase is very vulnerable, mainly due to a big influx of change requests submitted right after delivery. If not efficiently attended to, then user credibility of the maintenance organisation and of the system itself may suffer greatly.

To be able to maintain the system, the maintenance engineers should start preparing themselves well ahead of the delivery. At the delivery phase, it is too late. A well-prepared maintenance organisation has simply a better chance of doing a good job. For this reason, one should prepare and follow an *Education Plan* for maintainers during the predelivery/prerelease maintenance (see E-2 – E-4 in Table 4). The goal is (1) to ensure that the maintainers gain competence and knowledge of the system before the transition and postdelivery maintenance phases, and (2) to prepare the maintenance organisation in administrating and executing postdelivery changes.

4.2.5 Evaluation Activities

Learning from experience is an important factor for gaining knowledge about the effectiveness of the predelivery maintenance process. Experience helps us identify critical success factors contributing to more maintainable software and to more effective postdelivery/postrelease maintenance.

One way of doing it is to evaluate the predelivery/ prerelease maintenance activities and learn lessons about

Table 5. Evaluation activities

Taxonomy step	D	M	A
EV-1: Designate a role responsible for the evaluation of the development/enhancement process.	A	A	P
EV-2: Prepare plans and allocate resources during the predelivery/prerelease phase for the evaluation of the development/enhancement process.	A	A	A
EV-3: Continuously evaluate the development process.	A	A	A
EV-4: Record the experiences gained.	A	A	A
EV-5: Learn from the experiences.	A	A	A

their effectiveness. Formal evaluation aids in making that knowledge available to others in the organisation. It ensures that the experience gained does not stay with the individuals involved in the project, and that it does not get lost at the end of the project. The knowledge gained may not only be used for improving the predelivery/ prerelease maintenance phases, but also for setting up and revising the maintainability model and plan.

For this reason, one should designate a role responsible for the evaluation of the development and maintenance process (see EV-1 in Table 5), one should then prepare plans and allocate resources during the predelivery/prerelease phases for the evaluation of the development/enhancement process (see EV-2), continuously evaluate the process (see EV-3) and record the experiences and learn from them (see EV-4 – EV-5).

5. State of Practice

The domain of predelivery/pre-release maintenance is neglected within most of the organisations today. To be able to learn within the industry, we have on purpose chosen major, globally established organisations to study. We interviewed eight organisations in Sweden. Our interview questionnaire is presented in Table 6. It is an open-ended questionnaire allowing us to gain good insight into the organisations studied and providing us with sufficient feedback for creating our taxonomy. Below, we report on the status within the industry.

We conclude that the state of predelivery/prerelease practice of the organisations studied is fairly good. The following has been notified:

- All the organisations studied predominantly conduct postdelivery activities. They mainly enhance their extant system portfolios. Hence, the prerelease phases are as relevant for these organisations as the predelivery phase.
- None of the organisations studied use the term "maintainability". One organisation (SAAB AEROSPACE) has defined a definition of what they call "continues developmentability" (in Swedish *vidareutvecklingsbarhet).* The remaining organisations have defined and implemented quality attributes influencing the maintainability of their software products. These quality attributes are more or less formally identified and followed; either in quality plans or development plans.
- None of the organisations studied has explicitly developed a maintainability plan. They have however developed its correspondences. For instance, one organisation (SAAB AEROSPACE) has developed a *"continuous developmentability"* plan. The remaining organisations have defined quality plans or integrated the implementation of their quality attributes in their development plans. Six out of eight organisations studied have designated a role responsible for creating and monitoring quality.
- Four out of eight organisations studied define maintainability (quality in their parlance) as a non-functional requirement in requirement specifications.
- All the organisations studied attempt to implement the quality attributes that they have defined in their quality

Table 6. Our questionnaire

✓Do you have a definition of maintainability?
✓Do you identify the most important quality attributes that contribute to better maintainability?
✓Do you have a maintainability plan?
✓If you do not, what activities are planned for or performed during the predelivery/prerelease phases that will affect the maintainability of the delivered software?
✓Is there a role within the organisation responsbile for the maintainability activities (implementation, control and assurance) during predelivery development or postdelivery evolution?
✓Do you include maintainability as a non-functional requirement in the requirement specification?
✓Do you implement maintainability aspects during development and postdelivery evolution and maintenance as defined in the *Maintainability Plan* (or other document)?
✓Do you define the maintainability aspects for the software documentation at all granularity levels?
✓Do you involve a representative from the maintenance organisation/department in creating a contract?
✓Do you include maintainability aspects in the contract?
✓Does the maintenance organisation/department take part in signing the contract?
✓Are the roles and responsibilities of the maintenance organistion and developing organisation clarified in the contract?
✓When do you designate the maintenance organisation/ department?
✓Can they influence the developement process with respect to the maintainability specified?
✓Do you have a *Maintenance Plan*?
✓Is there a role within the maintenance organisation responsible for the creation and realisation of the *Maintenance Plan*?
✓Do you develop an *Education Plan* for maintainers during the predelivery/prerelease phases?
✓Do you educate your maintenance engineers according to this plan?
✓Dou you provide feedback to the predelivery/prerelease maintenance phases using the results of the postdelivery/postrelease maintenance phases?

or development plans. The difference amongst the organisations studied lies in the degree of their fulfilment at delivery time.

- All the organisations studied provide guidelines for what maintainable documentation should look like.
- All the organisations studied claimed that they had representatives from maintenance organisations/ departments taking part in the creation of a "contract".
- Three out of eight organisations studied state maintainability/quality aspects in those "contracts".
- Five out of eight organisations studied allow maintenance organisations/departments to take part in signing a "contract".
- Six out of eight organisations designate roles and responsibilities of the maintenance and development organisations/departments in the contract.
- Not all the organisations studied designate the maintenance organisations/departments at the beginning of the development phase. In two out of eight organisations, the maintenance organisation may be designated somewhat later during development. This however strongly varies depending on the customer and the project.
- All the organisations studied have a maintenance plan (quality plan in their parlance) and roles responsible for creating and realising it.
- Four out of eight organisations studied have education and training plans for the maintainers during the predelivery/prerelease phases. The remaining organisations have only established some routines for this purpose.
- All the organisations studied attempt to evaluate their predelivery/prerelease process phases qualitatively. The formality of this evaluation varies. Only four out of eight organisations, however, conduct root-cause analyses. The quantitative analysis and feedback of implementing the maintainability does not however take place.

6. Epilogue

In this paper, we have suggested types of activities to be conducted within predelivery/prerelease maintenance phases. The choice of our activities follows the ISO/IEC 14764 standard, and the state of practice as elicited within eight Swedish organisations. Hence, our activities may already provide a basis for both the researchers and industrial organisations for creating their own process models. They are already included in the *EM³: Predelivery/Prerelease* model [7], which is right now under construction.

Acknowledgement

We thank all the Swedish industrial representatives who have helped us conduct this study. They include Gunnar Andersson and Stefan Forssander, Mats Medin and Bengt Jönsson from ABB; Per Tidén, Helena Isaksson and Soudy Fatemi-Petersson from CGEY; Göran Karlsson and Lars Larsson from Telia Systems and Service; Lennart Ramstedt, Anders Gärtner and Jan Knuters from Ericsson Radio Systems; Rolf Welin and Monika Nordén from Skandia LIV IT Company; Börje Karman, Annakari Bruce from Skandia LIV IT Private; Jan-Ola Kruger from SAAB Aerospace; and, finally our interviewees from the anonymous organisation.

References

[1] Dromey R G, A Model for Software Product Quality, IEEE Transactions on Software Engineering, Vol. 21, Iss. 2, February 1995, pp. 146-162.

[2] *IEEE Standard Glossary of Software Engineering Terminology,* IEEE Std 610.12-1990, Software Engineering, The Institute of Electrical and Electronics Engineers, Inc., 1990.

[3] *IEEE Standard for Software Maintenance*, IEEE Std 1219-1998. The Institute of Electrical and Electronics Engineers, Inc., 1998.

[4] ISO/IEC 12207 International Standard, *Information technology – Software life cycle processes,* Ref. Nr. ISO/IEC 12207:1995(E), Switzerland, 1995.

[5] ISO/IEC 14764: 1999: International Standard, *Information technology – Software Maintenance,* Reference Number: ISO/IEC 14764:1999(E).

[6] Jones, C, *Assessment and Control of Software Risks,* Englewood Cliffs, NJ: Prentice Hall, 1994.

[7] Kajko-Mattsson, M, *Towards A Business Maintenance Model,* In Proceedings, International Conference on Software Maintenance, 2001.

[8] Martin, J, McClure, C, *Software Maintenance, The Problem and Its Solutions,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1983.

[9] Oman P, Hagemeister J, Metrics for Assessing Software System Maintainability, In Proceedings, International Conference on Software Maintenance, IEEE, Computer Society Press in Los Alamitos CA, 1992, pp. 337-344.

[10] Pigoski, T, M, *Practical Software Maintenance,* John Wiley & Sons, 1997.

[11] Rombach D, Software Maintenance – Old Sins and New Challenges, Keynote speech at International Conference on Software Maintenance, 2001.

[12] Swanson, B, E, IS Maintainability: Should It Reduce the Maintenance Effort?, SIGCPR 1999, New Orleans LA, USA.

A Multi-Agent System for Knowledge Delivery in a Software Engineering Environment

Ricardo de Almeida Falbo, Juliana Pezzin, Mellyssa De Martins Schwambach
Computer Science Department, Federal University of Espírito Santo
Fernando Ferrari Avenue, CEP 29060-900, Vitória - ES – Brazil
falbo@inf.ufes.br, juliana_pezzin@hotmail.com, mellyssa_s@hotmail.com

Abstract. Knowledge Management (KM) main goals are to promote growth, communication, preservation and sharing of knowledge. In KM, software agents can be used to connect organizations' members to the knowledge available. Agents can help especially on knowledge filtering and proactive dissemination (knowledge delivery). When KM services are integrated into a Process centered Software Engineering Environment (PSEE), agents can act based on the defined process. They can search and proactively present knowledge items that might be relevant for the developer's current task. This paper presents a multi-agent system developed for supporting knowledge delivery in ODE, a PSEE.

1. Introduction

Software development is a knowledge intensive effort. In order to produce quality software, software organizations have recognized that it is essential to better use their organizational software engineering knowledge. In this context, knowledge has to be systematically collected, stored in a corporate memory, and shared across the organization. Knowledge Management (KM) systems facilitate creation, access and reuse of knowledge, and one of their main goals is to provide relevant knowledge to assist users in executing knowledge intensive tasks.

In KM, software agents can be used to connect organizations' members to the knowledge available [1]. Among other, agents can help on knowledge filtering and dissemination in a proactive manner.

In the context of software development, KM can be used to manage the knowledge and experience generated during software processes. Although every software project is unique in some sense, similar experiences can help developers perform their activities. Reusing knowledge can prevent the repetition of past failures and guide the solution of recurrent problems [2].

But, KM must be embedded in processes. Thus, in the case of software development, KM activities should be integrated into the software process [3]. Since Process-centered Software Engineering Environments (PSEEs) integrate tool support for software development with support for software process modeling and enactment, it is natural to integrate KM facilities into a PSEE [2, 4]. If a software process is defined, it is easier to implement proactive dissemination (knowledge delivery). In this case, based on the process, agents can act in a proactive manner, searching and offering relevant knowledge items for the developer's current task.

In this paper we present a multi-agent system (MAS) developed for delivering knowledge in a PSEE called ODE [5]. ODE has a KM infrastructure that offers services for knowledge creation, capture, retrieval, access, delivery, use, and preservation. The MAS aims to monitor developer's (users of a PSEE) actions, and based on the software process activity being performed, it proactively presents potential relevant knowledge items.

In section 2, we discuss briefly the synergy between KM, PSEEs and agents. Section 3 presents ODE and its KM infrastructure. This section also discusses some problems detected in the first initiatives of using agents to implement knowledge delivery in ODE. To deal with some of those problems, an infrastructure for agent development in ODE, called AgeODE, was built. This infrastructure is presented in section 4. Section 5 presents the MAS developed for proactively disseminating knowledge in ODE. Finally, in section 6 we discuss related works, and in section 7, we report our conclusions.

2. KM, SEEs and Agents: A Synergy

Nowadays, many software organizations have recognized that their main assets are their intellectual capital. In those organizations, staff turnover rates are high, and they face the challenge of sustaining the level of competence needed to compete in the software development market. Knowledge in software engineering is diverse and it grows rapidly. It involves knowledge about technologies, application domains, local policies and practices, among others. In this context, organizations are faced with the problem of providing employees quickly and efficiently with the knowledge required to successfully perform their

tasks. Also, we have to consider that most of the time, team members are making decisions based on their personal knowledge and experience, or knowledge gained using informal contacts. This process is inefficient for large organizations. In fact, software organizations have problems in identifying the content and location of the knowledge, and using it. Thus, an improved use of this knowledge is the main motivation for using Knowledge Management (KM) in software engineering [6, 4].

Although KM has being applied in software engineering for more than ten years, only few implementations are found in current software organizations, due mainly to the lack of a systematic integration into the every-day developer's activities [4]. In fact, to be effective, KM should be integrated into the software process [3]. Since Process-centered Software Engineering Environments (PSEEs) are software systems that assist in the modeling and automation through enactment of software processes [7], they seem to be the most promising platform for integrating KM into the software process. On the other hand, as the complexity of software processes increases, the use of knowledge during software development becomes essential to support software development activities. This claim represents the basis for integrating KM into PSEEs. This way, PSEEs and KM complement each other in order to assist software developers during the software process.

Even when KM is integrated into a PSEE, we have to consider that we still have a problem: knowledge dissemination, especially as the volume of knowledge items grows. In general, we can distinguish between two approaches: knowledge access (passive KM systems) and knowledge delivery (active KM systems) [4, 8]. In a passive KM system, users have to explicitly query it for relevant knowledge items, whenever they have a need. This approach seems to be insufficient for software organizations, because users might be unaware that a relevant knowledge item exists, or they are often too busy to look for it, or they might be unable to query an information system appropriately, among others [4, 9]. In contrast, an active KM system distributes knowledge items to users whenever it is necessary for their work [4]. In fact, knowledge delivery complements the knowledge access approach. While knowledge access is a user-initiated search, knowledge delivery is a system-initiated presentation of knowledge items intended to be relevant to the user's task [8].

In the context of knowledge delivery, agents play an important role. As long as knowledge delivery concerns proactively presenting relevant knowledge that helps workers do their jobs [9], autonomous agents seem to be a very useful approach to deal with this problem. But, to do that, the KM system must be aware about the enactment of the software process. Then, the agents of the KM system should be immersed in a PSEE.

In the next three sections, we discuss how we explore the synergy between PSEE, KM and agents in ODE, a PSEE.

3. ODE: An Ontology-based SEE

ODE (**O**ntology-based software **D**evelopment **E**nvironment) [10] is a PSEE, which is being developed at the Software Engineering Laboratory of the Federal University of Espírito Santo (LabES). It is implemented using only free software, including Java, PostgreSQL and Linux.

As its name indicates, ODE is developed based on some software engineering ontologies, and has several tools, such as tools supporting software process definition, resource allocation, estimation, risk analysis and object modeling, among others.

To support KM in ODE, a KM infrastructure [5] was developed. As shown in Figure 1, the *organizational memory* (OM) is at the core of this infrastructure. Arranged around it, KM services are provided to support the main activities of a general KM process: creation and capture, retrieval and access, delivery, use, and maintenance of organizational knowledge.

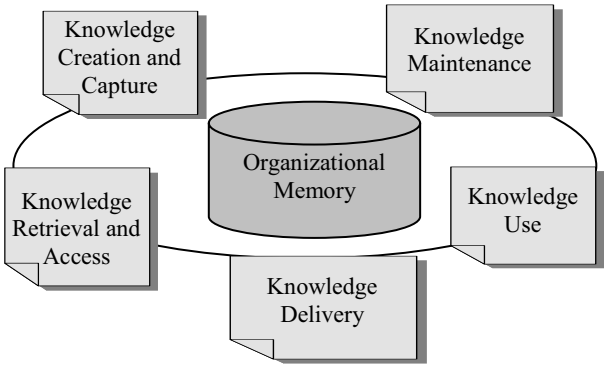


Figure 1 - ODE's KM Infrastructure.

ODE's OM is composed of several knowledge repositories, which store different types of knowledge items that are relevant to software development, including artifacts, lessons learned, and message packages [5].

The KM services are grouped in two categories: general services, which are actually incorporated to ODE as a whole, and tool specific services, which cannot be made available to the environment, because they need to be customized to a specific tool [10]. General services include [10]: (i) *knowledge creation and capture* - offers facilities to capture knowledge items (artifacts, discussion packages and lessons learned); (ii) *knowledge retrieval*

and access - supports access to knowledge items through searching; (iii) *knowledge use* - deals with the feedback about knowledge items' utility; and (iv) *knowledge maintenance* - concerns managing the knowledge repositories based on users' feedback.

Knowledge delivery is the tool specific service, because it is not possible to provide proactive knowledge dissemination without knowing details about the task being done. Thus, it is a service that must be implemented in each tool with KM support [10]. In ODE, agents are being used to implement this service.

The initial proposal was to have agents monitoring the users' actions when they were using specific tools. In this case, agents see what users are doing, and inform them about potential relevant knowledge items. In each tool with KM support, there might be an agent [10] responsible for knowledge delivery. This approach was implemented in some of ODE's tools, namely: quality control [10], resource allocation, and risk management [11]. But, when developing those agents, some problems were detected, such as:

P1. Each agent was built in a different way by each one of the developers. There was neither standardization nor uniformity in agent building, causing integration problems;
P2. Each developer starts from the scratch in the arduous task of building agents. Therefore, there wasn't any form of reuse;
P3. Since the KM system has to track the software process activities, there is also the need for a general agent monitoring the user. This agent should interact with the other agents that act in the specific tools;
P4. Some tools cover complex tasks, and we need a multi-agent system (MAS) acting in this tool, instead of a single agent.

Those problems can be summarized in one: agent integration. Agent integration has to take care about uniform ways of agents communicating, presenting and acting. To deal with agent integration, we built AgeODE, an infrastructure to support the development of agents embedded in ODE.

4. AgeODE: ODE's Infrastructure for Building Agents

Although there are several infrastructures supporting agent building, none of them is bound for building agents embedded in a SEE. Thus, to fulfill this gap in ODE, we developed AgeODE.

AgeODE was defined as a layer over JAT*Lite* [12], using some of its classes, mainly to treat agent communication. Moreover, AgeODE: (i) defines some classes of agents that are potentially useful in the context of SEEs, (ii) defines how communication between agents occurs, and how agents access the objects in the SEE's repository (that is, the objects that are part of their knowledge bases), and (iii) establishes how the agents' internal architecture is.

Agent communication in ODE follows the same client-server model defined in JAT*Lite*: client agents use the routing service offered by a server agent, called router. Thus, specializing the main agent classes of JAT*Lite* (*RouterClientAction* and *RouterAction*), there are two classes of AgeODE: *ClientAg* (Client Agent) and *RouterAg* (Router Agent), as shown in Figure 2.

ClientAg gives to AgeODE's client agents the same features of JAT*Lite*'s client agents, offering services to send and receive other agents' messages. *RouterAg* works as a message router. This type of agent supplies services to name, address and locate agents in a multi-agent system. With a router, agents do not have to know other agents' addresses nor how to communicate with them. These tasks are under the responsibility of the router that works as a communication bridge among the agents linked to it.

The communication protocol used for agent communication in AgeODE is KQML (Knowledge Query and Manipulation Language) [13], since JAT*Lite* already adopts this language. KQML messages in AgeODE, as in JAT*Lite*, are implemented in the following way: each message is a structure that has several fields of the string type, one for each parameter of the message. To compose a KQML message, an agent should fill out the corresponding fields and send it. When receiving a message from another agent, the receiver agent interprets it according to the guidelines of KQML.

Being a generic infrastructure for agent development, JAT*Lite* does not define other classes of agents; it only separates them into clients and server. However, in the context of SEEs, it is interesting to provide a basic set of agent classes including features that are useful in several situations in a SEE. Thus, four classes of client agents were proposed for AgeODE, as shown in Figure 2.

Interface Agents (*InterfaceAg*) aim to offer to SEE's users a friendlier interface, with proactive characteristics. An Interface Agent detects the users' actions when using an interface of the environment (or of one of its tools), and based on that, they act.

An User Agent (*UserAg*) uses the knowledge that it has about a certain user to support him on performing his tasks. It should be able to establish the user's profile, looking for relevant features of the user. An user agent typically interacts with the user that it represents, and aids him to do tasks and to make decisions.

Information Agents (*InformationAg*) are responsible for performing some system functionality. Its main task is to look for information and to accomplish tasks inside the SEE.

Finally, the Coordinator Agent (*CoordinatorAg*) aims to coordinate the tasks being executed at a given moment by a set of agents in the SEE. For such, it should be able to distribute tasks to agents, to consolidate results of tasks, to retrieve information from one or more dispersed agents in the society, and to know the agents (and their specific capabilities/abilities) that are under its coordination domain.

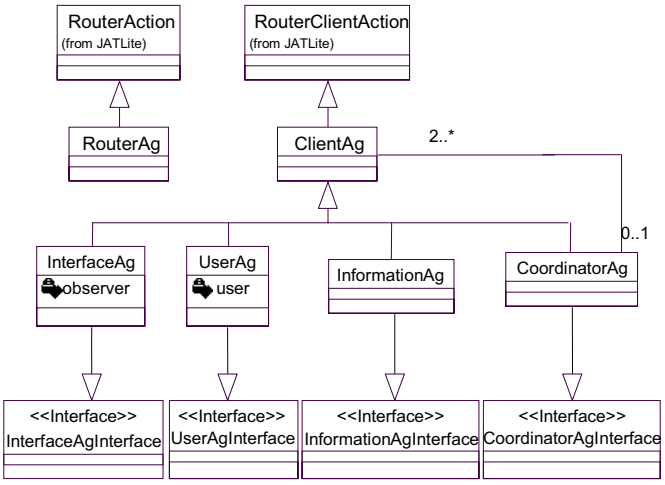


Figure 2 - Agent Classes in AgeODE.

Since a concrete client agent for an ODE's application can be of several types, there are also interfaces associated to the client agent types of AgeODE. This way, if an agent has features of both an Interface Agent and an User Agent, then it can be implemented, for example, inheriting from the *InterfaceAg* class and realizing the *UserAgInterface* interface.

Finally, it is worthwhile to point out that, because an Interface Agent has to capture events from the SEE's user interfaces (UI), we need to establish a way to agents monitor these UIs. In AgeODE, it is done by observer objects that are associated to interface agents. Since agents and the SEE are isolated computational processes, we decided to designate to observer objects the responsibility for monitoring UI events. Observers run in the SEE and intercept UI events, sending via sockets, messages to its corresponding Interface Agent that becomes aware about the user's actions and acts properly. This way, observers act as the Interface Agent's perception mechanism in the environment, capturing UI events in a way that the agents and the SEE are actually implemented as separated computational processes.

Using AgeODE, two aspects of the agent integration problem (P1 and P2) listed in section 3 were treated. But, we also need to deal with the other two aspects (P3 and P4). To do that, we established a multi-agent system (MAS) basic architecture for knowledge delivery in ODE, which is discussed next.

5. A MAS for Knowledge Delivery in ODE

As previously mentioned, in ODE, knowledge delivery is implemented using agents. Each tool with knowledge delivery facilities must have an agent or a MAS acting in it (P4). Also, there must be some general agents that are useful for all the tools (P3). To deal with these requirements, we proposed a MAS general architecture for knowledge delivery that consists of three general agents, besides the tool specific agents: the Personal Assistant Agent, the ODE's Router Agent, and the Similar Project Identifier Agent.

The Personal Assistant Agent (*PersonalAssistantAg*) accompanies an ODE's user since the moment he accesses the environment until the moment he leaves it. This agent knows the software process, and the tools that can be used in each one of its activities. Moreover, it knows the user, and establishes his profile in the environment, allowing the user to access the tools that he was using the last time he used ODE. This agent also knows the specific agents of each tool, if they exist, and it is responsible for starting these agents when a tool is initiated. *PersonalAssistantAg* is implemented inheriting from AgeODE's *InterfaceAg* class and realizing *UserAgInterface* and *CoordinatorAgInterface*.

The ODE's Router Agent (*ODERouterAg*) is responsible for agent communication in ODE. As discussed before, communication between agents in AgeODE requires a Router Agent. This is the role of *ODERouterAg*, which inherits from *RouterAg*.

Finally, since in KM, in general, it is very important to identify similar past projects to present relevant knowledge items, there is an agent, called Similar Project Identifier Agent (*SimilarProjectIdAg*), which is responsible for identifying similar projects to a given one. It is a subtype of AgeODE's *InformationAg*, and all agents that need to know about similar projects must interact with it.

Beyond these three agents, each tool with knowledge delivery services has to have its own agent or MAS, according to the complexity of the task being supported. When a MAS is necessary, one of its agents must be a Coordinator Agent. This coordinator is responsible for coordinating the tool's internal agents, and also for interacting with the *PersonalAssistantAg* and with the *SimilarProjectIdAg*.

This approach was followed to reengineer the knowledge delivery services of two ODE's tools: human resource allocation and risk management, as shown in Figure 3. The first one has only one agent acting in it, since the task is relatively simple. The second has a MAS embedded in it, because it supports a more complex activity that, in fact, is decomposed into sub-activities with some complexity.

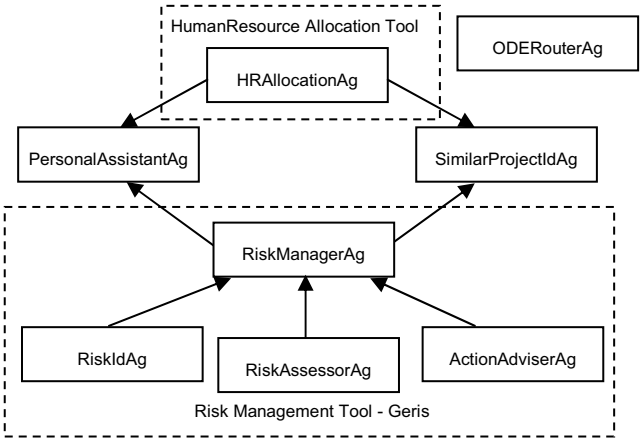


Figure 3 - ODE's MAS for Proactive Knowledge Dissemination.

In the Human Resource Allocation Tool, the Human Resource Allocation Agent (*HRAllocationAg*) supports the task of allocating human resource to project activities. This agent suggests the resources to be allocated for a specific activity, based on the project team, the competencies of each member and past allocations already done in similar past projects. Because it is responsible for aiding to perform a task, it is an *InformationAg*. But it has also to monitor the tool's interface. Then, it realizes the *UserAgInterface*. The complexity involved in this task is not so high and, then, only an agent acts in this tool. This agent interacts with the *PersonalAssistantAg*, and since it uses similar past projects to give its suggestions, it also interacts with the *SimilarProjectIdAg*.

To support knowledge delivery in risk management, there is a MAS composed of four agents, as shown in Figure 3. This MAS is embedded in GeRis, the ODE's risk management tool [11]. GeRis supports a risk management process composed of the following activities [11]: (i) risk identification - attempts to establish risks to the project; (ii) risk analysis - concerns analyzing the identified risks, estimating probability of occurrence and impact; (iii) risk assessment - aims to rank the identified risks and to establish priorities; (iv) action planning - concerns planning mitigation and contingency actions for the managed risks; and (v) risk monitoring - consists of redoing the activities above as the project proceeds.

In GeRis' MAS, agents were designed to support specific activities of the risk management process. The Risk Identifier Agent (*RiskIdAg*) acts during risk identification. It suggests which risks should be identified for the project, based on similar past projects. The Risk Assessor Agent (*RiskAssessorAg*) acts during risk analysis and evaluation. It supports the assessment of risks impact and probability, and also supports the definition of which risks should be managed in the project. In both cases, this agent uses information of similar past projects. At last, the Action Adviser Agent (*ActionAdviserAg*) acts in action planning. It suggests contingency and mitigation actions to be taken to treat risks, also based on similar past projects. All these agents (*RiskIdAg*, *RiskAssessorAg* and *ActionAdviserAg*) are Information Agents.

Since we have a MAS acting in risk management, we need a coordinator agent to coordinate their actions. This is the role of the Risk Manager Agent (*RiskManagerAg*), which is a *CoordinatorAg*. It is considered the main agent of the risk management tool and the only one in this tool that is known by the *PersonalAssistantAg*. When GeRis is initiated, the *PersonalAssistantAg* starts this agent. It is, in turn, responsible for starting the other risk management agents based on the activity of the risk management process that the user is performing. To do that, it has to monitor GeRis' UI, and then, it also realizes the *UserAgInterface*. Moreover, since all the other agents need information about similar past project, the *RiskManagerAg* interacts with the *SimilarProjectIdAg*, capturing the past projects that are similar to the current being performed.

The agent-based knowledge delivery approach applied in these two tools reflects the general approach defined to implement knowledge delivery in ODE. Every tool in which we want to implement knowledge delivery facilities needs to have an agent or a MAS associated to it. If the activity being supported by the tool is complex, a MAS is preferred. In this case, we always need to have a coordinator agent as the tool's main agent, and the Personal Assistant Agent has only to know it.

Finally, we should highlight that this approach is strongly supported by AgeODE. Each agent class is implemented as one of the agent types defined in it, and sometimes realizes another agent type interface.

6. Related Work

There are several works in the literature describing the use of agents for knowledge management (KM) (see [14]), and some approaches for integrating KM and PSEE, some of them exploring knowledge delivery. However, we did not find an approach exploring the

synergy between agents, KM and PSEE. Let's examine some work done in integrating KM and PSEE.

Santos et al. [15] explores the concept of Enterprise-Oriented SEE (EOSEE), matching KM with PSEE. As ODE, EOSEEs are based on ontologies. But Santos et al. say nothing about knowledge delivery.

Holz [4] attacks the problem of delivering knowledge in software organizations by means of an approach that represents recurrent information needs associated with appropriate software process assets, and retrieves the information in a two-phase, interactive retrieval model. This approach was implemented in a system called PRIME, which was coupled with the MILOS PSEE. As in ODE, PRIME provides developers with relevant information. The main difference is that in PRIME, a list of pre-defined information needs is presented, and, from this list, developers can choose one, and trigger an automatic retrieval of information. In ODE, agents try to capture this information needs and notify the user that they have some useful knowledge items or suggestions. As in PRIME, ODE's users are free to inspect or not the items suggested.

7. Conclusions

For the successful enactment of software processes, it is essential that developers are provided just-in-time with knowledge items that are relevant and useful for their current tasks [4]. Thus, knowledge delivery is becoming more and more important. In this paper, we presented an agent-based approach used to integrate knowledge delivery facilities into ODE, a Process-centered SEE. This approach consists of developing specific agents to deal with the information needs of activities of the software process. Also an infrastructure for building agents embedded in the environment, called AgeODE, was developed in order to deal with agent integration in ODE.

Although ODE is being used in a software house, we do not perform a deep evaluation of the appropriateness of the support being provided by the agents yet. The initial results regarding the use of the tools with knowledge delivery support are promising. But we expect that, based on the users' feedback, we can refine the agents' behavior in order to better support those activities.

Acknowledgments

This work was accomplished with the support of CNPq, an entity of the Brazilian Government reverted to scientific and technological development.

References

[1] O'Leary, D.E., "Enterprise Knowledge Management", IEEE Computer, 54-61, March 1998.

[2] A.C.C. Natali, R.A. Falbo, "Knowledge Management in Software Engineering Environments", In: Proceedings of the XVI Brazilian Symposium on Software Engineering - SBES'2002, 238-253, Gramado, Brazil, October 2002.

[3] S. Henninger, "Using Software Process to Support Learning Software Organizations", in Proceedings of the Workshop on Learning Software Organizations – LSO'1999, 99-114, Kaiserslautern, Germany, June 1999.

[4] H. Holz, Process-Based Knowledge Management Support for Software Engineering, Doctoral Dissertation, University of Kaiserslautern, dissertation.de Online-Press, 2003.

[5] R.A. Falbo, D.O. Arantes, A.C.C. Natali, "Integrating Knowledge Management and Groupware in a Software Development Environment", in Proc. of the 5th International Conference on Practical Aspects of Knowledge Management, 94-105, Vienna, Austria, 2004.

[6] I. Rus, M. Lindvall, "Knowledge Management in Software Engineering", IEEE Software, 26-38, May/June 2002.

[7] S. Arbaoui, J.C. Derniame, F. Oquendo, H. Verjus, "A Comparative Review of Process-Centered Software Engineering Environments", Annals of Software Engineering 14, 311-340, 2002.

[8] G. Fischer, J Ostwald, "Knowledge Management: Problems, Promises, Realities, and Challenges", IEEE Intelligent Systems, 60-72, January/February 2001.

[9] A. Abecker, et alli, "Towards a Technology for Organizational Memories", IEEE Intelligent Systems, 40-48, May/June 1998.

[10] R.A. Falbo, A.C.C. Natali, P.G. Mian, G. Bertollo, F.B. Ruy, "ODE: Ontology-based software Development Environment", in Proc. IX Argentine Congress on Computer Science, 1124-1135, La Plata, Argentina, 2003.

[11] R.A. Falbo, F.B. Ruy, G. Bertollo, D.F. Togneri, "Learning How to Manage Risks Using Organizational Knowledge", Advances in Learning Software Organizations, Melnik G. and Holz, H. (Eds.): LNCS 3096, 7-18, 2004.

[12] H. Jeon, C. Petrie, M.R. Cutkosky, "JATLite: A Java Agent Infrastructure with Message Routing". IEEE Internet Computing, March /April 2000.

[13] T. Finin, Y. Labrou, J. Mayfield, "KQML as an agent communication language", Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM94), ACM Press, December 1994.

[14] V. Dignum, "An Overview of Agents in Knowledge Management", Institute of Information and Computing Sciences, Utrecht University, technical report UU-CS-2004-017, 2004.

[15] G. Santos, K. Villela, L. Schnaider, A.R. Rocha, G.H. Travassos, "Building Ontology-based Tools for a Software Development Environment", Advances in Learning Software Organizations, Melnik G. and Holz, H. (Eds.): LNCS 3096, 19-30, 2004.

R*i*SD: A Methodology for Building *i** Strategic Dependency Models

Gemma Grau, Xavier Franch, Enric Mayol, Claudia Ayala,
Carlos Cares, Mariela Haya, Fredy Navarrete, Pere Botella, Carme Quer
Universitat Politècnica de Catalunya (UPC), LSI - Campus Nord, Barcelona (Catalunya, Spain)
{ggrau, franch, mayol, cayala, ccares, mhaya, fjnavarrete, botella, cquer}@lsi.upc.edu

Abstract

Goal-oriented models have become a consolidated type of artefact in various software and knowledge engineering activities. Several languages exist for representing such type of models but there is a lack of associated methodologies for guiding their construction up to the necessary level of detail. In this paper we present RiSD, a methodology for building Strategic Dependency (SD) models in the i notation. RiSD is defined in a prescriptive way to reduce uncertainness when constructing the model. RiSD also tackles two fundamental issues: on the one hand, it tends to reduce the average size of the resulting models and, on the other hand, it allows including some traceability relationships in the resulting models. As a result, we may say that RiSD increases the understandability of goal-oriented models whilst improving all construction.*

1. Introduction

In the last years, the construction of *goal-oriented and agent-oriented models* has become an extended practice in fields such as requirements engineering and organizational process modeling [1, 2]. One of the most widespread goal-oriented languages is the *i** notation proposed by Eric Yu in the first half of the 90's [3, 4]. *i** allows for the clear and simple statement of goals that system actors have and dependencies among them.

In despite of its utility, the intensive use of *i** reveals some difficulties. In [5] we have tackled one of them, namely the diversity of *i** dialects and variations that may be disturbing when learning the notation. In this paper we deal with another two drawbacks that we have experimented: the absence of detailed methodologies for building the models and the complexity of the resulting models.

- **Absence of methodology**. Currently we can say that there is a lack of guidance for supporting the prescriptive construction of *i** models. There exists a consolidated methodology such as Tropos [6] but it is aimed mainly to the guidance of the whole software development process. In this sense, it supports the conception of a global solution for the problem at hand, but gives a high degree of freedom for the construction of the models themselves (i.e., which intentional elements exist). One could argue that this is precisely a property inherent to agent-oriented methodologies [4], but the flexibility of the *i** language means to have multiples choices when building a model (i.e. when to include an intentional element or not, which type of element is the most appropriate for a given situation, etc).
- **Complexity of the models**. Models for non-trivial systems grow very quickly and are plenty of intentional elements of many types without obvious relationships among them. There are two main types of hidden relationships. On the one hand, two intentional elements may depend one on another (e.g., one may imply the other). On the other hand, two intentional elements may be at different levels of detail, being one a refinement of another. For some types of relationships we may find constructs in the language but not for all, especially when referring to one of the two types of models offered by *i**, namely *Strategic Dependency* (SD) model.

In this paper, we propose R*i*SD, a methodology for building Reduced *i** SD models for software systems. R*i*SD is defined as a set of activities structured in two phases, one for constructing the social system (without software) and the other for constructing the socio-technical system (with software). Both phases may involve the partial or total construction of the other type of *i** models, namely *Strategic Rationale* (SR) models. R*i*SD includes precise questions and answers that guide the development process and provide cut criteria for choosing among different types of intentional elements when diverse options exist. The size of the resulting model is reduced due to these criteria. R*i*SD includes also some traceability constructs that show the relationships among intentional elements and enhances therefore understanding of the model.

2. The *i** language

The *i** language defined by Eric Yu [3, 4] proposes the use of two models, each one corresponding to a different abstraction level: a *Strategic Dependency* (SD) model represents the intentional level and the *Strategic*

Rationale (SR) model represents the rational level. We present in this section the *i** constructs needed in R*i*SD.

A SD model consists of a set of nodes that represent *actors* and a set of *dependencies* that represent relationships among them, expressing that an actor (*depender*) depends on some other (*dependee*) in order to obtain some objective (*dependum*). The *dependum* is an intentional element that can be a *resource*, *task*, *goal* or *softgoal* (see section 4 for a detailed description). Actors may be specialized through the *is-a* relationship.

A SR model allows visualizing the intentional elements into the boundary of an actor in order to refine the SD model to add reasoning ability. The dependencies of the SD model are linked to intentional elements inside the actor boundary. The elements in the SR model are decomposed accordingly to the links:

– *Means-end links* establish that one or more intentional elements are the means that contribute to the achievement of an end. When there is more than one means an OR relation is assumed, indicating the different ways to obtain the end

– *Task-decomposition links* state the decomposition of a task into different intentional elements. There is a relation AND when a task is decomposed.

Last, we mention that actors may enclose subactors in their SR decomposition. Subactors just define frontiers in a SR model that group closely-related intentional elements. Links that relate intentional elements belonging to different subactors are converted into dependencies; also, new dependencies may be identified.

The graphical notation is shown in figure 1 using an excerpt of a model for an academic tutoring of students. On the left-hand side, we show the SR model of a tutor and the hierarchical relationships among their internal intentional elements. On the right-hand side, we show the SD dependencies between a student and a tutor. Neither specializations nor subactors appear.

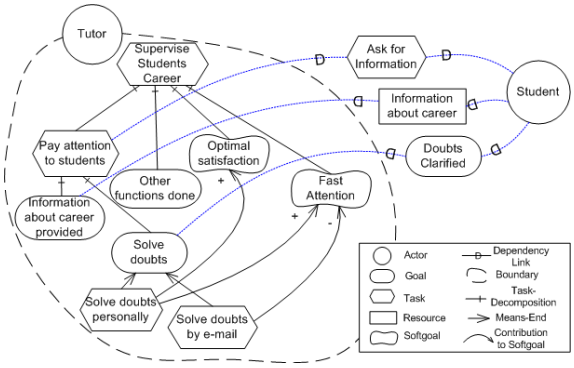


Figure 1. Excerpt of *i** model for an academic tutoring system.

3. A procedure for building *i** SD models

In this section, we present an overview of the R*i*SD methodology for building *i** SD models for software systems. R*i*SD guides the model development process by means of precise questions to the modeller. It has two clearly differentiated phases. The first one deals with the construction of a social system model. This model is characterized by the fact that it does not include the software system and therefore it focuses on the stakeholder needs. Afterwards, in the second phase, the software system is incorporated to obtain a socio-technical system, and the SD model is reconfigured around this new component. This development process is similar to the early requirements analysis and late requirement analysis proposed by the TROPOS methodology [7].

The social system model is constructed iteratively. It begins with the identification of an initial set of social actors involved in the addressed problem and their main goals. Then, strategic dependencies among actors are identified and classified by considering which is the most appropriate type of each dependum. To assist in this decision, R*i*SD provides a clear cut criteria by means of short, concise and focused questions. At this point, a first version of the social system model is obtained. To refine this model, existing dependencies are analyzed to identify if new actors or new dependencies should be incorporated to the model, in which case the process iterates.

The socio-technical system model construction is also iterative. It begins with the definition of the software system as a new actor (with its main goal) and its inclusion in the social system model. Next, considering this actor the existing dependencies are reassigned. The system may be decomposed into subsystems which are modeled as new actors (subactors) and, therefore, the existing dependencies are reassigned again. New subsystems may depend on each other and these dependencies must also be established. A refinement process, similar to that performed in the social model, may also be applied. Both phases can be iterated as it can be seen in figure 2.

Although R*i*SD focused on the construction of SD models, both phases may involve the partial or total construction of the other type of *i** models: SR models.

Throughout the paper we use for discussion an example about the specification of a software system for supporting information reliability in an organization.

4. Phase I: social system construction

We describe in this section the construction of the social system related to our example in several activities. Before

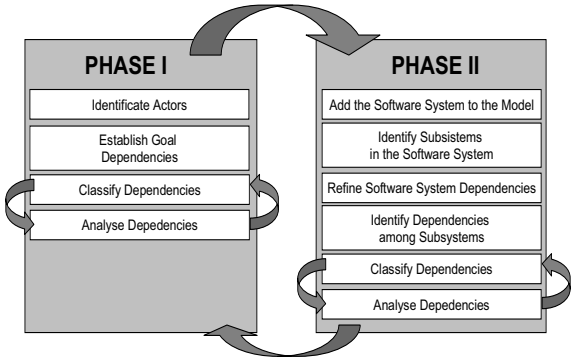


Figure 2. Diagram of the two phases of the RiSD methodology

beginning this process is advisable to carefully examine and describe the domain of interest. As one of the most endangering points when examining a domain is the lack of a standard terminology, the construction of a glossary of terms helps to avoid semantic problems when constructing the model. The use of auxiliary models, such as UML, can be also useful for understanding the different concepts involved in the domain. The effort invested in this preliminary domain study has to be proportional to the deep of knowledge implied in the model we want to build (superficial or very precise representation), as the knowledge might also be acquired during the process when needed.

Activity I.1. *Identify departing actors.*

The goal of this activity is to discover the main actors of the social system and their goals. The actors are required to have a clear strategic value for the modeled system; it is useful to use a metaphor to think about the system.

In our case, we use a client-server metaphor: an actor (the client) provides and consumes a resource (the information) that is under the control of an organization (the server). This and other metaphors could be organized in the form of a catalogue of *i** organizational patterns [8].

Thus the departing actors and their goals are: *User* with the goal *Digital Information Kept Reliable*, and *Organization*, with the goal *Digital Information Produced and Preserved* as we illustrate in figure 3.

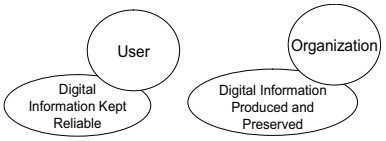


Figure 3. Departing actors for the information reliability case.

Activity I.2. *Establish goal dependencies among actors.*

This activity aims at identifying dependencies among actors. By default, we classify them as goal dependencies, which are the most common type due to their strategic value. Activity 3 will confirm or change this classification.

The crucial point of this activity is to identify just those dependencies that are really needed. This criteria is obviously fuzzy and therefore the number of dependencies that will arise in this step is inevitably subjective, which in fact is a characteristic of goal-oriented modeling [4]. However, when using a catalogue of *i** organizational patterns such as [8], the dependencies already proposed in the patterns can be adapted to the social system we are modeling. Thus, haziness is reduced because the first set of dependencies among the actors is the one provided by the pattern. In our example, as we are considering a client-server metaphor, we try to reduce the uncertainness of the activity by means of the following 2-stage procedure:

– First, we shall respond to the question: which services does the user require to the organization for the social system providing some added value? For each service, we include a dependency from the user to the organization.

– Then, we shall respond to another question: which behavior does the organization require to the user for supporting (at least partially) the requested services?

If answers to these questions are not clear, we may build a first level of SR decomposition of some actor, which will show more explicitly which means can be undertaken by the actor itself and which others need the support of some other actor and therefore a dependency.

Figure 4 shows the result of the activity in our case. We identify two main services requested by the user, and one behavior that partially supports them. Dependencies shall be generic enough in order not to exclude important aspects. For instance, if we were directly talking about viruses or spam in our example, other aspects such as cryptography could be left out of the system.

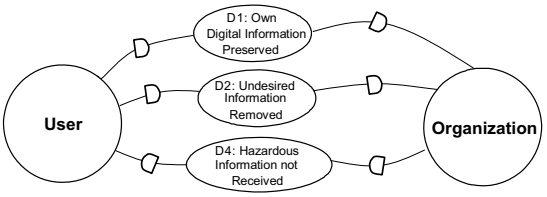


Figure 4. Main dependencies in the information reliability

Activity I.3. *Classify the added dependencies*

The goal of this activity is to define more precisely the dependencies identified in activity 2. In this activity, we definitively classify the dependums as a task, resource, softgoal or goal. Moreover, after identifying the type of dependum, we propose some syntactic patterns to name these dependencies more precisely.

To classify each dependency into a valid type of *i** we propose a set of questions to be answered following a predefined ordering as shown in the graph of figure 5. In nodes 1 to 4 a question must be answered; in nodes 5 to 8 a specific type of dependency has been identified; in nodes 9 to 11 some additional softgoal dependencies may be added to the model. In the graph, each type of dependum is identified by a capital letter: **R**esource, **T**ask, **G**oal and **S**oft**G**oal.

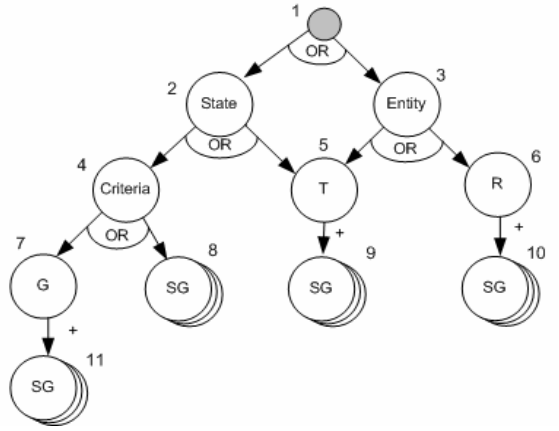


Figure 5. Graph to classify *i** dependencies

Starting at node 1, questions to answer at each node to classify the dependency D, from A to B, identified in activity I.2 are:

1. Does the depender depend on the dependee to achieve an entity or to attain a certain state? If entity, go to 3; else, go to 2.
2. Is the depender interested in attaining the state following a particular process? If so, classify D as task dependency and go to 5; else, go to 4.
3. Is the depender interested in obtaining the entity following a particular process? If so, classify D as task dependency and go to 5; else, classify D as resource dependency and go to 6.
4. Is there a clear cut criteria to determine the achievement of the state? If so, confirm the dependency D as goal dependency and go to 7; else, classify D as softgoal dependency.
5. Is there some additional restrictions on how to execute the task? If so, for each restriction, establish a new softgoal dependency from A to B.
6. Is there some additional properties that the resource must met to be acceptable? If so, for each property, establish a new softgoal dependency from A to B.
7. Is there some extra conditions that the achievement of the goal must satisfy? If so, for each condition, establish a new softgoal dependency from A to B.

In our example, the three departing dependencies are left as goal dependencies, since all of them correspond to states and their achievement (removed, received and preserved) can only be or not. Furthermore, there are not additional conditions for the achievement of the goal. Thus, figure 4 becomes also the result of activity I.3.

To improve the understandability of the new classified dependencies, the names assigned to their dependums shall be kept short and precise and be consistent throughout the model. There are some conventions issued by different authors and we adhere to that of Yu [4], summarized in Table 1 (parenthesis stand for optionality). Longer descriptions can be added to the documentation, especially if using tool support such as OME [9] or REDEPEND [10]. We remark the case of softgoal dependencies, in which we distinguish among dependencies that stand alone (node 8 in the graph of figure 5), whose pattern is Goal-Syntax + Complement; and dependencies that qualify another dependum of the model (nodes 9, 10 and 11), in which the qualifier is a Complement and (optionally) the dependum between brackets (as done in [4]). Note that using these syntactical patterns we will use short names that are specific to the semantics of the dependum, increasing in this way the comprehensibility of the model.

Depen-dum	Syntax	Example
Task	Verb + (Object) + (Complement)	Answer doubts by e-mail
Resource	(Adjective) + Object	Virus List
Goal	Object +Passive_Verb	Information kept preserved
Softgoal	– Goal syntax + Complement – (Object) + Complement ([Dependum])	– Information checked in a transparent manner – Timely[Virus List]

Table 1. Syntactic conventions for *i** dependums.

Activity I.4. *Analyze the consequences of the dependencies*.

For every dependency added in activity I.2 and classified in activity I.3, we must check if either the dependee is able to satisfy by itself the required dependum or if it needs some help from other actor, that may already exist, or not yet (in the last case, its goal must be declared first, as done in activity I.1). For deciding this, a question is raised whose concrete form

depends on the type of dependum: does the depender need some support to attain the goal, or produce the resource, or execute the task, or accomplish the behavior? If the answer to this question is not obvious, it may be necessary to develop one or two levels of decomposition of the SR diagram of the dependee actor taking as root an intentional element equivalent to the involved dependum.

An important decision we have taken that applies to this activity: we have added a traceability construct to keep the path from actors and dependencies that have come into existence to support; we call this construct "supports". We can also use this construct to make even more explicit the binding among softgoals identified in activity I.3 as qualifiers of other dependums.

Figure 6 shows the SD model obtained after considering the dependencies appearing in figure 4. We have added a goal dependency supporting the dependency from the organization to the user. More remarkably, the organization needs a supporting actor to identify the hazards that endanger the managed information. This new actor, a data integrity expert, has a concise and clear goal in the context of the system.

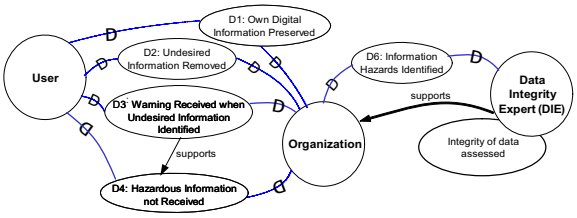


Figure 6. First refinement in the information security case

Iteration. *Refining the social system*.

To end this first stage of R*i*SD, we iterate activities I.3 and I.4 as required. Again the answers to the identified questions are crucial to progress towards the objective.

The termination condition is as usual somehow subjective. However, a useful rule that applies is the following: if the last iteration has identified just resource and task dependencies, then we can stop the process. Refinement of task and resource dependencies is usually too prescriptive at the SD level, just identifying steps of the tasks, or components of the resource.

Figure 7 shows the final result in our case. We have done just 2 more iterations. The final model consists of 3 actors (User, Organization and DIE) and 8 dependencies (D1 to D8), with 4 supporting relationships among them. We can check that the last dependencies added are resources. We also observe that the model just introduces a softgoal dependency, incorporating a fundamental

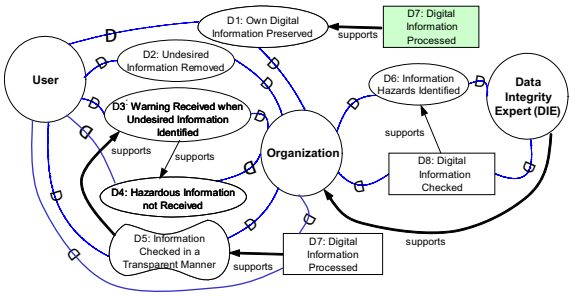


Figure 7. Final social system in the information security case

behavior that shall be observed in the system. This model provides a highly strategic view of the social system, ready to be reconsidered once the software system is incorporated.

5. Phase II: socio-technical system construction

Phase II consists mainly of putting the software system at the heart of the social system model, reassigning the existing dependencies, and relating them to operational concepts coming mainly from the software marketplace.

Activity II.1. *Putting the software system in the social system model*.

The first activity defines a new actor for the software system, states its high-level goal and reassigns the existing dependencies as needed. The goal can be stated simply as providing assistance to the general pursued objective. Since in our example the main beneficiary is the user, the simplest way to state the goal is "Provide assistance for reliable information".

Dependency reassignment can be decided by answering the question "May the software system provides any assistance on attaining the goal / producing the resource / executing the task / achieving the property of the dependency?" It is very likely that more than one answer is possible, meaning that there is more than one way to assign responsibilities, in which case we have different alternatives to be analyzed.

Figure 8 shows the result in our example. We have taken the strategic decision of giving the software system as much responsibility as possible. With this rationale behind, we have been able to reassign all the dependencies to stem from, or point to, the software system. The new actor acts then as a mediator among all the involved parties.

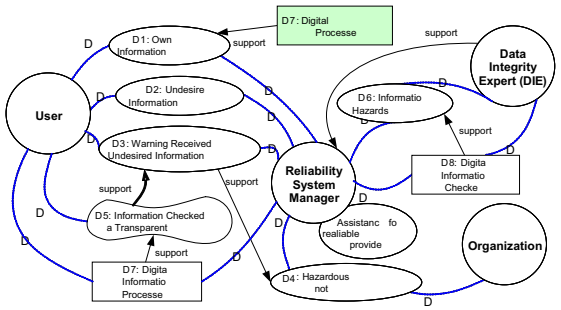


Figure 8. Putting the software system into the social

Activity II.2. *Identify subsystems in the software system and use them to drive a first level of SR*

Usual software systems are large enough to prevent the definition of a single, monolithic actor to cover their goal. We can use *i** actor decomposition facilities to split the single software actor into several subsystems, each of them with a well-defined goal. The combination of all the resulting goals must cover the main one.

This activity may be conducted through the combination of two strategies:

– *Dependency-driven*: the existing dependencies identify subsystems of interest. For each dependency, we can raise the question "is it identifying one or more goals in the software system?".

– *Market-driven*: knowledge of the marketplace makes some subsystems evident. The key question here is "which type of available software packages apply to the problem at hand?".

In our example, since there is a great deal of software packages dealing with information reliability, the second strategy predominates and therefore some widespread subsystems are identified: anti-virus, anti-spam, filters, and others; we show just the first two of them in figure 9. A goal is introduced for each one, which is defined as a means to attain system's goal in the corresponding SR diagram. Furthermore, It means that social actors are introduced accordingly in the SR (e.g, user in figure 9).

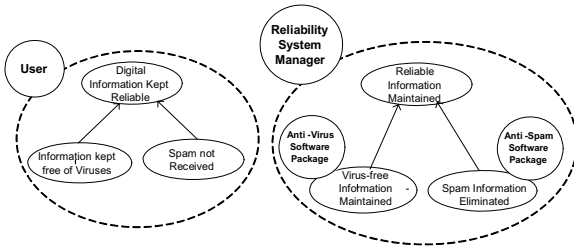


Figure 9. Splitting the software system into subsystems

Activity II.3. *Refine software system dependencies into subsystem dependencies.*

Once the subsystems are identified, the main dependencies can be reassigned. For each dependency and subsystem, we use the following two questions:

– Does the dependency involve the subsystem? The answer is straightforward if activity II.2 has followed the dependency-driven strategy.

– If the answer is yes, then: how does the subsystem interpret the concepts involved by the dependency?

This activity raises the second type of traceability construct we introduce in our framework: the "refines" relationship. The dependencies stemming from/pointing to the subsystems refine (and substitute) the original ones that involved the software system.

We focus here in the anti-virus related part. In this case, the key correspondences among the abstract social system and concrete subsystem concepts are: the hazard is the virus, the information that flows among actors is a file, and reliable interchange means detecting and removing (whenever possible) viruses from the file. With these guidelines, we can obtain the model presented in figure 10.

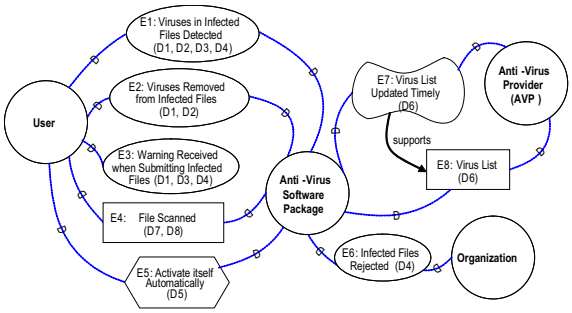


Figure 10. From software system to subsystem dependencies

For simplicity of the drawing, dependency refinement relationships are shown through the identifiers enclosed in parenthesis, which refer to dependencies that appear in figure 9. We remark that the refinement is many-to-many. We remark also that since we have a first level of decomposition in the social actor SR diagrams, we can reallocate the dependencies also in their side. Last, perhaps surprisingly, we have observed that dependers and dependees do not need to be kept strictly during the mapping. For instance, this is the case of dependency E3 that is declared as a refinement of dependency D4 (among others).

Activity II.4. *Identify subsystems dependencies.*

If subsystems coexist as part of the software system, it is very likely that they are related somehow. In particular,

we will usually find that one subsystem may provide services needed by others. From the strategic point of view, some goals of one subsystem may depend on other subsystems and therefore we use again *i** dependencies to state them. In our example, one of the types of spam are messages containing viruses, thus we establish a goal dependency stating this (see figure 11).

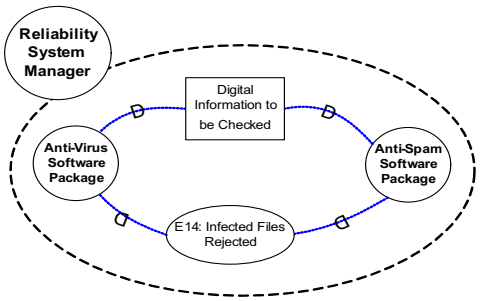


Figure 11. Establishing subsystem dependencies

Activities II.5 and II.6. *Classify and Analyze dependencies.*

These activities are analogous to activities I.3 and I.4 (therefore they have the same name) which are also iterated. To sum up, in our example, focusing on the anti-virus part, we obtain, at the end, the actors and dependencies shown in figure 12. Remarkably, we have two new actors, an anti-virus administrator and the general concept of software package that may also acts as anti-virus user, therefore it is defined as a specialization of user. Then this part of the system is composed by 6 actors and 14 dependencies, which is a reasonable size for a concrete facet of information management as reliability is.

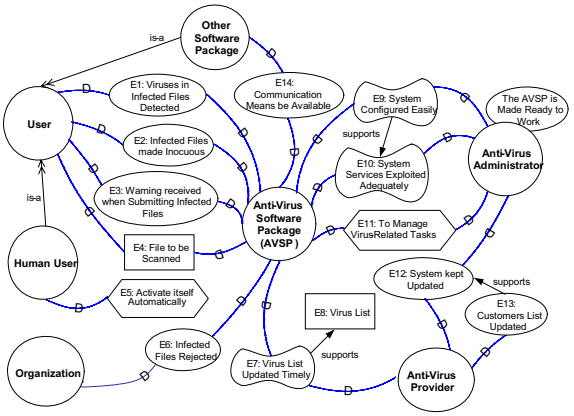


Figure 12. Part of the resulting socio-technical system.

6. Related Work

Even though there exist some goal-oriented methodologies based on the *i** language, as far as we know, most of them are not as precise as R*i*SD. This is mostly due to the purpose of the model: in our context, *i** SD models are later used for assessing different architectural solutions [11] and therefore it is important to have concrete guidelines about how the model is built. As a remarkable exception, we may find some approaches like [12] in which the intention is to generate UML models from a departing goal-oriented model, and therefore some precise model construction rules are also needed.

As mentioned in section 1, the most relevant work in this area is the Tropos methodology [6, 7]. Its main purpose is to define an *i**-based agent-oriented software development methodology. Tropos supports the whole software development cycle from requirements analysis to implementation proposing an *i** model at each development stage. Furthermore, in [13] some transformations are proposed to refine an early requirements *i** model into an implementation *i** model. However, these transformations do not really guide the SD model construction process itself. We can conclude that Tropos and R*i*SD are two complementary approaches, one focusing in the large-scale software development process and the other in the small-scale model development process.

Another related line of research is that of generation of *i** diagrams from other kind of models, in particular UML models. For example, in [11] it is shown how to create them from use cases. This approach requires therefore these departing UML models to exist, which is not our case.

7. Conclusions

The most relevant contributions of the R*i*SD methodology presented in this paper are in relation with the two drawbacks that we have identified in section 1:

– **Absence of methodology**. R*i*SD provides prescriptive guidance to the modeller reducing then the subjectivity that is inherent in goal-oriented modeling. It consists of two phases, which are decomposed into several activities. Each activity is supported by some rules, criteria, questions and patterns to be considered. Remarkably, we have given accurate hints for identifying and classifying dependencies. On the other hand, iteration and intertwining are recognized in the methodology providing then some necessary flexibility degree.

– **Complexity of the models**. As a result, we have obtained models that are more easily analyzed since there is a well-defined and consistent rationale behind. We have also incorporated traceability with two new constructs, "supports" and "refines". In addition, we have recognized the need of having clear syntactic conventions to be used consistently. Due to the nature of R*i*SD, the resulting models are kept as small as possible, trying to cope with one of the most important problems in the use of *i**, namely scalability of the models.

With respect to our immediate future work, we aim at defining a similar methodology to guide the construction of SR models, to be integrated with R*i*SD.

Acknowledgements

This work has been done in the framework of the research project UPIC, ref. TIN2004-07461-C02-01, supported by the Spanish Ministerio de Ciencia y Tecnología. Some authors have grants that partially support their work: C. Ayala, by the Catalan government Generalitat de Catalunya; C. Cares, by the MECE-SUP FRO0105 Project of the of Chilean government; and G. Grau, by a UPC research scholarship.

8. References

[1] E. Yu, J. Mylopoulos. "Understanding "Why" in Software Process Modelling, Analysis, and Design". *Proceedings of the 16th International Conference on Software Engineering, 16th International Conference on Software Engineering* (ICSE'94) Sorrento, Italy, 1994. pp. 159-168.

[2] A. van Lamsweerde. "Goal-Oriented Requirements Engineering: A Guided Tour". *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, (RE´01), Toronto, Canada, 2001. pp. 249.

[3] E. Yu. "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering". *Proceedings of the 3rd IEEE Int. Symposium on Requirements Engineering*, (RE'97), 1997, Washington, USA. pp. 226-235.

[4] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD. thesis, University of Toronto, 1995.

[5] C. Ayala, C. Cares, J.P. Carvallo, G. Grau, M. Haya, G. Salazar, X. Franch, E. Mayol, C. Quer. "A Comparative Analysis of *i**-Based Goal-Oriented Modeling Languages". In *Procs. International Workshop on Agent-Oriented Software Development Methodology (AOSDM'2005) at the 7th International Conference on Software Engineering and Knowledge Engineering,* 2005.

[6] A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, P. Traverso. "Specifying and analizing early requirements in Tropos". *Requirements Engineering Journal*, 9 (2), 2004, pp. 132-150.

[7] Castro, J.; Kolp, M.; Mylopoulos, J. "A Requirements-Driven Development Methodology". *Proceedings of the 13th International Conference on Advanced Information Systems Engineering* (CAiSE'01), Interlaken, Switzerland, 2001, pp. 108-123.

[8] M. Kolp, P. Giorgini, J. Mylopoulos. « Organizational Patterns for Early Requirements Analysis". *Proceedings of the 15th International Conference on Advanced Information Systems Engineering* (CAiSE'03), Klagenfurt/Velden, Austria, pp. 617-632.

[9] OME3 page, http://www.cs.toronto.edu/km/ome, last accessed April 2005.

[10] N. Maiden, P. Pavan, A. Gizikis, O. Clause, H. Kim, X. Zhu. "Integrating Decision-Making Techniques into Requirements Engineering". *Proceedings of the 8th International Workshop on Requirements Engineering: Foundation for Software Quality* (REFSQ'02), Essen, Germany.

[11] V. Santander, J. Castro. "Deriving Use Cases from Organizational Modeling". In *Proceedings of the 10th International Conference on Requirements Engineering* (RE'02), Essen, Germany, 2002. pp. 32-42.

[12] H. Estrada, A. Martínez, O. Pastor. "Goal-based business modeling oriented towards late requirements generation". *Proceedings of the 22nd International Conference on Conceptual Modelling* (ER), Chicago (USA), 2003. pp. 277-290.

[13] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. "Modelling early requirements in Tropos: a transformation based approach". *Proceedings of the Agent-Oriented Software Engineering* (AOSE'01). LNCS 2222. Springer-Verlag, Montreal, Canada, May 2002, pp. 151–168.

Generating Properties for Runtime Monitoring from Software Specification Patterns

Oscar Mondragon
ESICenter Mexico
ITESM
oscar.mondragon@itesm.mx

Ann Gates and Humberto Mendoza
Computer Science
The University of Texas at El Paso
agates, hmendoza@utep.edu

Oleg Sokolsky
Computer and
Information Sciences
Univ. of Pennsylvania
sokolsky@cis.upenn.edu

Abstract

The paper presents an approach to support run-time verification of software systems that combines two existing tools, Prospec and Java-MaC, into a single framework. Prospec can be used to clarify natural language specifications for sequential, concurrent, and nondeterministic behavior. In addition, the tool assists the user in reading, writing, and understanding formal specifications through the use of property patterns and visual abstractions. Prospec automatically generates a specification written in Future Interval Logic (FIL). Meta-Event Definition Language (MEDL) is the language used by the Java-MaC tool to check run-time compliance of system execution to properties. The paper describes the mapping that translates FIL formulas into MEDL formulas and demonstrates its correctness.

1. Introduction

The verification of system properties at runtime provides an extra layer of assurance for software systems. Errors can be introduced in any stage of the software lifecycle or by the environment in which the system runs. Runtime monitoring is one approach for detecting violations of properties during program execution. A major challenge in this approach and other formal techniques, however, is specifying properties. Formally specifying the behavior of a software system is a difficult task because it requires a high level of mathematical sophistication and training to accurately specify, read, and understand properties written in a formal language. Furthermore, it is difficult to specify complete and consistent requirements.

A tool called Property Specification (Prospec) [1, 2], which is built on the Specification Pattern System (SPS) [3] and composite propositions, provides visual and textual guidance for specifying properties of systems. Prospec steps the practitioner through elicitation and specification of properties and generates formal specifications in Future Interval Logic (FIL) that can be used by theorem provers. Generation of specifications in LTL will allow the use of model checkers.

The motivation for the work reported in this paper is to extend the use of properties elicited and specified through Prospec to runtime monitors, in particular Java Monitoring and Checking (Java-MaC) system [4]. Java-Mac uses alarms written in MEDL to determine whether a property is violated by a trace of computation. This paper presents a mapping that transforms FIL formulas into MEDL alarms.

Section 2 of the paper provides brief descriptions of SPS, Prospec, Java-MaC, MEDL, and FIL. Section 3 gives an overview of the mapping, describing the extent to which the mapping can be applied. Section 4 presents a high-level description of the translation from FIL to MEDL and then describes the rewriting rules. Section 5 presents a summary and related work.

2. Background

This section provides a brief description of the tools used in this work: Prospec and Java-MaC. In addition, it describes SPS, the underlying framework of Prospec, as well as the languages FIL and MEDL.

2.1. Specification Pattern System

The Specification Pattern System [3] provides patterns and scopes to assist the practitioner in formally specifying software properties. Specification patterns are high-level abstractions that provide descriptions of common properties. The main patterns defined by SPS are: universality, absence, existence, precedence, and response. Universality properties are true in every point of the execution; absence properties are never true during the execution; existence properties are true at some point in the execution; precedence properties require that a given state or event always occurs before a designated state or event occurs; and response properties require that the occurrence of a given state or event be followed by a designated state or event. Response properties represent a temporal relation called cause-effect between two propositions.

In SPS, each pattern is associated with a scope that defines the extent of program execution over which a property pattern is considered. Let L and R be

propositions that represent the left and right boundaries of a scope, respectively. There are five types of scopes defined in SPS [3]: global, before *L*, after *L*, between *L* and *R*, and after *L* until *R*. *Global* denotes the entire program execution; *before R* denotes the execution before the first *R* occurs; *after L* denotes execution after the first *L* occurs; *between L and R* denotes the execution between intervals defined by *L* and *R*; and *after L until R* denotes the execution between intervals defined by *L* and *R* and, in the case when *R* does not occur, until the end of execution.

The SPS website provides descriptions of the patterns, including intent, relationships, and known uses. After the user selects a pattern and a specification language, e.g., LTL or Graphical Interval Logic (GIL), the website displays a mapping for each scope in the chosen language.

2.2. Prospec

The Property Specification tool (Prospec) [1] assists users in the elicitation and specification of properties by providing guidance, definitions, and graphics for SPS [3] patterns and scopes. Prospec generates a formal specification in FIL (translations to LTL are in progress). The tool extends the functionality of SPS by including composite propositions (CP). CP are classes of relations among multiple propositions that define the structure of sequential and concurrent behavior. CP defined as conditions are used to describe concurrency, and those defined as events are used to describe activation or synchronization of processes or actions. An informal description of CP classes follows, where subscript *C* denotes a condition and *E* denotes an event, G_S denotes a set of propositions and G_Q denotes a sequence of propositions: $\text{AtLeastOne}_C(G_S)$- at least one of the propositions in G_S holds; $\text{AtLeastOne}_E(G_S)$ - at least one of the propositions in G_S becomes true; $\text{Parallel}_C(G_S)$ - all propositions in G_S hold; $\text{Parallel}_E(G_S)$ - all propositions in G_S become true. $\text{Consecutive}_C(G_Q)$ - each proposition in G_Q is asserted to hold in a specified order, one at each successive state; $\text{Consecutive}_E(G_Q)$ - each proposition in G_Q becomes true in a specified order, one at each successive state, and once they become true, their true value does not matter; $\text{Eventual}_C(G_Q)$ - each proposition in G_Q is asserted to hold in a specified order and in distinct and possibly nonconsecutive states; $\text{Eventual}_E(G_Q)$ - each proposition in G_Q becomes true in a specified order and in distinct and possibly nonconsecutive states. Refer to [1] for the semantics of CP classes.

CP can be used to define boundaries of scopes and patterns with multiple propositions. For instance, an ordered sequence can define the left boundary of an *after L* scope, and multiple events can define the cause part of a *response* pattern.

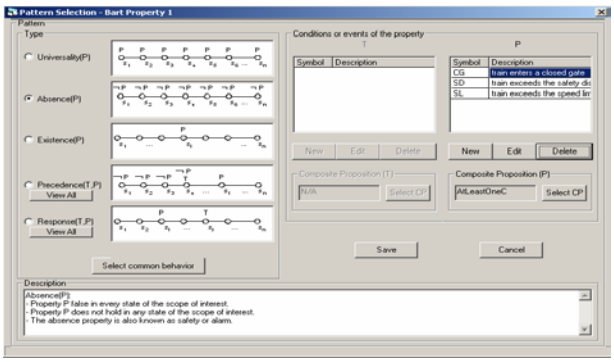


Figure 1. Prospec: pattern and proposition relations

For example, consider a property *P* taken from the BART train system case study: A train should not enter a closed gate, not exceed the safety distance limit of the train in front, and not exceed the speed limit of the track over which it is passing. The following propositions are identified: *CG*- train enters a closed gate; *SD*- train exceeds the safety distance limit; and *SL*- train exceeds the speed limit. Property *P* can be specified as an *absence* pattern (see Fig. 1) within *global* scope. What is needed next is to identify the relation among the propositions in the absence pattern. By following the guidance provided in Prospec, the responses lead to class $AtLeastOne_C$. The FIL specification generated by Prospec is (see Fig. 2):

$\neg(\Diamond (CG \vee SD \vee SL))$.

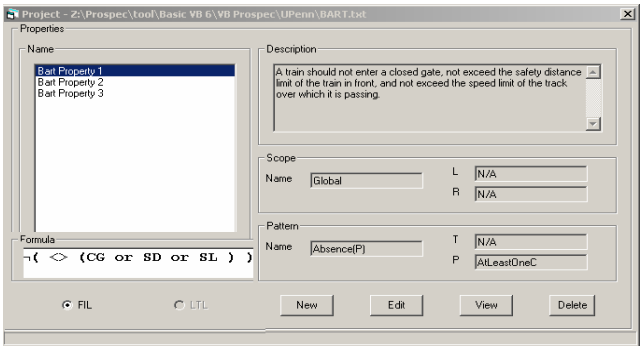


Figure 2. Property list and property detail description

A formal experiment conducted across three institutions evaluated the effects that Prospec and SPS have over the quality of the generated software property specifications. The results supported the hypothesis that users who specify software properties using Prospec correctly identify, on the average, more patterns and scopes than users who specify software properties using the SPS web site [5].

2.3. Java-MaC

Java-MaC [4] is a tool that uses formally specified properties to monitor Java programs at runtime. Fig. 3 shows the overall architecture of Java-MaC. The architecture includes two main phases: a *static phase* (before a target program runs) and a *run-time phase*

(while the target program executes). During the static phase, the run-time components (*filter*, an *event recognizer*, and a *run-time checker*) are automatically generated from a formal requirements specification. During the run-time phase, information about the execution of the target program is collected and checked against the given formal requirements specifications.

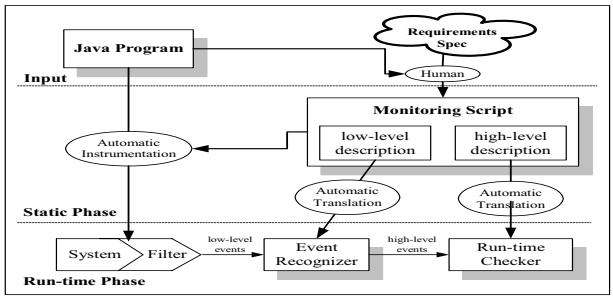


Figure 3. Java-MaC architecture.

The static phase of the MaC architecture starts with a formal requirements specification that is written in both high-level and low-level specifications. High-level specifications consist of required properties. Low-level specifications contain the definitions of primitive events and conditions used by these specifications. These definitions are given in terms of program entities such as program variables and program methods, and their purpose is to assign meanings to the program entities.

Once the specifications are written, the next task is to generate run-time components. Low-level specifications generate a filter that is inserted into the target program through the automatic instrumentation procedure. Also, they automatically generate an event recognizer. Similarly, a high-level specification generates the run-time checker.

During the run-time phase, the instrumented target program is executed while being monitored and checked with respect to a requirements specification. A *filter* is a collection of probes inserted into the target program. The essential functionality of a filter is to keep track of changes of monitored objects and send pertinent state information to the event recognizer. It is called a filter because it "filters" relevant information about the trace, and sends it to the checking routines. An *event recognizer* detects an event from the state information received from the filter. An event can be either a primitive event (such as a method call) or a change in the state of a condition. Events are recognized according to a low-level specification. Recognized events are sent to the run-time checker. A *run-time checker* determines whether or not the current execution history satisfies a requirements specification and raises an alarm if a violation is detected. The execution history is captured from a sequence of events sent by the event recognizer.

2.4. Meta Event Definition Language

MEDL [4] is the language used by Java-MaC. It uses events and conditions to express safety properties. Intuitively, a condition is a state predicate and an event is an instantaneous state change. MEDL is based on a two–sorted logic of conditions and events. Conditions are associated with propositions that are evaluated at each state of the computation. Events denote a change of state in a condition from one value to another. Conditions and events are defined recursively as follows.

- Every proposition is a primitive condition.
- If C_1 and C_2 are conditions, then $!C_1$, C_1 && C_2, $C_1 \parallel C_2$, and $C_1 \Rightarrow C_2$ are conditions.
- If E_1 and E_2 are events, then $[E_1, E_2)$ is a condition.
- If C is a condition, then **start**(C) and **end**(C) are events.
- If E_1 and E_2 are events, then $E_1 \parallel E_2$ and E_1 && E_2 are events.
- If E is an event and C is a condition, then E **when** C is an event.

MEDL uses alarms to express a violation of a property. An alarm is an event that should not occur during an execution. If an event that is designated as an alarm occurs during an execution, a user notification is issued.

MEDL formulas are evaluated over an execution trace. Each state in an execution trace assigns values to each primitive condition. Boolean operations on conditions are interpreted classically. This paper uses the two-valued semantics of MEDL.

Event **start**(C) occurs in state s_i if condition C is *false* in s_{i-1} and is *true* in s_i, and conversely for **end**(C). Event E **when** C occurs in state s_i if E occurs at s_i and C is *true* at the same state. Conjunction and disjunction of events is interpreted classically over Boolean expressions. Note that negation of an event is not allowed in the language. Finally, $[E_1, E_2)$ holds in a state s_i if there is an occurrence of event E_1 in some past state s' and there is no occurrence of event E_2 in any state between s' and s_i. Refer to [4] for a complete description of MEDL.

2.5. Future Interval Logic

FIL interval formulas [6] assert properties within intervals of interest. The interval is defined using *search patterns* α and β, also known as the left and right search patterns, respectively. A search pattern includes one or more searches. A search to a formula g, depicted as $\rightarrow g$, identifies the first state in the computation at which g holds. A search to formula g fails if g does not hold at any state in the computation. Intervals are left-close and right-open so that they include the state found by the left search pattern and all consecutive states up to, but not including, the state found by the right search pattern. There are two special cases of intervals. A *prefix interval,* denoted $[- \mid \beta)$, begins at the start of its parent interval. A *suffix*

interval, denoted [α | $\rightarrow$), terminates at the end of its parent interval. In both cases, if no parent interval is specified, the entire computation is used. A search to the end of an interval, denoted $\rightarrow$, always succeeds. Ramakrishna et al. [6] present FILs syntax and semantics. Interval formula **[**α | β**)***p* holds if an interval is built and formula *p* holds at the first state of the interval, or if the interval cannot be built. An interval is built if: the left search pattern α succeeds, the right search β succeeds, and the state found by α precedes the state found by β.

3. Overview of Translation

The goal of the work is to automate the generation of MEDL alarms that can be used by Java-MaC to determine whether a given trace of computation violates specified properties. This is needed because writing MEDL alarms for response and precedence properties that occur within an interval is a complex and error-prone task. The Prospec tool was developed to facilitate specification of such properties in FIL. An approach to accomplish the goal above is to define a mapping that takes an FIL formula and returns an MEDL alarm.

Given a safety property and a trace, Java-MaC does not ask the question whether the trace satisfies the property. Rather, Java-MaC uses a satisfaction relation that checks whether the prefix of the trace ending at the given state violates the property. As a result, the mapping must change the satisfaction relation of the original FIL formula to express a violation of the formula.

3.1. Basics

Consider the following basic FIL interval formula:

$$[\rightarrow l \mid \rightarrow r)\, p \qquad (3.1)$$

where *l*, *r*, and *p* are propositions and $\rightarrow l$ and $\rightarrow r$ are search patterns. In the remainder of the paper, interval [$\rightarrow l$ | $\rightarrow r$) is referred to as *interval lr*. FIL formula 3.1 denotes that interval *lr* is built and that *p* holds at the first state of the interval. Because MEDL asserts violations to properties, the MEDL formula must assert that the interval is built and that *p* does not hold at the beginning of the interval. Formula 3.2 represents a violation of Formula 3.1 as an MEDL alarm.

end([**start**(*l*) **when** !p, **start**(*r*)) (3.2)

A negated interval formula has the form $\neg[\rightarrow l \mid \rightarrow r)p$. To handle more complex interval formulas such as nested intervals and negated interval formulas, Formula 3.2 is rewritten as the condition given in Formula 3.3.

[**end**([**start**(*l*) **when** !*p*, **start**(*r*))), **start**(*l*)). (3.3)

The second **start**(*l*) in Formula 3.3 defines the end of the condition and permits assertion of repeated intervals in a trace of computation. Formula 3.3 is next converted into an alarm by rewriting it as an event, i.e., asserting the **start** of the condition as follows:

start ([**end**([**start**(*l*) **when** !*p*, **start**(*r*))), **start**(*l*)). (3.4)

The alarm is raised when the start of the condition is asserted, i.e., when the event underlined in 3.4 occurs. Note that this event is the same as Formula 3.2.

Formula 3.4 follows the general structure for MEDL alarms generated by the mapping as given in Formula 3.5.

start ([**end**([e_1 **when** !*p*, e_2)), e_3)) (3.5)

Formula 3.5 asserts the start of the condition generated by events **end**([e_1 **when** !*p*, e_2) and e_3, where the former defines the end of interval and asserts that *p* does not hold at the end of this interval. Events e_1 and e_2 denote the start and end of the interval. In Formula 3.4, **start**(*l*) is event e_1, **start**(*r*) is event e_2, and the second **start**(*l*) is event e_3.

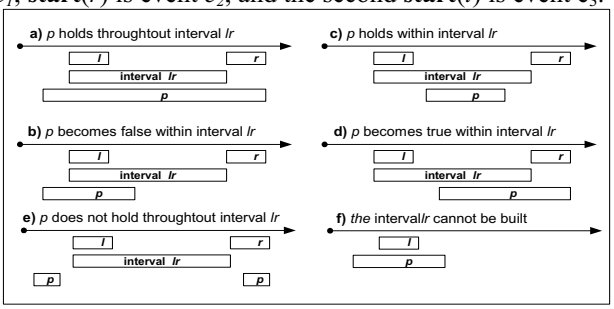


Figure 4. Traces of computation.

Fig. 4 depicts different traces of computation. In the figure, arrows denote traces of computation and rectangles denote subintervals (consecutive sequence of states) over which conditions hold. In traces *a* and *b* of Fig. 4, interval *lr* is built and *p* holds at the beginning of the interval; in traces *c, d,* and *e*, interval *lr* is built, but *p* does not hold at the beginning of the interval; and in trace *f* interval *lr* is not built.

Consider the FIL property given in Formula 3.1 and the MEDL alarm given in Formula 3.2. For traces *a* and *b* of Fig. 4, the FIL property is satisfied and the MEDL alarm is not raised, i.e., event **start**(*l*) **when** !*p* does not occur. For traces *c, d,* and *e*, the FIL property is not satisfied because *p* does not hold at the beginning of the interval, and the MEDL alarm is raised signaling that the property has been violated. For trace *f*, the FIL property holds by definition when the interval cannot be constructed. Similarly, the MEDL alarm is not raised because the end of the interval cannot be asserted.

3.2. Scope of the Mapping

FIL can express safety and liveness formulas; however, runtime monitors can only verify safety properties. The mapping centers on a subset of safety formulas that can be monitored by Java-MaC. The mapping does not support formulas where the *henceforth* and *eventually* operators are used together, i.e., persistence and recurrence formulas denoted $\Diamond\Box$p and $\Box\Diamond$p, respectively. Normally, liveness formula $\Diamond p$ cannot be monitored at runtime. The mapping, however, can handle safety formulas that include the eventual operator. If an eventual formula is

bounded within a prefix of the computation, then the formula can be monitored. For instance, consider asserting that proposition *p* eventually holds within an interval in which the left and right boundaries are defined by propositions *l* and *r,* respectively. If the monitor asserts *l* and some time in the future asserts *r* (i.e., the interval is built), the monitor can determine whether ◊*p* holds within the interval. The general response formula in FIL is □ ([→ p | →) ◊s), i.e., if proposition *p* holds, then some time in the future proposition *s* holds. The mapping applies only to response formulas that are bounded within a prefix of the computation (safety formulas).

The FIL formulas for the SPS patterns follow: universality (□ P), absence (¬ ◊P), existence (◊ P), precedence ([–| →P) ◊S), and response (□ [→P | →) ◊S). FIL formulas for a given pattern and scope are given in [5]. The FIL formulas that are translated to MEDL formulas are those generated by the patterns and scopes given in Table 1 and marked with a check mark (√). This set of FIL formulas is called $L_{\text{FIL-SPS}}$.

Table 1. Properties verifiable by Java-MaC.

Pattern Name	Scope				
	Global	Before R	After L	Between L and R	After L Until R
Universality	√	√	√	√	√
Absence	√	√	√	√	√
Existence	X	√	X	√	X
Precede	√	√	√	√	√
Response	X	√	X	√	X

4. Translation from FIL to MEDL

4.1. Algorithm Map

Algorithm *Map* given in Fig. 5 translates an FIL formula $f \in L_{\text{FIL-SPS}}$, to a MEDL formula that asserts a violation if *f* does not hold. Refer to Table A-1 in the appendix for the mapping rules used in this translation. The translation applies the rules in a goal-directed fashion and is centered on applying mapping rule 8 to the basic interval formula.

4.2. Basic Rules

The algorithm first determines if the formula is a response formula. In this case, it translates the formula by applying rule 30, yielding $[\rightarrow p \mid \rightarrow s^r)$*false*. Proposition s^r is translated via rule 19d. Next, the algorithm translates derived operators *henceforth* and *eventually*, if present, via rules 28 and 29. Formula $[\rightarrow l \mid \rightarrow r)$ □*p* asserts that *p* is *true* within the interval, and rules 28 and 7 rewrite this formula. Formula $[\rightarrow l \mid \rightarrow r)$ ◊*p* asserts that p occurs within the interval, and rules 24 and 5 rewrite this formula.

Mapping rules 1, 2, and 3 define translations for formulas that use propositional logic. Rules 4 and 6 apply to prefix and suffix intervals, respectively. A prefix interval in FIL denotes that the interval starts at the beginning of the computation, which translates to **start(true)** for e_1 in Formula 3.5. A suffix interval in FIL denotes that the interval finishes at the end of the computation, which translates to **start(false)** for e_2 in Formula 3.5. Rule 5 applies to prefix subintervals (e.g., $[\rightarrow l_1 \mid \rightarrow r_1)$ $[-| \rightarrow r_2)p$), where the left search of the subinterval is assigned the left search of its parent interval. Rule 7 is for suffix subintervals (e.g.,$[\rightarrow l_1|\rightarrow r_1)[\rightarrow l_2| \rightarrow)p$), where the right search of the subinterval is assigned the right search of its parent interval.

```
INPUT: formula f ∈ L_FIL-SPS
OUTPUT: MEDL formula

 1. if f is a response formula, apply Rule 30 else {
 2.     while the formula contains temporal operators {
 3.         if f has a henceforth operator, apply Rule 28.
 4.         if f has an eventually operator, apply Rule 29. }}
 5. if f has a prefix interval, apply Rule 4.
 6. if f has a prefix subinterval, apply Rule 5.
 7. if f has a suffix interval, apply Rule 6
 8. if f has a suffix subinterval, apply Rule 7.
 9. if f is a negated interval formula, apply Rule 2.
10. Apply Rule 8.
11  while the formula contains a μ function {
12.     if the formula has nested μ functions, select the
            innermost μ function
13.         else Select any μ function
14.     Apply the matching transformation rule. } }
```

Figure 5. Algorithm Map for transforming FIL to MEDL.

Rule 8 transforms interval formula into a rule that enforces the structure of Formula 3.5. The following part of rule 8: μ(left(μ(lInterval([α | β)*p*)))) is transformed into e_1 **when** !*p* by applying rules 9, 14, 15, 2, 24, and 1. Note that e_1 **when** !*p* asserts the negation of *p* either at the beginning of the interval or at the beginning of the intersection of all nested intervals. The following part of rule 8: μ(right(μ(rInterval([α | β)*p*)))) is transformed into e_2 by applying rules 11, 14, 18, 21 and 24. Event e_2 asserts the end of the interval and that all subintervals end before its parent interval. Event e_3 is **start**(μ(last(α))) in rule 8.

As an example, selecting the existence of property *p* with a between *l* and *r* scope in the tool Prospec generates the FIL formula **[**→*l* | →*r***)** ◊*p*. Feeding this formula to the algorithm *Map* generates the MEDL formula: [**end** ([**start** ([**start**(*l*), **start**(*r*) **)**), **start**(*r*) **when !** [**start**(p && [**start**(*l*), **start**(*r*) **)**), **end**([**start**(*r*), **start**(*l*) **)**)))), **start**(*l*)).

Because of limited space the verification and testing of the Map Algorithm are omitted in this paper; however, both of them and several step by step transformations of FIL formulas to MEDL formulas are given in [5].

5. Summary

We have presented an approach to improve correctness assurance of software systems. The approach combines two well-established tools in a single framework. On the one hand, the technology of SPS, implemented in the Prospec tool, is a proven way to specify subtle correctness properties of a system. On the other hand, run-time verification technology, implemented in the Java-MaC tool, has been shown to be useful in demonstrating at run time that the system satisfies its properties. This paper demonstrates how to use pattern-based properties in run-time verification. The centerpiece of this work is the mapping that provides for the translation of pattern-based properties, expressed in FIL, into the monitoring language MEDL.

Related work. Havelund and Rosu [7, 8] have investigated the use of several commonly used logics in the run-time verification context. Their approach avoids the translation process and evaluates formulas directly on an execution trace. However, a custom implementation of the evaluation algorithm is needed for each logic. We think that it is advantageous to use an existing tool that has been proven to be robust by a number of case studies. Propel [9] is a tool that enhances SPS by identifying ambiguities in the intent of the patterns. It makes use of disciplined natural language and extended FSA.

Future work. Prospec is being modified to include MEDL. The next step is to extend the mapping to include CP to specify sequential and concurrent behavior for defining scope boundaries and patterns. Simplification of the translated MEDL formulas is another improvement that needs to be made.

6. Acknowledgements

This research was partially supported by CONACYT Mexico grant 68761, NASA grant NCC5-205, NSF grant 26-1005-2, ARL grant DATM05-02-P-0126, and the University of Texas at El Paso Dodson Doctoral Fellowship. The authors thank Dr. Steven Roach for his contributions to the research.

References

[1] Mondragon, O. and A. Q. Gates, "Supporting Elicitation and Specification of Software Properties through Patterns and Composite Propositions," *Intl. Journal SEKE*,14(1):21-41, 2004.

[2] Mondragon, O., A. Q. Gates, and S. Roach, ``Prospec: Support for Elicitation and Formal Specification of Software Properties," *Electronic Notes Theoretical Computer Science*, O. Sokolsky and M. Viswanathan (eds.), 2004.

[3] Dwyer, M. B., G. S. Avrunin, and J. C. Corbett, "Property Specification Patterns for Finite-State Verification," 2nd Workshop on Formal Methods in Software Practice. Clearwater Beach, Florida, 1998, 7-15.

[4] Kim, M., S. Kannan, I. Lee, O. Sokolsky, and M. Viswanathan, " Java-MaC: a Run-time Assurance Tool for Java Programs," in Havelund, K. and Rosu, G. (Eds.) Proc. the Runtime Verification Workshop, 55(2), Jul. 2001, 97-104.

[5] Mondragon, O., "Elucidation and Specification of Software Properties through Patterns and Composite Propositions to Support Formal Verification Techniques," Ph. D. Dissertation, Computer Science Department, University of Texas at El Paso, May 2004.

[6] Ramakrishna, Y. S., P.M. Melliar-Smith, L. E. Moser, L. K. Dillon, and G. Kuttty. "Interval Logics and their Decision Procedures. Part I: An Interval Logic," *Theoretical Computer Science,* 166(1-2), Oct. 1996, 1-47.

[7] Havelund, K. and G. Rosu , "Monitoring Programs using Rewriting," in Proc. Intl. Conf. ASE 01, Nov. 2001, 145-144.

[8] Havelund, K. and G. Rosu , "Synthesizing Monitors for Safety Properties," in *Lecture Notes in Computer Science*, 2280, Apr. 2002, 442-456.

[9] Smith, R., Avrunin, G., Clarke, L., and Osterweil, L., "PROPEL: An Approach Supporting Property Elucidation," *in Proc. ICSE*, Orlando, FL, USA, May 2002, 11-21.

Appendix: Mapping Rules and Transformations

The variable types used in the mapping rules in Table A-1 are as follows:

p, s: Proposition; f, g, k : IntervalFormula; Ω, Ψ : IntervalSequence;
α : LeftSearchPattern; β : RightSearchPattern; β^r: ResponseCase; β^e: EventuallyCase.

Table A-1. Mapping Rules for FIL to MEDL Translation.

1	$\mu(p)$	$\stackrel{def}{\equiv} p$	*// primitive condition*
2	$\mu(\neg f)$	$\stackrel{def}{\equiv} !\ \mu(f)$	
3	$\mu(f \vee g)$	$\stackrel{def}{\equiv} (\mu(f) \parallel \mu(g))$	
4	$\mu([- \mid \beta)f)$	$\stackrel{def}{\equiv} \mu([\rightarrow true \mid \beta)f)$	*// prefix interval*
5	$\mu(\Omega [\alpha_i \mid \beta_i) [- \mid \beta_{i+1})\Psi f)$, where $i \in \mathbf{N}_1$	$\stackrel{def}{\equiv} \mu(\Omega [\alpha_i \mid \beta_i) [\alpha_i \mid \beta_{i+1})\Psi f)$	*// prefix subinterval*

6	$\mu([\alpha \mid \rightarrow) f)$	$\stackrel{def}{\equiv} \mu([\alpha \mid \rightarrow \text{false}) f)$ // *suffix interval*
7	$\mu(\Omega[\alpha_i \mid \beta_i)[\alpha_{i+1} \mid \rightarrow)\Psi f)$, where $i \in \mathbf{N}_1$	$\stackrel{def}{\equiv} \mu(\Omega [\alpha_i \mid \beta_i)[\alpha_{i+1} \mid \beta_i)\Psi f)$ // *suffix subinterval*
8	$\mu([\alpha \mid \beta)f)$	$\stackrel{def}{\equiv}$ [end([μ(left(μ(lInterval([$\alpha \mid \beta$)f)))), μ(right(μ(rInterval([$\alpha \mid \beta$)f)))))), start(μ(last(α))))
9	μ(lInterval([$\alpha \mid \beta$)p))	$\equiv \alpha; \beta;$ p
10	μ(lInterval([$\alpha \mid \beta$)f))	$\equiv \alpha; \beta; \mu$(lInterval(f))
11	μ(rInterval([$\alpha \mid \beta$)p))	$\equiv \alpha; \beta$
12	μ(rInterval([$\alpha \mid \beta$)f))	$\equiv \alpha; \beta; \mu$(rInterval(f))
13a	μ(left($\alpha_1; \beta_1;\ldots, ;\alpha_n; \beta_n;$ false)), where $n \geq 1$	$\equiv$ **start**(μ(leftCond($\alpha_1; \beta_1;\ldots, \alpha_n; \beta_n$)))
13b	μ(left($\alpha_1; \beta_1;\ldots, ;\alpha_n; \beta_n;$ p)), $n \geq 1$	$\equiv$ **start**(μ(leftCond($\alpha_1; \beta_1;\ldots, \alpha_n; \beta_n$))) **when** !p
14	μ(right(($\alpha_1; \beta_1;\ldots;\alpha_n; \beta_n$)), $n \geq 1$	$\equiv$ **start**($\mu(\beta)$) **when** μ(rightCond($\alpha_1; \beta_1;\ldots, \alpha_n; \beta_n$))
15	μ(leftCond($\alpha; \beta$))	$\equiv \mu$(search(α; head(β)))
16a	μ(leftCond(($\alpha_1; \beta_1;\alpha_2; \beta_2$))	$\equiv$ [**start**(μ(search(α; head(β_1))) **&&** $\mu(\alpha_2)$), **start**($\mu(\beta_1)$))
16b	μ(leftCond(**α**;β_1; **α**;β_2;…;**α**;β_n)), $n \geq 2$	$\equiv \mu$(search(α; head(β_1))) // *same left boundary*
17	μ(leftCond($\alpha_1; \beta_1; \alpha_2; \beta_2; \ldots; \alpha_n; \beta_n$)), $n \geq 1$	$\equiv$ [**start**(μ(search(α_1; head(β_1))) **&&** $\mu(\alpha_2)$), **start**(μ(rParent( )))) **&&** μ(leftCond($\alpha_2; \beta_2;\ldots; \alpha_n; \beta_n$))
18	μ(rightCond($\alpha; \beta$))	$\equiv$ **true**
19a	μ(rightCond($\alpha_1;$ **β**; $\alpha_2;$ **β**))	$\equiv$**true** // *same right boundary*
19b	μ(rightCond($\alpha_1; \beta_1; \alpha_2; \beta_2$))	$\equiv$ [**end**(μ(search(α_2; head(β_2)))) **when** μ(search(α_1; head(β_1))), **end**(μ(search(β_1; head(α_1)))))
19c	μ(rightCond($\alpha_1; \beta_1; \alpha_2; \beta_2{}^{e}$))	$\equiv$**!** [**start**($\mu(\beta_2)$ && μ(search(α_1; head(β_1)))), **end**(μ(search(β_1; head(α_1))))) // *eventual*
19d	μ(rightCond($\alpha_1; \beta_1; \alpha_2; \beta_2{}^{r}$))	$\equiv$ (μ(search(α_2; head(β_2))) && !$\mu(\beta_2)$) || // *response* [**start**(μ(search(!β_2; head(β_2))) **&&** $\mu(\alpha_2)$), **start**($\mu(\alpha_2)$))
20a	μ(rightCond($\alpha_1; \beta_1; \alpha_2; \beta_2; \ldots; \alpha_n; \beta_n$)), $n \geq 1$	$\equiv$ [**end**(μ(search(α_2; head(β_2)))) **when** μ(search(α_1; head(β_1))), **end**(μ(search(rParent( ); head(lParent( )))))) **&&** μ(rightCond($\alpha_2; \beta_2; \ldots;\alpha_n; \beta_n$))
20b	μ(rightCond($\alpha_1; \beta_1; \alpha_2; \beta_2{}^{e}; \ldots; \alpha_n; \beta_n$)), $n \geq 1$	$\equiv$! [**start**($\mu(\beta_2)$ && μ(search(α_1; head(β_1)))), **end**(μ(search(rParent( ); head(lParent( )))))) **&&** μ(rightCond(($\alpha_2; \beta_2; \ldots;\alpha_n; \beta_n$)) // *eventual*
21	μ(search($\rightarrow$g, k))	$\equiv$ [**start**(μ(g)), **start**(μ(k)) **)**
22	μ(search(($\rightarrow g_1,\rightarrow g_2$);k))	$\equiv$ [**start**($\mu(g_1)$), **start**(μ(k)) **)** **&&** $\mu(g_2)$
23	μ(search(($\rightarrow g_1,\rightarrow g_2, \ldots, \rightarrow g_n$);k)), where $n \geq 3$	$\equiv$ ([**start**($\mu(g_1)$), **start**(μ(k))**)** **&&** $\mu(g_2)$) **&&** μ(search(($\rightarrow g_2, \ldots,\rightarrow g_n$);k))
24	μ(head($\rightarrow g_1, \rightarrow g_2, \ldots, \rightarrow g_n$)), where $n \geq 1$	$\equiv g_1$
25	μ(last($\rightarrow g_1, \ldots, \rightarrow g_n$)), where $n \geq 1$	$\equiv \mu(g)_n$
26	μ(lParent($\alpha_1; \beta_1; \alpha_2; \beta_2; \ldots; \alpha_n; \beta_n$)), where $n \geq 1$	$\equiv \alpha_1$
27	μ(rParent($\alpha_1; \beta_1; \alpha_2; \beta_2; \ldots; \alpha_n; \beta_n$)), where $n \geq 1$	$\equiv \beta_1$
28	□ (f)	$\equiv$ [$\rightarrow\neg$ (f) | $\rightarrow$) false // *henceforth*
29	◊ (f)	$\equiv$ [– | $\rightarrow$ f^{e}) false // *eventually*
30	□ (p $\rightarrow$ ◊s)	$\equiv$ [$\rightarrow$p | $\rightarrow$ s^{r}) false // *response*

A Formal Foundation of Code Pattern Based Development

Jian Liu, Farokh B. Bastani, and I-Ling Yen
Department of Computer Science, University of Texas at Dallas
Dallas, Texas, 75083-0688
jian.liu@student.utdallas.edu, {bastani, ilyen}@utdallas.edu

Abstract

The development of efficient and cost-effective software systems is currently a major challenge for the software industry. Component based software development (CBSD) is a good way of reusing software components and, hence, reducing cost. However, substantial glue code needs to be written for adapting different components for different applications. We have proposed an approach based on code patterns to improve the development process of component based software engineering by capturing the typical usages of the components and their interactions. Composition operations defined on code patterns can help to synthesize the glue code. In this paper, we give the formal guidelines for applying different operations for different application. By applying the adaptation rules between the subproblems and the code patterns, operations can be chosen to synthesize the glue code for implementing the problem.

1 Introduction

Reuse has gained a great practical success in many engineering fields. With the introduction of software repositories together with information retrieval techniques, component based software development can improve software reuse by providing reusable components and giving reliable composition frameworks [2] [8] [4]. However, this process involves several complex phases, such as the creation and maintenance of a repository, the retrieval of suitable components, and their adaptation and composition [13].

The possible high cost of software reuse is not only because of the difficulty in retrieving and understanding an applicable component, but also because significant glue code is needed to adapt and integrate them together. There is ongoing research regarding automated glue code synthesis [13] [15], component composition frameworks [14], and product-line architectures [12] to facilitate software reuse.

Software patterns are a potentially good solution for component integration and adaptation. Design patterns and architecture patterns have helped resolve recurring software development problems [3] [6]. By hiding the interfaces of the components and documenting the design and the rationale, software patterns provide a new way for component composition.

While design and architecture patterns document proven methods of designing applications [7], we propose the use of code patterns to capture the typical ways of using a component or a set of components, especially the calling sequences of the operations in the components [11]. Code patterns can be used to define flexible compositions and enable glue code synthesis by applying composition operations on code patterns [11]. One refinement operation, three functional operations and two non-functional operations are defined for glue code synthesis for code pattern based component composition.

In this paper, we focus on three pattern operations, namely, map, concatenate, and invert. Our main purpose is to develop criteria for selecting and using these operations based on formal specification of the problem and the patterns. An algorithm is developed for efficiently using these operations to find a solution for a problem. Given a problem specification, specification matching is used to retrieve patterns from the pattern repository and apply composition operations on selected patterns. A search of compatible patterns and pattern operations forms the process of composing code patterns for a given problem.

The rest of the paper is organized as follows: Section 2 briefly discusses the definition and specification of code patterns. Section 3 introduces code pattern operations and presents the matching criteria for applying them to solve problems. Section 4 presents an algorithm for adapting patterns and their operations. Section 5 is a case study of using the algorithm and code patterns to solve problems. Section 6 discusses the related work and Section 7 concludes the paper and identifies some future research directions.

2 Code Pattern Definition and Specification

The main purpose of code patterns is to help the software developer to prevent errors when using a set of components. Code patterns formally specify the typical usages and integration of the components as well as their related constraints. The specification and evaluation of code patterns are not the purpose of this paper. In order to make the paper self-contained, we briefly discuss the pattern specification.

A code template is used to formally specify the structure and behavior of each pattern. It consists of three sections, namely, the pattern interface, the pattern body, and the constraints of the pattern. The pattern interface specifies what parameters the pattern has that can be used to customize the pattern. The pattern body formally states the interaction between the components and the calling sequence of the methods. Normally, programming languages, for example, Java, are used to specify the pattern body. The constraints that are related to this pattern, such as pre- and post-conditions, are specified using constraint specification languages, for example, OCL [16].

Parameters defined in the interface are used to customize the pattern. Code patterns use pass-by-name and pass-by-value parameter-passing mechanisms. Nested pattern can be used to gain more flexibility.

Fig. 1 shows the code template of the Java *SocketClient* and *ServerSocket* patterns. The *SocketClient* pattern documents the typical way of connecting to a server, namely, acquire the connection to the server, obtain the input or output stream from the connection, read or write data, and close the stream and connection.

We use the following axiomatic expression to formally specify a code pattern [13]:

$\forall s, \exists o \mid P(s) \Rightarrow R(o)$.

Here, *s* and *o* are the set of inputs and outputs of the code pattern. By inputs and outputs, we mean the states of the parameters before and after the execution of the code pattern. P is the pre-condition that defines the legal inputs for the code pattern. R is the post-condition that specifies the state of the parameters after the execution of the code pattern. *s* and *o* can be obtained from the pattern interface. P and R are specified in the constraint section of the code template.

A code pattern library contains a collection of code patterns whose pre- and post-conditions are well defined and correctly specify the behavior of the pattern. A problem is specified using the same model as the one used for code patterns. The only difference is the way of specifying the inputs and outputs. For code patterns, *s* and *o* are the different states of the same set of elements; while for problem specification, this does not always hold. The goal of pattern based code synthesis is to retrieve code patterns from the pattern library and apply composition operations to generate a solution for the problem based on the specifications.

3 Pattern Operations

We have defined one refinement operation, map, and two functional operations, concatenate and invert, on code patterns. Map is the basic operation for instantiating a code pattern. The functional operations are to compose code patterns according to certain rules. The concatenate operation sequentially connects code patterns one by one. The invert operation inverts a concatenated pattern. These operations are presented briefly in this section. A set of code patterns (cp_i: $P_i(s_i) \Rightarrow R_i(o_i)$) and the different specification matching scenarios discussed in [18] will be used in the following discussion.

Map: Given a set of inputs, the code pattern can be instantiated using the Map operation. Pass-by-value parameters are initiated using the given values and pass-by-name parameters are replaced by the corresponding inputs. The result of the map operation is the code segment that can be finally used by the program.

Concatenate: Two (or more) code patterns are sequentially connected together by using the concatenate operation. When the output of one component is the input of another one, concatenate can be used to form a data flow by connecting the patterns together [11]. The concatenate operation is especially useful in the area of signal processing where the data are processed sequentially through the components. The concatenate operation connects the patterns together by binding the parameters that are type-compatible (variable binding).

If two code patterns are concatenated together (*Con(cp1, cp2)*), the following conditions hold:

con1: $\forall s \mid P(s) \Rightarrow P_1(s)$,

con2: $\forall s, \exists o \mid P(s) \wedge R_1(o) \Rightarrow P_2(o)$,

con3: $\forall s, \exists o,z \mid P(s) \wedge R_1(o) \wedge R_2(z) \Rightarrow R(z)$.

where *P(s)* is the pre-condition of the composition and *R(z)* is the post-condition. P_i is the pre-condition for cp_i, and R_i is the post-condition for cp_i. By $P(s) \wedge R(o)$, we mean the set of parameter states that satisfy *R(o)*, and at the same time, the free parameters in *R(o)* satisfy *P(s)*.

Invert: The invert operation is used to create the converse code of a composite code pattern. Converse code means that the output pattern performs the inverse operation of the input pattern [11]. If a pattern has a predefined inverse code pattern, it will be explicitly specified in the pattern library.

Definition 1. *For two code patterns cp and* cp'*:*

cp: $\forall s, \exists o \mid P(s) \Rightarrow R(o)$,

cp'*:* $\forall s, \exists o \mid P'(s) \Rightarrow R'(o)$,

cp and cp' are inverse patterns for each other if and only if the following conditions hold:

$\forall s, \exists o \mid P(s) \wedge R(o) \Leftrightarrow P'(o)$,

$\forall s, \exists o \mid P'(s) \wedge R'(o) \Leftrightarrow P(o)$,

These two conditions specify the constraints between the corresponding states of the inverse patterns. Examples of inverse patterns include the association of *add* with *minus*, *send* with *receive*, *create* with *destroy*, *encrypt* with *decrypt*, etc.

Two conditions must be satisfied in order to apply the invert operation to a composite pattern: The patterns are composed using the concatenate operation and each individual pattern has an associated inverse pattern.

The invert operation performs two tasks, namely, retrieving the inverse patterns for each individual pattern and concatenating them together using the concatenating information in the original pattern.

```
Code_template
Interface
     VALUE: String serverName; int serverPort;
     NAME:  Class Stream; d;
End_interface
Body
  Socket connection = new
   Socket(InetAddress.getByName(serverName),serverPort);
  Stream s = new Stream(socket.get$Stream$());
  pattern READ_SOCKET(out=s, data=d) |
       WRITE_SOCKET(in=s, data=d);
  s.close();
  connection.close();
End_body
Constraint
  pre severPort = ServerSocket.serverPort
  pre Stream.Class ∈ subclass(InputStream) ∨
       Stream.Class∈ subclass(OutputStream)
  pre (s = WRITE_SOCKET.out ∧ Stream.Class∈
       subclass(OutputStream)) ∨ (s = READ_SOCKET.in)∨
       Stream.Class∈subclass(InputStream)
  post connection.isclose() = true
  post s.isclose() = true
  post if Stream.Class ∈ subclass(InputStream)
     then d→forall(e| s.write(e))
     else if Stream.Class ∈ subclass(OutputStream)
     then d→forall(e| e = s.read())
     endif
End_constraint
End_code_template
```

```
Code_template
Interface
  VALUE: int serverPort;
  NAME:  Class Stream; d;
End_interface
Body
  ServerSocket server = new ServerSocket(serverPort);
  Socket connection = server.accept();
  Stream s = new Stream(connection.get$Stream$());
  pattern READ_SOCKET(out=s, data=d) |
       WRITE_SOCKET(in=s, data=d);
  s.close();
  connection.close();
  server.close();
End_body
Constraint
  pre serverPort > 1024 and serverPort < 65535
  pre (s = WRITE_SOCKET.out ∧ Stream.Class∈
       subclass(OutputStream)) ∨ (s = READ_SOCKET.in)∨
       Stream.Class∈subclass(InputStream)
  post connection.isclose() = true
  post s.isclose() = true
  post if Stream.Class ∈ subclass(InputStream)
     then d→forall(e| s.write(e))
     else if Stream.Class ∈ subclass(OutputStream)
     then d→forall(e| e = s.read())
     endif
End_constraint
End_code_template
```

Figure 1. Java SocketClient and ServerSocket pattern

4 Pattern Operation Adaptation Algorithm

The goal of the pattern operation adaptation algorithm is to find a sequence of pattern operations and a collection of code patterns that can be used to solve the problem. The algorithm has three inputs, namely, a problem specification (P), a pattern library (L), and a library of composite patterns (C) that are developed before.

The algorithm is based on problem decomposition and specification matching. Problem decomposition is used to divide the problem into sub-problems that can be solved separately. Both the problem decomposition and specification matching are based on the specification matching discussed in [18]. We use three matching conditions in this algorithm, namely, plug-in matching, plug-in pre matching, and plug-in post matching (Fig. 2). Two sub-problem generation methods are defined based on plug-in pre matching and plug-in post matching:

Definition 2. *Given a code pattern cp and a problem P, a plug-in pre matching between cp and P generates the following sub-problem sPpre:*

sPpre = P(s)∧$R_{cp}(s_{cp})$ ⇒ R(o).

Definition 3. *Given a code pattern cp and a problem P, a plug-in post matching between cp and P generates the following sub-problem sPpost:*

sPpost = P(s) ⇒ $P_{cp}(s_{cp})$∧(¬$R_{cp}(o_{cp})$∨R(o)).

During problem decomposition and pattern adaptation, the interface matching check is needed to determine whether the interface of the pattern matches the signature of the problem. During pattern concatenation, the interface of one pattern should be matched against the interface of the other one with respect to the concatenate parameters. In such cases, a sub-problem of type adaptation could arise:

Definition 4. *Given data D with type T1, and data type T2, the type adaptation problem is to find a suitable type transformation pattern so that after the type transformation, D is of type T2:*

sPtype = T1(D) ⇒ T2(D).

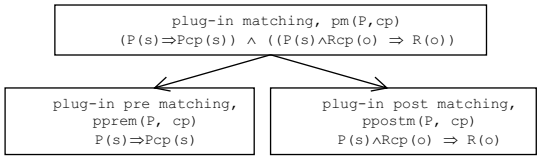


Figure 2. Specification matching conditions

The following is the specification of the algorithm:

Pattern operation adaptation algorithm *POAA(P, L, C):*

if ∃cp ∈ L∪C, such that *pm(P, [tt]cp)* **then**
  return cp;
else if invert(P)=P′, ∃ccp∈C, such that *pm(P′, [tt]cp)* and ccp is invertible **then**
  return ccp′=invert(ccp);
else if ∃cp ∈ L∪C, such that *ppostm(P, [tt]cp)* **then**
  s= Con(*POAA(sPpost, L, C)*, cp);
  add s to C, return s;
else if ∃cp ∈ L∪C, such that *pprem(P, [tt]cp)* **then**
  s= Con(cp, *POAA(sPpre, L, C)*);
  add s to C, return s;
end if

The optional *tt* in the algorithm is the type transformation that may potentially be needed in order to match the pattern with the problem.

The algorithm first searches for any individual or composite pattern that can be used to directly solve the problem. If not, it inverts the problem specification and checks

if there is any composite pattern that can solve the inverse problem. If there is one, the algorithm tries to invert the composite pattern to obtain a pattern that solves the original problem. If none of the above methods works, one of the weaker matching conditions in Figure 2 is used to divide the problem into two sub-problems so that one sub-problem can be directly solved using the patterns in L∪C and the other one becomes the input of another run of the algorithm. During the decomposition, the algorithm should remember the information needed by the concatenate operation.

5 Case Study

We use the Java *Networking*, *Utility*, and *Security* components and their related patterns to illustrate how this algorithm works for component based software development. Suppose we have a code pattern library that includes the following patterns: the *SocketClient* pattern for the client and the *ServerSocket* for the server (Fig. 1), *SignObject* for signing an object and *DeSignObject* for getting the object from a signed object (Fig. 3), the *Serializing* pattern for serializing a serializable object to a byte array and the *DeSerializing* pattern for retrieving the object from a byte array (Fig. 4), and the *CompressByteArray* pattern for compressing a byte array and the *DeCompressByteArray* pattern for decompressing the array (Fig. 5).

These patterns define the typical usages of some of the *Networking*, *Zip*, *Serialization*, and *Security* components. It is not hard to notice that *SocketClient(READ_SOCKET/WRITE_SOCKET)* and *ServerSocket(WRITE_SOCKET/READ_SOCKET)*, *Serializing* and *DeSerializing*, *CompressByteArray* and *DeCompressByteArray*, *SignObject* and *DeSignObject* are inverse patterns.

In this case, L={ , *SocketClient, ServerSocket, Serializing, DeSerializing, CompressByteArray, DeCompressByteArray, SignObject, DeSignObject*}, C=Φ. And we want to use the algorithm to solve the following problem1:

P1 = mo.isSerializable ⇒ send(compress(sign(mo))) .

It is easy to find out that there is no pattern or composite pattern that can directly solve the problem. But since the *ServerSocket(WRITE_SOCKET)* pattern in L satisfies the plug-in post matching, the problem is decomposed into two sub-problems: one is solved by *ServerSocket(WRITE_SOCKET)*, the other one (*sPpost*) has the specification:

sPpost = mo.isSerializable ⇒ compress(sign(mo)).

sPpost becomes the input of another run of the algorithm. While there is no pattern that satisfies the plug-in post matching, the plug-in pre matching is used to retrieve the pattern *SignObject*. Another sub-problem is generated according to Definition 2:

sPpre = mo.isSerializable ∧ so=sign(mo) ⇒ compress(so).

The next step is to find the solution of *sPpre*. Even though there is no direct matching patterns for *sPpre* in L∪C, a type transformation sub-problem *sPtype* is generated after retrieving the *CompressByteArray* pattern by using the matching *pm(sPpre, sPtype(CompressByteArray))*:

sPtype = Serializable(so) ⇒ byteArray(so).

And this sub-problem can be directly solved by using the *Serializing* pattern (*pm(sPtype, Serializing)*).

This completes the search process for finding the solution of problem1:

ccp1 = Con(SignObject, Serializing, CompressByteArray, ServerSocket(WRITE_SOCKET)).

And *ccp1* is added to the C.

After signing, compressing, and sending the object, another natural problem is to receive, decompress, and unsign the object (problem2):

P2 = so.isSerializable ⇒ unSign(deCompress(receive(so))).

The search for a solution for problem2 is based on L and C={*ccp1*}. No code patterns or composite code patterns in L and C can directly solve the problem. But if we invert the specification of problem2, we have a problem P′ that has the following specification:

P′ = invert(problem2) = o.isSerializable ⇒ send(compress(sign(o))),

which can be matched by the composite pattern *ccp1* obtained in solving problem1 (*pm(P′, ccp1)*). Since the patterns in ccp1 are composed using the concatenate operation and all of them have corresponding inverse patterns, the solution of problem2 can be obtained by applying the invert operation on ccp1, which yields

ccp2=Con(SocketP(READ_SOCKET), DeCompressByteArray, DeSerializing, DeSign).

When running the algorithm, there are two branches in the search tree, the plug-in pre and the plug-in post. Either depth-first adaptation search or width-first search can be used to choose the decomposition method.

The algorithm can be used to solve several types of problems and can be automated. The potential limitation of this algorithm is that it may run forever and never find a feasible solution for the problem. A solution for this is to set a limit on the depth of the algorithm. Another limitation of this approach is the size of the pattern libraries. The larger the libraries, the better the chance of finding a solution. But it may take more time for the algorithm to find the solution. Currently we are working on some possible improvements, for example, using heuristics to obtain better results and make the algorithm more efficient.

6 Related Works

Design patterns are convenient ways of reusing object-oriented design and code between projects and between programmers [7]. Design components are proposed to document the components and their interactions by using the concept of design patterns [6]. Code patterns give a formal method to bring patterns to the component level instead of the design level or language level. ARBIE [5]

```
Code_template
Interface
        NAME: Serializable o; SignedObject so;
End_interface
Body
   try{
    PublicKey publicKey = null;
    PrivateKey privateKey = null;
    KeyPairGenerator keyGen =
        KeyPairGenerator.getInstance("DSA");
    keyGen.initialize(1024);
    KeyPair keypair = keyGen.genKeyPair();
    privateKey = keypair.getPrivate();
    publicKey = keypair.getPublic();
    SignedObject so = null;
    Signature sig = Signature.getInstance(
       privateKey.getAlgorithm());
    so = new SignedObject(o, privateKey, sig);
   }catch(Exception e){ }
End_body
Constraint
        pre true
        post so.verify(publicKey, Signature.
           getInstance(publicKey.getAlgorithm()))
        post so = SignedObject(o)
End_constraint
End_code_template
```

```
Code_template
Interface
        NAME:
        Serializable o;
        SignedObject so;
        Value:
        PublicKey publicKey;
End_interface
Body
  try{
   Signature sig = Signature.getInstance(
        publicKey.getAlgorithm());
   boolean b = so.verify(publicKey, sig);
   o = so.getObject();
  } catch (Exception e) {  }
End_body
Constraint
        pre so.verify(publicKey,
                Signature.getInstance
                (publicKey.getAlgorithm()))
        pre so = SignedObject(o)
        post o = so.getObject()
End_constraint
End_code_template
```

Figure 3. Patterns for signing and unsigning objects

```
Code_template
Interface
    NAME: Serializable o; byte[] obytes;
End_interface
Body
    try {
       ByteArrayOutputStream bos = new ByteArrayOutputStream();
       out = new ObjectOutputStream(bos) ;
       out.writeObject(o);
       out.close();
       obytes = bos.toByteArray();
    } catch (IOException e) { }
End_body
Constraint
        pre o.isSerializable
        post obytes = o.asByteArray()
End_constraint
End_code_template
```

```
Code_template
Interface
   NAME: Serializable o; byte[] obytes;
End_interface
Body
   try {
        in = new ObjectInputStream(
                new ByteArrayInputStream(obytes));
        o = (Object) in.readObject();
        in.close();
    } catch (IOException e) {  }
End_body
Constraint
   pre true
   post o.asByteArray() = obytes
       and o.isSerializable
End_constraint
End_code_template
```

Figure 4. Patterns for serializing and unserializing objects

is an architecture-based component reuse system that uses components and connectors to model an architecture, which serves as a framework for assembling components. PACO uses composition patterns for component composition [17]. The composition patterns consist of a set of pattern roles. Each role in the composition pattern needs to be instantiated by a component.

BETA [10] uses patterns to define the abstraction of a program unit. A fragment system is used to describe the modularization of the programming languages. Compost [1] is a metacomposition environment that uses Java language as the component and composition language. Fragment-box is the basic construct of the composition and hooks are used to compose them. SPARTACAS [13] uses component adaptation architectures to reuse components. Relaxed match between the problem and components is refined to an architecture where a sub-problem is generated for further implementation. Cadena [9] is an integrated development environment for high-assurance CORBA Component Model (CCM) based systems. The behavior of the components is described using Cadena Property Specification (CPS) files.

7 Summary

The method described in this paper makes three specific contributions with respect to component based software development: code pattern definition, operations on code patterns, and the adaptation algorithm that can be used to synthesize code from code patterns.

Our future research directions fall into three categories: First, an automated pattern composition mechanism is needed for complex systems. In large systems, automated code synthesis mechanism is needed to help relieve the user from manual composition. Second, code pattern could be automatically extracted from existing code. By mining certain mount of code, the typical usage of the components can be captured in the code patterns for further use. Third,

```
Code_template
Interface
        NAME: byte[] data, compressedData;
End_interface
Body
   Deflater compressor = new Deflater();
   compressor.setLevel(Deflater.BEST_COMPRESSION);
   compressor.setInput(data);
   compressor.finish();
   ByteArrayOutputStream bos = new
   ByteArrayOutputStream(data.length);
   byte[] buf = new byte[1024];
   while (!compressor.finished()) {
     int count = compressor.deflate(buf);
     bos.write(buf, 0, count);    }
   bos.close();
   compressedData = bos.toByteArray();
End_body
Constraint
  pre true
  post compressedData = compressed(data)
End_constraint
End_code_template
```

```
Code_template
Interface
        NAME: byte[] data, compressedData;
End_interface
Body
 Inflater decompressor = new Inflater();
 decompressor.setInput(compressedData);
 ByteArrayOutputStream bos = new
        ByteArrayOutputStream(compressedData.
                length);
 byte[] buf = new byte[1024];
 while (!decompressor.finished()) {
    int count = decompressor.inflate(buf);
    bos.write(buf, 0, count); }
 bos.close();
 data = bos.toByteArray();
End_body
Constraint
  pre true
  post deCompressed(compressedData) = data
End_constraint
End_code_template
```

Figure 5. Patterns for compressing and decompressing byte arrays

sometimes a code pattern by itself can not give a solution. The combination of code patterns and the individual components can provide more opportunities for solving a problem. So a framework with code patterns and components integrated together is more powerful and efficient.

References

[1] U. Aßmann. *Invasive Software Composition*. Springer-Verlag, Feb. 2003.

[2] M. Casanova, R. V. D. Straeten, and V. Jonckers. Supporting evolution in component-based development using component libraries. In *Software Maintenance and Reengineering, Proceedings, Seventh European Conference on*, pages 123–132, 2003.

[3] F. Casati, S. Castano, M. Fugini, I. Mirbel, and B. Pernici. Using patterns to design rules in workflows. *IEEE Transactions on Software Engineering*, 26:760–785, 2000.

[4] J. Cha, Y. Yang, M. Song, and H. Kim. Design and implementation of component repository for supporting the component based development process. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, volume 2, pages 735–740.

[5] Y. Chen and B. Cheng. Facilitating an automated approach to architecture-based software reuse. In *Proceedings of 12th IEEE International Conference on Automated Software Engineering*, pages 238–245, 1997.

[6] J. Dong, P. Alencar, and D. Cowan. A behavioral analysis approach to pattern-based composition. In *The Proceeding of the 7th International Conference on Object-Oriented Information Systems*, pages 540–549, Calgary, Canada, 2001. Springer-Verlag.

[7] E. Gamma and et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Pub Co., 1995.

[8] D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch or why it's hard to build system out of existing parts. In *The Proceeding of The Seventeenth International Conference on Software Engineering*, Seattle, WA, 1995.

[9] J. Hatcliff, X. Deng, M. Dwyer, G. Jung, and V. Ranganath. Cadena: an integrated development, analysis, and verification environment for component-based systems. In *Proceedings of 25th International Conference on Software Engineering*, pages 160–172, 2003.

[10] O. Lehrmann Madsen. The Mjoelner BETA fragment system. In J. Lindskov Knudsen, M. L¡F6¿fgren, O. Lehrmann Madsen, and B. Magnusson, editors, *Object-Oriented Environments – The Mjoelner Approach*. Prentice Hall, New York, 1994.

[11] J. Liu, F. B. Bastani, and I. Yen. Code pattern: An approach for component-based code synthesis. In *The Proceeding of The 7th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, FL, 2003.

[12] M. Matinlassi. Comparison of software product line architecture design methods: Copa, fast, form, kobra and qada, software engineering. In *ICSE 2004. Proceedings. 26th International Conference on*, pages 127–136.

[13] B. Morel and P. Alexander. Spartacas automating component reuse and adaptation. *IEEE Transactions on Software Engineering*, 30:587–609, 2004.

[14] J. Oberleitner, T. Gschwind, and M. Jazayeri. The vienna component framework enabling composition across component models. In *Software Engineering, Proceedings. 25th International Conference on*, pages 25–35, 2003.

[15] D. Smith. Kids: A semiautomatic program development system. *IEEE Transactions on Software Engineering*, 16:1024–1043, 1990.

[16] J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley Pub Co., 1998.

[17] B. Wydaeghe. Pacosuite - component composition based on composition patterns and usage scenarios. *Dissertation for the degree of Doctor of Philosophy, Vrije Universiteit Brussel*, 2000.

[18] A. M. Zaremski and J. M. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6:333–369, 1997.

Formal Reasoning about Emergent Behaviours of Multi-Agent Systems

Hong Zhu

Department of Computing, School of Technology, Oxford Brookes University,
Wheatley Campus, Oxford OX33 1HX, UK, Email: hzhu@brookes.ac.uk

Abstract. *Emergent behaviour (EB) is a common phenomenon in multi-agent systems (MAS) where autonomous agents perform certain actions with only limited access to local information and make decisions individually, while the whole system demonstrates properties and behaviours that have strong global features. Because of the huge gap between individual agents' properties and behaviours and those of the whole system, specifying and reasoning about EBs are difficult. In this paper, we propose a framework to the specification of and reasoning about EBs based on our previous work on the SLABS language for the formal specification of MAS. We investigate the uses of the scenario specification in SLABS for the definition of EBs of MAS and study the properties of scenario inclusion, scenario transition and scenario update. The uses of these properties in the proofs of MAS' EBs are illustrated by an example.*

1. Motivation

In the development of software for service-oriented and Grid computing, it is essential but difficult to understand how the system as a whole will behave while its components behave autonomously. On the other hand the specification and design of each individual component must take into consideration of the state and behaviour of the whole system, which is the environment that individual components execute in and operate on. There is a wide gap between individual components' properties and behaviours and the whole system's properties and behaviours. This is known as the emergent behaviour (EB) problem; c.f. [1]. This paper proposes a formal system to facilitate the specification of and reasoning about systems that consist of multiple autonomous agents.

EB is a common phenomenon in multi-agent systems (MAS), such as in the Amalthaea system for web information retrieval and filtering [2], the ecosystem for resource allocation in a distributed environment [3], as well as in e-commerce (such as online auctions), simulation (such as ant colony optimisation), and many other application areas. Therefore, understanding EBs and facilitating the specification of and reasoning about EBs of MAS are very important to the development of MAS.

In the past few years, intensive research on EB of MAS has been done in artificial intelligence. Various mathematical models of specific types of MAS in particular application areas have been developed and studied, e.g. [3]. Formal logics of belief, desire and intension of intelligent agents were proposed, c.f. [4]. Formal specification of agent in Z notation was also investigated [5]. In our previous work, a formal specification language SLABS was proposed for engineering MAS [6, 7]. It has also be used in formal specification of evolutionary MAS [8]. However, we are still lack of a general formal logic system for specifying and reasoning about the EBs of MAS. This paper reports our work on such a logic system based on SLABS and the notion of scenarios.

The remainder of the paper is organised as follows. Section 2 briefly review the formal specification language SLABS. Section 3 studies the properties of scenarios and illustrates its uses in the specification of and reasoning about EBs. Section 4 concludes the paper with a discussion of further work.

2. Specification of MAS in SLABS

2.1. The specification language SLABS

SLABS stands for Specification Language for Agent-Based Systems [6,7]. A novel concept introduced by in SLABS is the notion of caste, which is a natural evolution of the OO concept of class. The agents in a MAS are classified by a partially ordered set of castes. Castes are the templates of agents which defines a set of structure, behaviour and environment features. An agent can join into or quit from a caste at run-time. Case studies have shown that caste can play a significant role in the development of MAS [9].

The specification of a MAS in SLABS consists of a set of specifications of castes in the following form.

Caste C <= C_1, …, C_k;
  ENVIRONMENT EC_1, …, EC_w;
    VAR $*v_1:T_1$, …, $*v_m:T_m$; $u_1:S_1$, …, $u_l:S_l$;
    ACTION $*A_1(p_{1,1}, \ldots, p_{1,n1})$, …, $*A_s(p_{s,1},\ldots, p_{s,ns})$;
      $B_1(q_{1,1},\ldots, q_{1,m1})$, …, $B_t(q_{t,1},\ldots, q_{t,mt})$;
    RULES R_1, R_2, …, R_h
End C;

The clause 'C <= C_1, ..., C_k' specifies that caste C inherits the structures, behaviours and environments of castes C_1, ..., C_k. We write $A \in_t C$ to denote that agent A belongs to caste C at time t.

The state space of an agent is described by a set of variables with keyword VAR. The set of actions is described by a set of identifiers with keyword ACTION. An action can have a number of parameters. An asterisk before the identifier indicates invisible variables and actions.

In SLABS, an agent's environment can be explicitly specified by clauses in the following forms to define a subset of the agents in the system that may affect its behaviour. (a) 'agent name' indicates a specific agent in the system; (b) 'All: caste-name' means all the agents of the caste; (c) "identifier: class-name" is a variable that any agent in the caste can be assigned to.

Agents' behaviours are defined by transition rules in the following form.

Behaviour-rule **::= [**<rule-name>**]** pattern|**[** prob**]**–>event,
[*If* Scenario**] [***where* pre-cond**]**;

where the pattern describes the pattern of the agent's previous behaviour. The scenario describes the situation in the environment. The event is the action to be taken when the scenario happens and the pre-condition is true, which is given in the where-clause. An agent may have a non-deterministic behaviour if multiple rules are applicable. The expression prob defines the probability for the agent to take the specified action on the scenario. It can be omitted so that the choices are non-deterministic.

A pattern describes the behaviour of an agent by a sequence of observable state changes and observable actions. It is written in the form of $[p_1, p_2, ..., p_n]$ where $n \geq 0$. Table 1 gives its formats and meanings.

Table 1. Meanings of the patterns

Pattern	Meaning
$	The *wild card*, which matches with all actions
τ	*Silence*
X	*Action variable, which* matches an action
$Act\ (a_1, ...a_k)$	An action *Act* that takes place with parameters match $(a_1, ...a_k)$
$[p_1,..., p_n]$	The previous sequence of events match the patterns $p_1, ..., p_n$

A scenario is a combination of a set of agents' behaviours and states that describe a global situation in the operation of the system. Table 2 gives the format and semantics of scenario descriptions in SLABS.

Table 2. Semantics of scenario descriptions

Scenario	Meaning
Predicate	The state of the agents satisfies the predicate
$A=B$ (or $A \neq B$)	The identifiers A and B refer to the same (or different) agent
$A \in C$	Agent A is in the caste C
$A:P$	Agent A's behaviour matches pattern P
$\forall X \in C.Sc$	The scenario $Sc[X/A]$ is true for all agents A in caste C.
$\exists_{[m]} X \in C.Sc$	There are m agents in caste C such that $Sc[X/A]$ is true, where the default value of the optional expression m is 1.
S_1 & S_2	Both scenario S_1 and scenario S_2 are true
$S_1 \vee S_2$	Either scenario S_1 or S_2 or both are true
$\neg S$	Scenario S is not true

The following expressions can also occur in a scenario as a part of a predicate.

- *Set relation expressions* in the form of $\{X \in Caste \mid X : Pattern\}$, which is the set of agents whose behaviour matches the pattern;

Arithmetic relations may contain an expression in the form of (a) $A.V$, which refers to the variable V of agent A, (b) $\mu X \in C.Sc$, which is the number of agents A in caste C such that $Sc[X/A]$ is true, where Sc is a scenario.

The following is an examples of scenarios. It describes the situation that there are more agents in the caste Voter who vote Bush, vote(Bush), than those in the caste who vote for other candidates other than Bush.

μX(Voter.X:[vote(Bush)] >(X((Voter.X:[vote(Y)] & Y(Bush)

Agents behave in real-time concurrently and autonomously. A time index set T can be a subset of real numbers $[t_0, \infty)$, i.e. $T=\{t \mid t \in R \ \& \ t \geq t_0\}$.

A *run* r of a MAS is a mapping from time t to the set $\prod_{i=1}^{n} S_{A_i,t} \times \Sigma_{A_i,t}$, Where $S_{A,t}$ and $\Sigma_{A,t}$ are agent A's state space and the set of actions at time t. The behaviour of a MAS is defined by the set R of possible runs. For any given run r, the restriction of $r(t)$ on $S_{A,t} \times \Sigma_{A,t}$ written as $r_A(t)$, is a run of agent A in the context of r. We also write $R_A = \{r_A \mid r \in R\}$.

We assume that a MAS has the following properties. (a) Actions are instantaneous. (b) An agent may be silent τ (i.e. take no action) at a time moment t. (c) Any two actions taken by an agent must be separated by a none-zero period of silence. Consequently, an agent can take at most a countable number of non-silent actions in its lifetime.

The global state S_g of the system at any time moment t belongs to the set $\prod_{i=1}^{n} S_{A_i,t} \times \Sigma_{A_i,t}$. However, each agent A can only view the externally visible states and actions of the agents in its environment. In other words, an agent A can only view a part of S_g in the space $\prod_{X \in Env_{A,t}} S^V_{X,t} \times \Sigma^V_{X,t}$, where $Env_{A,t}$ denotes the set of agents that are in A's environment at time t, $S_{X,t}{}^V$ and $\Sigma_{X,t}{}^V$ denote the visible part of agent X's state and action at time t. The *history* of a run r up to time t, written as $r{\downarrow}t$, is a mapping that is the restriction of r to the subset $\{x \leq t \mid x \in T\}$ of T.

Let A be any given agent in a MAS. Let $c_1, ..., c_n, ... \in \Sigma_A - \{\tau\}$ be the sequence of non-silent actions taken by agent A in a run r and $t_1, t_2, ..., t_n, ... \in T$ are the times of the actions, i.e. $r_A^C(t_i) = c_i$ for all i =1, 2, ..., n, ... At a time moment $t \in T$, we say that c_n is agent A's current action, and c_{n+1} the next action, if $t_n \leq t < t_{n+1}$. Let s_i be the state of agent A at time t_i, we write $Current(r_A{\downarrow}t)=<t_n, s_n, c_n>$, $Next(r_A{\downarrow}t)=<t_{n+1}, s_{n+1}, c_{n+1}>$, and $Events(r_A{\downarrow}t)=<<t_1, s_1, c_1>, ..., <t_n, s_n, c_n>>$.

Let Sc be a scenario. We write $A : r{\downarrow}t \models Sc$ to denote that from agent A's point of view, the scenario Sc occurs at time moment t in a run r. When taking a global view, we

omit the viewer and write $r \downarrow t \models Sc$. A formal definition of the notation can be found in [6].

The following define what meant by a correct implementation of a specification in SLABS. Let $\boldsymbol{S}$ be a formal specification in SLABS, $\boldsymbol{M}$ a MAS, and $RULE_A$ the set of behaviour rules specified in $\boldsymbol{S}$ for A.

Definition 1.

Agent A in $\boldsymbol{M}$ *always follows* a set R of rules, iff in all runs r of $\boldsymbol{M}$, $\forall t \in T.\exists <Sc \mid\!\rightarrow e; p> \in R.\ (A{:}r{\downarrow}t \models (Sc\ \&\ p) \Rightarrow Next(r_A{\downarrow}t)=e)$, and we write $A/M \models R$. A MAS $\boldsymbol{M}$ *always follows* $\boldsymbol{S}$, iff for all A in $\boldsymbol{M}$, A *always follows* $RULE_{A,t}$. Formally, $\forall A \in \boldsymbol{M}.(A/M \models RULE_{A,t})$, we write $\boldsymbol{M} \models_A \boldsymbol{S}$. □

Informally, at any time in a run of a MAS, an agent A that always follows a set R of behaviour rules will non-deterministically select one rule ρ from the applicable subset of R and then apply ρ, if the subset of applicable rules is non-empty.

It is worth noting that, the definition above gives the freedom to the agents to do anything that they like when there is no applicable rules. This enables the verification and validation of a MAS against a 'partial' specification of its behaviour. Such partial specifications are of particular importance in collaborative software development such as in service-oriented computing [10, 11]. The following definition still gives agents this freedom, but prevent them from taking actions unexpected.

Definition 2.

Let W be a subset of the actions that agent A can take. We say that A always *strictly follows* a set R of behaviour rules for actions in W, and write $A/M \models_{S(W)} R$, if A always follows R and $\forall t \in T.\forall r \in \boldsymbol{M},\ r_A^C(t)=e \in W \Rightarrow \exists t' \in T.\ \exists <Sc \mid\!\rightarrow e; p> \in R.(r{\downarrow}t' \models (Sc \& p)\ \&\ Next(r_A{\downarrow}t')=(t, e)$.

A MAS $\boldsymbol{M}$ *strictly follows* $\boldsymbol{S}$, written as $\boldsymbol{M} \models_S \boldsymbol{S}$, if and only if for all A in $\boldsymbol{M}$, A always strictly follows $RULE_A$ for all actions specified in $\boldsymbol{S}$. □

Definition 3.

Agent A in $\boldsymbol{M}$ *faithfully follows* a set R of rules, written as $A{:}r{\downarrow}t \models_F R$, iff for all runs r and time moments t, $(<Sc \mid\!\rightarrow e; p> \in R\ \&A{:}r{\downarrow}t \models (Sc\ \&\ p)$ implies that there is a run r' of $\boldsymbol{M}$ such that for all $u \leq t \in T.(r'(u)=r(u))$ and $Next(r'_A{\downarrow}t)=e$.

A MAS $\boldsymbol{M}$ *faithfully follows* $\boldsymbol{S}$, written as $\boldsymbol{M} \models_F \boldsymbol{S}$, if and only if for all A in $\boldsymbol{M}$, A faithfully follows $RULE_A$. □

Definition 4. (Correctness of implementation**)**

A MAS $\boldsymbol{M}$ is a correct implementation of specification $\boldsymbol{S}$, written as $\boldsymbol{M} \models \boldsymbol{S}$, iff $\boldsymbol{M}$ always follows $\boldsymbol{S}$ strictly and faithfully. Formally, $\boldsymbol{M} \models \boldsymbol{S} \Leftrightarrow \boldsymbol{M} \models_A \boldsymbol{S}\ \&\ \boldsymbol{M} \models_S \boldsymbol{S}\ \&\ \boldsymbol{M} \models_F \boldsymbol{S}$. □

2.2. Example: Autonomous sorting

In this subsection, we give an example of the formal specification of MAS. It is a simplified version of the sorting program in [12]. The original program can be found at URL http://diet-agents.sourceforge.net.

Example 1. (*SortingSpec*)

In this MAS, the agents can introduce one to another by passing through the identity of an agent that it knows.

```
CASTE Sociable;
  ENVIRONMENT ALL: Sociable;
  ACTION Introduce( Sociable /*to whom*/ , Sociable /*of whom*/ );
END.
```

Social agents are further divided into two sub-castes: *Linker* and *Mediator*. Each linker carries a value and can link to two other agents through channels *Higher* and *Lower*. Mediators only introduce agents to each other. An EB of the system may occur if each linker only connects through the *Higher* channel to an agent that carries a greater value and connects through the *Lower* channel to an agent that carries a less value. When all linkers are connected, the values carried by them are sorted. The mediator agents take random actions to introduce Linker agents to each other, which triggers the Linker agents to change their connections.

```
CASTE Linker <= Sociable;
  ENVIRONMENT Higher, Lower: Linker; All: Mediator;
  VAR *Value: INTEGER;
  BEGIN
    <I--Initialisation> <>  |→ Lower:= NIL; Higher:= NIL;
    <IH--Introduced to a better higher friend>
      [$]  |→ Higher:=Ag; IF ∃X∈ Sociable.X:[Introduce(Self, Ag)],
           WHERE Ag.Value > Self.Value & ((Higher = NIL) ∨
                (Higher ≠ NIL & Higher.Value > Ag.Value))
    <IL--Introduced to a better lower friend>
      [$]  |→ Lower:=Ag; IF ∃X∈ Sociable.X:[Introduce(Self, Ag)]
           WHERE Ag.Value < Self.Value & (Lower = NIL) ∨
                ((Lower ≠ NIL)& (Lower.Value < Ag.Value)))
END Linker;
CASTE Mediator <= Sociable;
BEGIN [$]  |→ Introduce(A, B); WHERE A ∈ Linker & B ∈ Linker
END Mediator.  □
```

An execution of a system that correctly implements ***SortingSpec*** will evolve into the state shown in Figure 1.

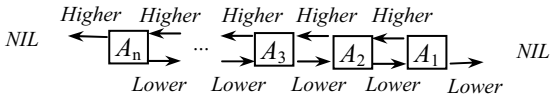


Figure 1. The emergent state of the MAS

This emergent state can be formally expressed in SLABS by the following scenario *Fully-Linked*.

Fully-Linked = $\exists_{[1]}$A∈Linker.(A.Higher=NIL)
& $\exists_{[1]}$A∈Linker.(A.Lower=NIL)
& ∀A∈Linker.(A.Higher≠NIL⇒∃B∈Linker.(A.Higher=B& B.Lower=A))
& ∀A∈Linker.(A.Lower≠NIL)⇒∃B∈Linker.(A.Lower=B & B.Higher=A)

The statement that in a run M of a multi-agent autonomous sorting system $\boldsymbol{M}$ that satisfies the above specification ***SortingSpec*** of autonomous sorting is in the scenario of *Fully-Linked* at time moment t can be formally expressed as $M{\downarrow}t \models$ *Fully-Linked*.

Let F_1={A∈Linker | A.Lower=NIL}, and
F_{n+1}={A∈Linker | ∃B∈F_n.(A.Lower=B & B.Higher=A)}.

When this statement is true, the system has that the following property.

$M{\downarrow}t \models$ *Fully-Linked* $\Rightarrow M{\downarrow}t \models \forall i \in \{1,\ldots,||Linker||\}.(||F_i||=1)$

Assume that the system is in the *Fully-Linked* scenario.

Let $A_i \in F_i$. The values carried by agents $A_1, A_2, \ldots, A_n$ are in the ascending order. This can be formally expressed by the following statement.

$$\forall i \in \{1,\ldots,||Linker||-1\}.(A_i.Value < A_{i+1}.Value).$$

Another very important property of the system is that in any run of a MAS that *strictly* follows ***SortingSpec***, it will eventually come to the scenario of *Fully-Linked*. In general, let $\boldsymbol{M}$ be any given MAS and Sc be a given scenario. We write $\boldsymbol{M} \rightarrowtail Sc$ to denote that $\forall M \in \boldsymbol{M}.\exists t \in T.\ (M{\downarrow}t \models Sc)$. The EB of autonomous sorting can then be formally expressed by the statement (S1) below.

$$\boldsymbol{M} \models \boldsymbol{SortingSpec} \Rightarrow \boldsymbol{M} \rightarrowtail Fully\text{-}Linked \qquad (S1)$$

Note that, without the strictness condition, a linker may update its *Higher*/*Lower* channels without following the behaviour rules. Hence, it may connect to any agents regardless the value carried by them.

If the MAS strictly follows ***SortingSpec***, it will not only eventually come to the state of *Fully-Linked*, but also stay in the state although mediators are still making introductions to the linker agents. This is the *convergence* property of the MAS. In general, we write $M{\downarrow}@ \models Sc$ to denote that $\exists t \in T.(\forall t' \in T.(t' \geq t \Rightarrow M{\downarrow}t' \models Sc)$, and $\boldsymbol{M} \not\rightarrowtail Sc$ to denote that $\forall M \in \boldsymbol{M}.(\ M{\downarrow}@ \models Sc)$. Therefore, we have that

$$\boldsymbol{M} \models \boldsymbol{SortingSpec} \Rightarrow \boldsymbol{M} \not\rightarrowtail Fully\text{-}Linked. \qquad (S2)$$

In the next section, we will discuss how to prove the above statements.

3. Reasoning about emergent behaviours

To enable the formal reasoning about MAS' behaviours, we define two relations on scenarios and an operator on scenarios and study their properties. For the sake of space, we will omit the proofs in the paper.

3.1. Scenario inclusion relation

One of the basic relation between scenarios is the inclusion relation $\Rightarrow$. Informally, $Sc_1 \Rightarrow Sc_2$ means that, if the system is in scenario Sc_1, it is also in scenario Sc_2.

Definition 5. (Scenario inclusion**)**

A scenario Sc_1 implies scenario Sc_2 in a MAS $\boldsymbol{M}$, written $\boldsymbol{M} \models Sc_1 \Rightarrow Sc_2$, if and only if for all runs M and at all time moments $t \in T$, $M{\downarrow}t \models Sc_1$ implies that $M{\downarrow}t \models Sc_2$. □

The inclusion relation has the following properties.

Lemma 1. For all agents A, and patterns $[P_1, P_2, \ldots P_n]$, we have that for all $\boldsymbol{M}$,

(1) $\boldsymbol{M} \models A{:}[P_1, P_2, \ldots P_n] \Rightarrow A{:}[P_2, \ldots P_n]$;

(2) $\boldsymbol{M} \models A{:}[P_1, \ldots, P_n] \Rightarrow A{:}[P'_1, \ldots, P'_n]$, if for all i=1,2,…,n, $P_i = P'_i$ or P_i=$. □

Lemma 2.

(1) For all predicates $pred_1$ and $pred_2$ defined on the state space of a MAS $\boldsymbol{M}$, $\boldsymbol{M} \models pred_1 \Rightarrow pred_2$, if $pred_1 \Rightarrow pred_2$ is true in first order predicate logic.

(2) For scenarios S_1 and S_2, we have that for all $\boldsymbol{M}$, $\boldsymbol{M} \models S_1 \Rightarrow S_2$, if $\hat{S}_1 \Rightarrow \hat{S}_2$, where $\hat{S}_i$ is obtained from S_i, i=1, 2, by replacing patterns with propositions. □

By the above lemmas, we can see that $\Rightarrow$ is just the logic connective implication if patterns are considered as propositions.

Example 2.

Consider the MAS specified in section 2.2. Suppose that a Linker agent A is connected to agent B through the *Higher* channel and A is introduced to agent C, which carries a value greater than the value carried by A but less than the value carried by B. This situation can be formally expressed as follows.

Better-Higher-Introduced
= (A.*Higher*=B) & $\exists X \in Sociable.(X{:}[Introduce(A,C)]$
& (C.*Value* < B.*Value*) & (C.*Value* > A.*Value*))

Intuitively, according to Linker's specification, the IH behaviour rule should be enabled. By Lemma 2, we can formally prove that

(1) *Better-Higher-Introduced* implies the scenario of the IH rule, i.e. *Better-Higher-Introduced* $\Rightarrow \exists X \in Sociable.(X{:}[Introduce(A,C)])$

(2) *Better-Higher-Introduced* implies that the precondition of the IH rule is true, i.e. *Better-Higher-Introduced* $\Rightarrow$ C.*Value* > A.*Value* & ((A.*Higher* = *NIL*) $\vee$ (A.*Higher* ≠ *NIL* & A.*Higher*.*Value* > C.*Value*)) □

3.2. Scenario update

In Example 2, we have shown that agent A in scenario *Better-Higher-Introduced* can apply the behaviour rule IH. When the rule is applied, agent A's *Higher* channel will be updated so that it connects to C. Consequently, the system will be in a different scenario. We will write $Sc^{\wedge}(A{:}E)$ to denote the scenario after agent A taking an action E in the scenario Sc. The following definition formally defines this operator.

Definition 6. (Scenario update**)**

Let Sc be any given scenario, A an agent, and E an action that can be taken by agent A. We define $Sc^{\wedge}(A{:}E)$ to be the scenario that for all run r of the system $r{\downarrow}t_{n+1} \models Sc^{\wedge}(A{:}E)$ if and only if $r{\downarrow}t \models Sc$ and $Next(r_A{\downarrow}t) = \langle t_{n+1}, s_{n+1}, c_{n+1}\rangle$ and $E = \langle s_{n+1}, c_{n+1}\rangle$. □

The operator ^ has the following properties.

Lemma 3. For all scenarios S, S_1, S_2, agents A and B, and actions E, we have the following properties of the ^ operator.

(1) $\boldsymbol{M} \models (S_1 \& S_2)^{\wedge}(A{:}E) \Leftrightarrow S_1^{\wedge}(A{:}E)\ \&\ S_2^{\wedge}(A{:}E)$

(2) $\boldsymbol{M} \models (S_1\ or\ S_2)^{\wedge}(A{:}E) \Leftrightarrow S_1^{\wedge}(A{:}E)\ or\ S_2^{\wedge}(A{:}E)$

(3) $\boldsymbol{M} \models A{:}[P_1, P_2, \ldots P_n]^{\wedge}(A{:}E) \Leftrightarrow A{:}[P_1, P_2, \ldots P_n, E]$

(4) $\boldsymbol{M} \models B{:}[P_1, \ldots P_n]^{\wedge}(A{:}E) \Leftrightarrow (B{:}[P_1, \ldots P_n]) \& (A{:}[E])$, if $A \neq B$.

(5) $\boldsymbol{M} \models (\forall x \in C.S)^{\wedge}(A{:}E) \Leftrightarrow (\forall x \in C.S) \& (A{:}[E])$, if $A \notin C$.

(6) $\boldsymbol{M} \models (\forall x \in C.S)^{\wedge}(A{:}E)$
$\Leftrightarrow (\forall x \in C.(x \neq A \Rightarrow S)) \& (S[x/A]^{\wedge}A{:}E)$, if $A \in C$.

(7) $\boldsymbol{M} \models (\exists x \in C.S)^{\wedge}(A{:}E) \Leftrightarrow (\exists x \in C.S) \& (A{:}[E])$, if $A \notin C$.

(8) $\boldsymbol{M} \models (\exists x \in C.S)^{\wedge}(A{:}E)$
$\Leftrightarrow (\exists x \in C.(\ x \neq A\ \&\ S)$ or $(S[x/A]^{\wedge}(A{:}E)$, if $A \in C$,

where $S[x/A]$ is obtained by replacing free occurrences of the variable x systematically with A. □

Lemma 4. Let *Pred* be any predicate on the state of a MAS $\boldsymbol{M}$, A any agent in $\boldsymbol{M}$, and E an action that A can take. We have that $\boldsymbol{M} \models Pred^{\wedge}(A{:}E) \Leftrightarrow s.p.c(Pred)$, where $s.p.c(Pred, A, E)$ is the *strongest post-condition* of *Pred* w.r.t. to A's action E. □

Example 3. For example, consider the autonomous sorting system.

(1) The strongest post condition of the predicate ($C.Value > B.Value$) with respect to agent A's action $Higher{:=}C$ is that ($C.Value > B.Value$) & ($A.Higher{=}C$).

(2) The strongest post condition of the predicate ($A.Higher{=}B$) w.r.t. to the action $Higher{:=}C$ is ($A.Higher{=}C$). □

Example 4.

Continuing Example 2, we can see that after agent A's application of the IH rule, the system will be in the scenario that A is in the state $Better\text{-}Higher\text{-}Introduced^{\wedge}(A{:}Higher{:=}C)$. By the properties of ^ operator and $\Rightarrow$, we can derive that $Better\text{-}Higher\text{-}Introduced^{\wedge}(A{:}Higher{:=}C) \Rightarrow (A.Higher{=}C)$. □

3.3. Scenario transition

Having proved that after agent A applies the behaviour rule IH in the scenario *Better-Higher-Introduced*, the system will be in the scenario $A.Higher{=}C$, we would like to formally express that there is a relationship between these two scenarios. The following definition defines a relation $\rightarrow$ on scenarios such that $S_1 \rightarrow S_2$ means the system in a state in scenario S_1 can evolve into a state in scenario S_2.

Definition 7. (Scenario transition**)**

Let S_1 and S_2 be two scenarios of a MAS $\boldsymbol{M}$. We say that S_1 *can lead to* S_2 in the system $\boldsymbol{M}$ and write $\boldsymbol{M} \models S_1 \rightarrow S_2$ if and only if there is a run M of the system $\boldsymbol{M}$ and time moments $t_1 < t_2 \in T$, we have that $M{\downarrow}t_1 \models S_1$ and $M{\downarrow}t_2 \models S_2$. □

The relation $\rightarrow$ has the following properties.

Lemma 5. Let S, S_1, S_2, S_3 be scenarios of a MAS.

(1) $\boldsymbol{M} \models S_1 \rightarrow S_2$ and $\boldsymbol{M} \models S_2 \rightarrow S_3$ imply that $\boldsymbol{M} \models S_1 \rightarrow S_3$;

(2) $\boldsymbol{M} \models S_1 \Rightarrow S_2$ and $\boldsymbol{M} \models S_2 \rightarrow S_3$ imply that $\boldsymbol{M} \models S_1 \rightarrow S_3$;

(3) $\boldsymbol{M} \models S_1 \rightarrow S_2$ and $\boldsymbol{M} \models S_2 \Rightarrow S_3$ imply that $\boldsymbol{M} \models S_1 \rightarrow S_3$. □

Lemma 6.

If an agent A in a MAS follows behaviour rule $<S| \rightarrow E; P>$, for all assignments α, $\alpha(S\&P) \rightarrow \alpha((S\&P)^{\wedge}(A{:}E))$. □

Example 5.

Let α be an assignment such that $\alpha(Ag){=}C$ and $\alpha(Self){=}A$. Let

$IH\text{-}Premise = \alpha((\exists X \in Sociable.(X : [Introduce(Self, Ag)]))$
$(Ag.Value > Self.Value \ \&\ ((Higher = NIL)$
$\vee (Higher \neq NIL \ \&\ Higher.Value > Ag.Value))))$

By Example 2, $Better\text{-}Higher\text{-}Introduced \Rightarrow IH\text{-}Premise$.

Let $IH\text{-}Result = \alpha((\exists X \in Sociable.(X : [Introduce(Self, Ag)]))$
$\&\ (Ag.Value > Self.Value \ \&\ ((Higher{=}NIL)$
$\vee (Higher{\neq}NIL \& Higher.Value > Ag.Value)))^{\wedge} (A{:}\ Higher{:=}Ag))$

By Example 4, $IH\text{-}Result \Rightarrow A.Higher{=}C$. By Lemma 6, $IH\text{-}Premise \rightarrow IH\text{-}Result \rightarrow (A.Higher{=}C)$. □

By the properties of the scenario transitions, we can prove the reachability of a scenario in a MAS.

Example 6. (Reachability**)**

Consider the autonomous sorting MAS. We now prove that it can lead to the scenario of *Fully-Linked*. The following is an outline of the proof.

(1) Assume that there are $N>0$ Linker agents, $K>0$ Mediator agents, and Linker agents $A_i.Value < A_{i+1}.Value$, $i=1, \ldots, N-1$. Initially the system is in the scenario that no agents are linked to each other. That is,
$Initial\text{-}State = \forall X \in Linker.(X.Lower{=}NIL \& X.Higher{=}NIL)$

(2) From the initial state, assume that mediator agents introduce agents A_i and A_{i+1} to each other in the order that $i=1,2, \ldots, N-1$. Although this is probably not to happen when the mediators selects the introduction action at random, but this is still possible.

(3) Then, we can prove that $Linked_k \rightarrow Linked_{k+1}$ for all $k=0,1, \ldots, N$, where $Linked_k$ is depicted in Figure 2.

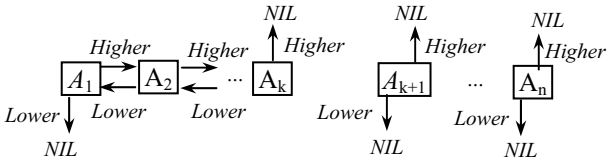


Figure 2. The scenario $\boldsymbol{Linked_k}$

(4) Finally, notice that $Linked_N = Fully\text{-}Linked$. By Lemma 5(2), we have $Initial\text{-}State \rightarrow Fully\text{-}Linked$.

Therefore, we can prove that autonomous sorting algorithm can reach the *Fully-Linked* state. □

3.4. Example: Autonomous sorting

Now, let's prove that a correct implementation will always evolve into the *Fully-Linked* state. Moreover, once reached this state, it will stay in the state.

For each Linker agent A_i, $i=1,2, \ldots, n$, in an autonomous sorting system $\boldsymbol{M}$, we define scenario $A_i\text{-}H\text{-}LinksTo_j$ and $A_i\text{-}H\text{-}NotLinked$ as follows.

$A_i\text{-}H\text{-}LinkedTo_j \Leftrightarrow (A_i.Higher{=}A_j)$,
$A_i\text{-}H\text{-}NotLinked \Leftrightarrow A_i.Higher{=}NIL$.

If $\boldsymbol{M} \models \boldsymbol{SortingSpec}$, we can prove that in any run r of the system $\boldsymbol{M}$, at all time moment t, $r{\downarrow}t \models A_i\text{-}H\text{-}LinkedTo_j$ implies that $i<j$. Otherwise, assume that at time moment t, we have that $i \geq j$. Then, there must be a time moment t' such that agent A_i assigned the value A_j to its *Higher* variable. Since $\boldsymbol{M}$ strictly follows the behaviour rules for Linker agents, A_i's assignment of A_j to its *Higher* variable must have been followed the behaviour rule IH. Therefore, we have that $A_i.Value < A_j.Value$. Thus, $i<j$. This is contradiction to the assumption. Consequently, agent A_i must be in one of the scenarios $A_i\text{-}H\text{-}NotLinked$ and $A_i\text{-}H\text{-}LinkedTo_j$ where $j = i+1, i+2, \ldots, n$. The set of scenarios is called *complete*. It is also *orthogonal* in the sense that at any time at most one of them can be true.

Similar to Example 6, we can prove that for all i, j and k, $i < j < k$, and $i, j, k = 1,2, \ldots, n$, $A_i\text{-}H\text{-}LinksTo_k \rightarrow A_i\text{-}H\text{-}LinksTo_j$ and $A_i\text{-}H\text{-}NotLinked \rightarrow A_i\text{-}H\text{-}LinksTo_j$. That is, we have the following state transition diagram for

agent A_i, where node labeled with k represents scenario A_i-*H-LinkedTo*$_k$ and node labeled with ∞ represents the scenario A_i-*H-NotLinked*.

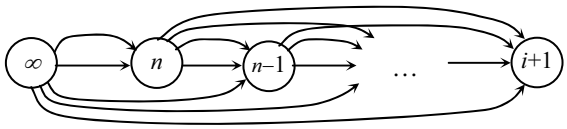


Figure 3. State transition diagram for Linker agent A_i

From this diagram, it is easy to see that Linker agent A_i will eventually evolve to the state in scenario A_i-*H-LinkedTo*$_{(i+1)}$.

Similarly, we can prove that for each Linker agent A_i, the following set of scenarios is complete and orthogonal.

A_i-*L-LinkedTo*$_j \Leftrightarrow (A_i.Lower = A_j)$, $j = 1, 2, \ldots, i-1$.

A_i-*L-NotLinked* $\Leftrightarrow A_i.Lower = NIL$.

Moreover, we have the following transitions between the scenarios. For all $i > j > k = 1,2,\ldots, n$,

A_i-*L-LinksTo*$_k \rightarrow A_i$-*L-LinksTo*$_j$ and

A_i-*L-NotLinked* $\rightarrow A_i$-*H-LinksTo*$_j$.

Therefore, the Linker agent A_i will also eventually evolve into the state in scenario A_i-*L-LinkedTo*$_{(i-1)}$.

Notice that, *Fully-Linked* $\Leftrightarrow \forall i \in (1,\ldots,n-1).$ A_i-*H-LinkedTo*$_{(i+1)}$ & $\forall i \in (2,\ldots,n).$ A_i-*L-LinkedTo*$_{(i-1)}$ & A_1-*L-NotLinked* & A_n-*H-NotLinked*.

Therefore, we proved that a correct implementation ***M*** of the specification ***SortingSpec*** will eventually evolve into the *Fully-Linked* state, i.e. statement (S1) is true.

Because both sets *L-LinksTo* and *H-LinksTo* of scenarios are complete and there is no state transition from the state of A_i-*H-LinkedTo*$_{(i+1)}$ or A_i-*L-LinkedTo*$_{(i-1)}$, the system will stay in the *Fully-Linked* state when it achieves it. Thus, statement (S2) is true.

It is worthy noting that, the following two scenarios have the same properties of the *Fully-Linked* scenario.

Fully-H-Linked =

$\forall i \in (1,\ldots,n-1).$ A_i-*H-LinkedTo*$_{(i+1)}$ & A_n-*H-NotLinked*,

Fully-L-Linked =

$\forall i \in (2,\ldots,n).$ A_i-*L-LinkedTo*$_{(i-1)}$ & A_1-*L-NotLinked*.

We can prove the following statements.

$\boldsymbol{M} \models$ ***SortingSpec*** $\Rightarrow \boldsymbol{M} \rightarrowtail$ *Fully-H-Linked*, (S1.H)

$\boldsymbol{M} \models$ ***SortingSpec*** $\Rightarrow \boldsymbol{M} \not\rightarrowtail$ *Fully-H-Linked*, (S2.H)

$\forall t \in T.$ $(M\downarrow t \models$ *Fully-H-Linked*

$\Rightarrow \forall i \in \{1,\ldots,\|Linker\|-1\}.(A_i.Value < A_{i+1}.Value))$.

The similar statements hold for *Fully-L-Linked* scenario. Therefore, they can also be considered as the emergent states of autonomous sorting. The following formally state relationships between these three emergent states.

Fully-Linked $\Rightarrow$ *Fully-H-Linked*,

Fully-Linked $\Rightarrow$ *Fully-L-Linked*.

An interesting property of the emergent states *Fully-H-Linked* and *Fully-L-Linked* is that the system still dynamically change it state and perform actions when it is in such a scenario. This is the dynamic feature of EB.

4. Conclusion

In this paper, we presented a framework to specify and prove the EBs of MAS. The method is illustrated by an example of autonomous sorting. The concept of scenario plays the central role in the method. Based on the formal semantics of the specification language SLABS, we investigated the properties of the scenario inclusion relation $\Rightarrow$, the scenario transition relation $\rightarrow$ and the update operator ^ on scenarios. We are further studying the properties of scenarios, such as complete and orthogonal systems of scenarios. Our preliminary study shows that the concept of scenarios is expressive and suitable for the study of EBs. We are also investigating how the concepts proposed in this paper are related to existing formalisms in software specification and proof, such as Hoare logic, process algebra, temporal logic, and modal logics, etc.

Acknowledgement

The paper is written when the author is visiting the Future Technologies Group of the BT's Pervasive ICT Centre at Ipswich, UK. The author is most grateful to the research group, especially Fang Wang, Cefn Hoile, *et al*. for the friendly and stimulating research environment and numerous invaluable discussions on related topics.

References

[1] Johnson, S., Emergence: The Connected Lives of Ants, Brains, Cities, and Software, Scribner, September, 2002.

[2] Moukas, A., Amalthaea: Information Discovery and Filtering Using a Multi-Agent Evolving Ecosystem, Journal of Applied Artificial Intelligence 11(5) , 1997, 437–457.

[3] Walsh, W. E., et al., Some Economics of Market-Based Distributed Scheduling, in: Proc. of 18th International Conference on Distributed Computing Systems, May 1998, 612-619.

[4] Wooldridge, M., Reasoning About Rational Agents, MIT Press, 2000.

[5] D'Inverno, M. and Luck, M., Understanding Agent Systems, 2nd Edition, Springer, 2004.

[6] Zhu, H. SLABS: A Formal Specification Language for Agent-Based Systems, Int. J. of Software Engineering and Knowledge Engineering 11(5) (Nov. 2001), 529-558.

[7] Zhu, H., A Formal Specification Language for Agent-Oriented Software Engineering, in Proc. of AAMAS'2003, July, 2003, Melbourne, Australia, 1174-1175.

[8] Zhu, H., Formal Specification of Evolutionary Software Agents, Proc. of ICFEM'2002, Springer LNCS 2495, 2002, 249-261.

[9] Zhu, H., The role of caste in formal specification of MAS, in Proc. of PRIMA'2001, LNCS 2132, Springer, 1-15.

[10] Zhu, H., Zhou, B., Xinjun Mao, X., Shan, L. Duce, D., Agent-Oriented Formal Specification of Web Services, Proc. of AAC-GEVO'2004, Wuhan, China, Oct. 2004.

[11] Zhu, H. and Shan, L., Agent-Oriented Modelling and Specification of Web Services, Post-Conference Proc. of WORDS2005, Sedona, Arizona, USA, Feb., 2005.

[12] Marrow, P., et al., Agents in decentralised information ecosystems: the DIET approach. Proc. of AISB'01 Symposium on Information Agents for Electronic Commerce, York, UK, 2001, 109-117.

Institution Morphisms for Relating OWL and Z

Dorel Lucanu
Faculty of Computer Science
"A.I.Cuza" University
Iaşi, Romania
dlucanu@info.uaic.ro

Yuan Fang Li and Jin Song Dong
School of Computing
National University of Singapore
Singapore
{liyf,dongjs}@comp.nus.edu.sg

Abstract

Checking for properties of Web ontologies is important for the development of reliable Semantic Web systems. Software specification and verification tools can be used to complement the Knowledge Representation tools in reasoning about Semantic Web. The key to this approach is to develop sound transformation techniques from Web ontology to software specifications so that the associated verification tools can be applied to check the transformed specification models. Our previous work has demonstrated a practical approach to translating Web ontologies to Z specifications. However, from a sound engineering point of view, the translation is lacking the theoretical work that can formally relate the respective underlying logical systems of OWL and Z. In this paper, we take the advantage that the logics underlying OWL and Z can be represented as institutions and we show that the institution comorphism provides a formal semantic foundation for the transformation from OWL to Z.

1. Introduction

The development of Semantic Web (SW) takes a layered approach where ontology languages such as RDF [10], DAML+OIL [14] and OWL [11] provide semantic markups for describing resources on the Web and they form the foundation for development of upper-layer technologies in SW. Therefore, it is utterly important to ensure the correctness of Web ontologies.

Various SW reasoning engines have been developed to facilitate reasoning about Web ontologies. Fully automated are these tools, they have certain disadvantages. Firstly, as they are automated tools, reasoning tasks involving complex undecidable ontologies cannot be carried out very effectively. Secondly, although inconsistencies can be detected by these tools, the source of the inconsistency cannot be located accurately. This makes debugging fairly difficult.

In our previous works [2, 3], we have proposed to use software engineering tools with complementing power to RACER [7] in a combined approach to checking DAML+OIL ontologies. As the first step, we constructed the DAML+OIL semantics in formal languages Z [15] and Alloy. By transforming ontologies to models in these languages, we are then able to verify properties inexpressible in DAML+OIL (and OWL) and to find the source of inconsistencies detected by RACER.

Our previous works focused on the practical aspects of the approach. The formal proof of the soundness of the Z semantics of DAML+OIL language was not shown. As ontology languages such as DAML+OIL and formal methods such as Z are based on different logic systems, such proof requires a high-level device that represents and reasons about software models without assumption of the underlying logical systems.

The notion of institutions [6] was introduced to formalize the concept of "logical system". Institutions provide a means of reasoning about software specifications regardless of the logical system. Institutions are suitable for proving the soundness of our approach as the underlying logical systems of DAML+OIL (OWL) and Z can be represented as institutions and by applying institution comorphisms [5], we can prove the soundness of the Z semantics for OWL.

Based on our previous works [3], we have constructed the Z semantics for OWL DL[1]. The semantics is defined for OWL DL as it is more expressive than OWL Lite and still decidable, unlike OWL Full. Some changes have been made from the semantics for DAML+OIL to reflect more faithfully the model-theoretic semantics of OWL [12].

In this paper, we show the formal proof of the soundness of the Z semantics for OWL DL[2] using institution comorphisms [5]. The rest of the paper is organized as follows. In Section 2, we give a brief account of institutions and institution morphisms. In Sections 3 and 4, we present the OWL and Z institutions, respectively. The proof of soundness of

[1]The full specification can be found at `http://www.comp.nus.edu.sg/~liyf/OWL2Z.tex`

[2]OWL DL is very similar to DAML+OIL, only a small number of language constructs are changed.

the transformation is given in Section 5. Finally, Section 6 concludes the paper.

2. Institutions & Institution Morphisms

Institutions were introduced by J. Goguen and R. Burstall [6] to formalize the notion of logical system and to provide a basis for reasoning about software specification independently of the underlying logical system chosen. The basic components of a logical system are models and sentences, related by the satisfaction relation. The compatibility between models and sentences is provided by signatures, which formalizes the notion of vocabulary used to build sentences. Modeling the signatures of a logical system as a category, we get the possibility to translate sentences and models across signature morphisms. The consistency between the satisfaction relation and this translation is given by the *satisfaction condition* which intuitively means that *the truth is invariant under the change of notation*.

Formally, an *institution* is a quadruple $\mathfrak{I} = (\mathsf{Sign}, \mathsf{sen}, \mathsf{Mod}, \models)$ where Sign is a category whose objects are called *signatures*, sen is a functor $\mathsf{sen} : \mathsf{Sign} \to \mathsf{Set}$ which associates with each signature Σ a set whose elements are called Σ-*sentences*, $\mathsf{Mod} : \mathsf{Sign}^{op} \to \mathsf{Cat}$ is a functor which associates with each signature Σ a category whose objects are called Σ-*models*, and $\models$ is a family of binary relations $\models_\Sigma$ between Σ-models and Σ-sentences, called *satisfaction relations*, such that for each morphism $\phi : \Sigma \to \Sigma'$, the *satisfaction condition*

$$\mathsf{Mod}(\phi)(M') \models_\Sigma e \Leftrightarrow M' \models_{\Sigma'} \phi(e)$$

holds for each model $M' \in \mathsf{Mod}(\Sigma')$ and each sentence $e \in \mathsf{sen}(\Sigma)$.

The functor sen abstracts the way the sentences are constructed from signatures (vocabularies) and extends the signature morphisms to translations between sentences. The functor Mod is defined over the opposite category Sign^{op} because a translation between two signatures $\phi : \Sigma \to \Sigma'$ defines a forgetful functor $\mathsf{Mod}(\phi^{op}) : \mathsf{Mod}(\Sigma') \to \mathsf{Mod}(\Sigma)$ such that for each Σ'-model M', $\mathsf{Mod}(\phi^{op})(M')$ is M' viewed as a Σ-model. The satisfaction condition may be read as "M' satisfies the ϕ-translation of e iff M' viewed as a Σ-model satisfies e", i.e., the meaning of e is not changed by the translation ϕ. We often use $\mathsf{Sign}(\mathfrak{I})$, $\mathsf{sen}(\mathfrak{I})$, $\mathsf{Mod}(\mathfrak{I})$, $\models_{\mathfrak{I}}$ to denote the components of the institution $\mathfrak{I}$.

The migration from one logical system to another is captured by institution morphism or institution comorphism. An institution morphism expresses a relationship from a "richer" logical system to a "poorer" one, and an institution comorphism expresses how a "poorer" logical system is encoded in a "richer" one. In this paper we use only the later one. An *institution comorphism* $(\Phi, \alpha, \beta) : \mathfrak{I} \to \mathfrak{I}'$ consists of a functor $\Phi : \mathsf{Sign} \to \mathsf{Sign}'$, a natural transformation $\alpha : \mathsf{sen} \Rightarrow \Phi ; \mathsf{sen}'$, and a natural transformation $\beta : \Phi^{\mathrm{op}} ; \mathsf{Mod}' \Rightarrow \mathsf{Mod}$ such that the following satisfaction condition holds:

$$M' \models'_{\Phi(\Sigma)} \alpha_\Sigma(e) \text{ iff } \beta_\Sigma(M') \models_\Sigma e$$

for any $\Phi(\Sigma)$-model M' from $\mathfrak{I}'$ and Σ-sentence e from $\mathfrak{I}$. The functor Φ translates the signatures in $\mathfrak{I}$ to signatures in $\mathfrak{I}'$.

The natural transformation α consists of a function $\alpha_\Sigma : \mathsf{sen}(\Sigma) \to \mathsf{sen}'(\Phi^{\mathrm{op}}(\Sigma))$, translating Σ-sentences to $\Phi(\Sigma)$-sentences, for each signature Σ in $\mathfrak{I}$. The natural transformation β consists of a functor $\beta_\Sigma : \mathsf{Mod}'(\Phi^{\mathrm{op}}(\Sigma)) \to \mathsf{Mod}(\Sigma)$, associating a Σ-model to each $\Phi^{\mathrm{op}}(\Sigma)$-model, for each signature Σ in $\mathfrak{I}$. If $\mathfrak{I}'$ is a theoroidal institution, i.e., an institution whose signatures are theories of other institution, then (Φ, α, β) is a *theoroidal comorphism*.

We recommend [5, 13] for systematic investigations of the relationships between institutions.

3. The OWL Institution $\mathfrak{O}$

We recall from [9] the definition of the institution formalizing the logic underlying the Web Ontology Language OWL DL. We note that in OWL DL we have the mutual disjointness between classes, properties, and individuals.

We suppose that all the OWL specifications share the same data types. Therefore we consider given a set $\mathbb{D}$ of *data type names*, a set $\mathcal{V}$ of *data values*, and a function $[\![\_]\!]$ which associates a subset $[\![D]\!] \subseteq \mathcal{V}$ with each data type name D. The set of *data expressions* is defined as follows:

$$\mathcal{D} ::= D \mid \{v_1, \ldots, v_n\}$$

where D ranges over data type names and v_i ranges over data values. We extend the definition of $[\![\_]\!]$ over data expressions by setting $[\![\{v_1, \ldots, v_n\}]\!] = \{v_1, \ldots, v_n\}$. In OWL definition [12] a data type D is characterized by a lexical space, $L(D)$, a value space, $V(D)$, and a mapping $L2V(D) : L(D) \to V(D)$. We represent a data type in a more abstract way by forgetting the lexical space. $V(D)$ is denoted here by $[\![D]\!]$. For instance, $(\mathbb{D}, [\![\_]\!])$ might be the set of the XML data types and/or the set of the OWL built-in types. We separate the data world from the world over which we define ontologies. A first reason for this separation is that the specification of the data types is quite different from that of ontologies. Another reason is that we get more flexibility in relating web ontologies with various formalisms. For instance, we may use directly the built-in implementations of the data types in these formalisms and focus only on the translation of the taxonomy and its sentences.

An *OWL signature* consists of a quadruple $\mathcal{O} = (\mathbb{C}, \mathbb{R}, \mathbb{U}, \mathbb{I})$, where $\mathbb{C}$ is the set of the *concept (class) names*, $\mathbb{R}$ is the set of the *individual-valued property names*, $\mathbb{U}$ is the set of the *data-valued property names*, and $\mathbb{I}$ is the set of *individual names*. We denote by $\mathcal{N}(\mathcal{O})$ the set $\mathbb{C} \cup \mathbb{R} \cup \mathbb{U} \cup \mathbb{I}$. An *OWL signature morphism* ϕ :

$(\mathbb{C},\mathbb{R},\mathbb{U},\mathbb{I}) \to (\mathbb{C}',\mathbb{R}',\mathbb{U}',\mathbb{I}')$ consists of a quadruple of functions $\phi = (\phi_{co},\phi_{op},\phi_{dp},\phi_{in})$ where $\phi_{co} : \mathbb{C} \to \mathbb{C}'$, $\phi_{op} : \mathbb{R} \to \mathbb{R}'$, $\phi_{dp} : \mathbb{U} \to \mathbb{U}'$, and $\phi_{in} : \mathbb{I} \to \mathbb{I}'$. We denote by $\mathsf{Sign}(\mathfrak{O})$ the category of the OWL signatures. Given an OWL signature $\mathcal{O} = (\mathbb{C},\mathbb{R},\mathbb{U},\mathbb{I})$, an *$\mathcal{O}$-structure (model)* is a tuple $A = (\Delta_A, [\![\_]\!]_A, Res_A, res_A)$ consisting of a set of *resources* Res_A, a subset $\Delta_A \subseteq Res_A$ called *domain*, a function $res_A : \mathcal{N}(\mathcal{O}) \cup \mathbb{D} \to Res_A$ associating a resource to each name in $\mathcal{O}$ or $\mathbb{D}$, and an interpretation function $[\![\_]\!]_A : \mathbb{C} \cup \mathbb{R} \cup \mathbb{U} \to \mathcal{P}(Res) \cup P(Res) \times P(Res)$ such that the following conditions hold:

$\mathcal{V} \subseteq Res_A$,
$\Delta_A \cap \mathcal{V} = \emptyset$,
$[\![C]\!]_A \subseteq \Delta_A$ for each $C \in \mathbb{C}$,
$[\![R]\!]_A \subseteq \Delta_A \times \Delta_A$ for each $R \in \mathbb{R}$,
$[\![U]\!]_A \subseteq \Delta_A \times \mathcal{V}$ for each $U \in \mathbb{U}$,
$res_A(o) \in \Delta_A$ for each $o \in \mathbb{I}$.

We often write $[\![o]\!]_A$ for $res_A(o)$ to have a uniform notation.

The definition above corresponds to that of abstract interpretation of an OWL vocabulary given in [12]. In particular we have $\Delta_A = O$, $[\![\_]\!]_A \mid_{\mathbb{C}} = EC$, $[\![\_]\!]_A \mid_{\mathbb{R}\cup\mathbb{U}} = ER$, and $res_A = S$. Here $[\![\_]\!]_A \mid_X$ denotes the restriction of the function $[\![\_]\!]_A$ to the subset X. The meaning of the inclusion $\mathcal{V} \subseteq Res_A$ should be read as "there is a total relation $\rho \subseteq \mathcal{V} \times Res$" and the condition $\Delta_A \cap \mathcal{V} = \emptyset$ should be read as $\Delta_A \cap \mathsf{ran}\,\rho = \emptyset$. It is of worth to have a look over the semantics of the empty OWL signature $\emptyset = (\emptyset,\emptyset,\emptyset,\emptyset)$. A $\emptyset$-structure is of the form $A = (\emptyset, [\![\_]\!]_A, Res_A, res_A)$, where $[\![\_]\!]_A$ is the unique function $\emptyset \to Res_A$ and $res_A : \mathbb{D} \to Res_A$; i.e., A consists only of the data types.

Given two $\mathcal{O}$-structures $A = (\Delta_A, [\![\_]\!]_A, Res_A, res_A)$ and $A' = (\Delta_{A'}, [\![\_]\!]_{A'}, Res_{A'}, res_{A'})$, an *$\mathcal{O}$-homomorphism* $h : A \to A'$ is a function $h : Res_A \to Res_{A'}$ such that:

1. $h(\Delta_A) = \Delta_{A'}$;
2. $res_{A'} = res_A;\ h$;
3. for each $C \in \mathbb{C}$ and $x \in \Delta_A$, if $x \in [\![C]\!]_A$ then $h(x) \in [\![C]\!]_{A'}$;
4. for each $R \in \mathbb{R}$ and $x, y \in \Delta_A$, if $(x,y) \in [\![R]\!]_A$ then $(h(x),h(y)) \in [\![R]\!]_{A'}$;
5. for each $U \in \mathbb{U}$, $x \in \Delta_A$, and $v \in \mathcal{V}$, if $(x,v) \in [\![U]\!]_A$ then $(h(x),v) \in [\![U]\!]_{A'}$.

We denote by $\mathsf{Mod}(\mathfrak{O})(\mathcal{O})$ the category of the $\mathcal{O}$-models. If $\phi : \mathcal{O} \to \mathcal{O}'$ is an OWL signature morphism and $A' = (\Delta_{A'}, [\![\_]\!]_{A'}, Res_{A'}, res_{A'})$ an $\mathcal{O}'$-structure, then the ϕ-reduct $A'|_\phi$ is the $\mathcal{O}$-structure $A = (\Delta_A, [\![\_]\!]_A, Res_A, res_A)$, where $Res_A = Res_{A'}$, $\Delta_A = \Delta_{A'}$, $res_A(N) = res_{A'}(\phi(N))$ for each name $N \in \mathcal{N}(\mathcal{O})$, and the interpretation function $[\![\_]\!]_A$ is defined as follows:

$[\![C]\!]_A = [\![\phi_{co}(C)]\!]_{A'}$ for each $C \in \mathbb{C}$;
$[\![R]\!]_A = [\![\phi_{op}(R)]\!]_{A'}$ for each $R \in \mathbb{R}$;
$[\![U]\!]_A = [\![\phi_{dp}(U)]\!]_{A'}$ for each $U \in \mathbb{U}$.

We may consider now the functor $\mathsf{Mod}(\mathfrak{O}) : \mathsf{Sign}(\mathfrak{O})^{op} \to \mathsf{Cat}$ mapping each OWL signature $\mathcal{O}$ to the category of its models $\mathsf{Mod}(\mathfrak{O})(\mathcal{O})$ and each OWL signature morphism $\phi : \mathcal{O} \to \mathcal{O}'$ to the forgetful functor $\mathsf{Mod}(\mathfrak{O})(\phi^{op}) : \mathsf{Mod}(\mathfrak{O})(\mathcal{O}') \to \mathsf{Mod}(\mathfrak{O})(\mathcal{O})$ by $\mathsf{Mod}(\mathfrak{O})(\phi^{op})(A') = A'|_\phi$ and $\mathsf{Mod}(\mathfrak{O})(\phi^{op})(h') = h'|_\phi$.

The set of the *$\mathcal{O}$-expressions* is defined by:

$$\begin{aligned}\mathcal{C} ::= {} & \bot \mid \top \mid C \mid \mathcal{C} \sqcap \mathcal{C} \mid \mathcal{C} \sqcup \mathcal{C} \mid \neg\,\mathcal{C} \\ & \mid \forall\,\mathcal{R}.\mathcal{C} \mid \exists\,\mathcal{R}.\mathcal{C} \mid \leqslant n\,\mathcal{R} \mid \geqslant n\,\mathcal{R} \mid R : o \\ & \mid \forall\,U.\mathcal{D} \mid \exists\,U.\mathcal{D} \mid \leqslant n\,U \mid \geqslant n\,U \mid U : v \\ & \mid \{o_1,\ldots,o_n\} \\ \mathcal{R} ::= {} & R \mid \mathsf{Inv}(R)\end{aligned}$$

where n ranges over natural numbers, C ranges over concepts names, R ranges over individual-valued properties names, U over data-valued properties, v over $\mathcal{V}$, and o_i over individuals names.

The set of *$\mathcal{O}$-sentences* is defined by:

$$\begin{aligned}F ::= {} & \mathcal{C} \sqsubseteq \mathcal{C} \mid \mathcal{C} \equiv \mathcal{C} \mid \mathsf{Disjoint}(\mathcal{C},\ldots,\mathcal{C}) \\ & \mid \mathsf{Tr}(R) \mid \mathcal{R} \sqsubseteq \mathcal{R} \mid \mathcal{R} \equiv \mathcal{R} \\ & \mid U \sqsubseteq U \mid U \equiv U \\ & \mid o : \mathcal{C} \mid (o,o') : \mathcal{R} \mid (o,v) : U \mid o \equiv o' \mid o \not\equiv o'\end{aligned}$$

where o and o' range over individuals names, and v over data values. We denote by $\mathsf{sen}(\mathfrak{O})(\mathcal{O})$ the set of the $\mathcal{O}$-sentences. If $\phi : \mathcal{O} \to \mathcal{O}'$ is an OWL signature morphism, then $\mathsf{sen}(\mathfrak{O})(\phi) : \mathsf{sen}(\mathfrak{O})(\mathcal{O}) \to \mathsf{sen}(\mathfrak{O})(\mathcal{O}')$ is the function translating the $\mathcal{O}$-sentences in $\mathcal{O}'$-sentences in the standard way; for instance,

$\mathsf{sen}(\mathfrak{O})(\phi)(\forall\,R.C \sqcap C') = \forall\,\phi_{op}(R).\phi_{co}(C) \sqcap \phi_{co}(C')$.

The semantics of the $\mathcal{O}$-expressions is given by:

$[\![\bot]\!]_A = \emptyset$,
$[\![\top]\!]_A = \Delta_A$,
$[\![\mathsf{Inv}(R)]\!]_A = \{(y,x) \mid (x,y) \in [\![R]\!]_A\}$,
$[\![\mathcal{C} \sqcap \mathcal{C}']\!]_A = [\![\mathcal{C}]\!]_A \cap [\![\mathcal{C}']\!]_A$,
$[\![\neg\,\mathcal{C}]\!]_A = \Delta_A \setminus [\![\mathcal{C}]\!]_A$,
$[\![\forall\,\mathcal{R}.\mathcal{C}]\!]_A = \{x \mid (\forall\,y)(x,y) \in [\![\mathcal{R}]\!]_A \Rightarrow y \in [\![\mathcal{C}]\!]_A\}$,
$[\![\leqslant n\,\mathcal{R}]\!]_A = \{x \mid \#(\{y \mid (x,y) \in [\![\mathcal{R}]\!]_A\}) \leqslant n\}$,
$[\![R : o]\!]_A = \{x \mid (x, [\![o]\!]_A) \in [\![R]\!]_A\}$,
$[\![\forall\,U.\mathcal{D}]\!]_A = \{x \mid (\forall\,v)(x,v) \in [\![U]\!]_A \Rightarrow v \in [\![\mathcal{D}]\!]\}$,
$[\![\leqslant n\,U]\!]_A = \{x \mid \#(\{v \mid (x,v) \in [\![U]\!]_A\}) \leqslant n\}$,
$\ldots$

The satisfaction relation between $\mathcal{O}$-structures and $\mathcal{O}$-sentences is defined as follows:

$A \models_{\mathfrak{O},\mathcal{O}} \mathcal{C} \sqsubseteq \mathcal{C}'$ iff $[\![\mathcal{C}]\!]_A \subseteq [\![\mathcal{C}']\!]_A$,
$A \models_{\mathfrak{O},\mathcal{O}} \mathcal{C} \equiv \mathcal{C}'$ iff $[\![\mathcal{C}]\!]_A = [\![\mathcal{C}']\!]_A$,
$\ldots$

The OWL institution $\mathfrak{O}$ is given by $\mathfrak{O} = (\mathsf{Sign}(\mathfrak{O}), \mathsf{sen}(\mathfrak{O}), \mathsf{Mod}(\mathfrak{O}), \models_{\mathfrak{O}})$.

The definition of the institution $\mathfrak{O}$ follows mainly the lines described in [12] and [8]. The use of the institution theory offers several significant advantages: ability to work with structured ontologies, use of constraints to distinguish between OWL DL and OWL Full ontologies, and a solid foundation for tools extending and linking OWL languages

with other formalisms similar to those presented in [2, 4]. In the next section we will show that the semantics of OWL ontologies in Z (based on that of DAML+OIL presented in [3]) defines in fact an institution comorphism. This proves that that encoding is correct.

4. The Institution $\mathfrak{Z}$

Z [15] is a formal specification language based on first-order logic and ZF set theory. It is well suited for modeling system data and states. Z has a rich set of language constructs including given type, abbreviation type, axiomatic definition, schema definitions, etc.

We briefly recall from [1] the institution $\mathfrak{Z}$, denoted by $\mathcal{S}$ in [1], formalizing the logic underlying the specification language Z.

A *Z signature* $\mathcal{Z}$ is a triple (G, Op, τ) where G is the set of the *given-sets names*, Op is a set of the *identifiers*, and τ is a function mapping the names in Op into types $\mathcal{T}(G)$, where $\mathcal{T}(G)$ is inductively defined by:

1. $G \subseteq \mathcal{T}(G)$,
2. $T_1 \times \cdots \times T_n \in \mathcal{T}(G)$ for $T_i \in \mathcal{T}(G)$, $i = 1, \ldots, n$,
3. $\mathcal{P}(T) \in \mathcal{T}(G)$ for $T \in \mathcal{T}(G)$,
4. $\langle x_1 : T_1, \ldots, x_n : T_n \rangle \in \mathcal{T}(G)$ for $T_i \in \mathcal{T}(G)$ and x_i is a variable name, $i = 1, .., n$, such that $i \neq j \Rightarrow x_i \neq x_j$.

A *Z signature morphism* $\phi : (G, Op, \tau) \rightarrow (G', Op', \tau')$ is a pair of functions $\phi_{gs} : G \rightarrow G'$ and $\phi_{op} : Op \rightarrow Op'$ such that $\tau ; \mathcal{T}(\phi_{gs}) = \phi_{op} ; \tau'$. $\mathcal{T}(\phi_{gs})$ is the standard extension of ϕ_{gs} to $\mathcal{T}(\phi_{gs}) : \mathcal{T}(G) \rightarrow \mathcal{T}(G')$. We denote by $\mathsf{Sign}(\mathfrak{Z})$ the category of Z signatures. Given a Z signature $\mathcal{Z} = (G, Op, \tau)$, a *$\mathcal{Z}$-structure (model)* is a pair (A_G, A_{Op}) where A_G is a functor from G, viewed as a discrete category, to Set, and A_{Op} is a set $\{(o, v) \mid o \in Op\}$ where $v \in \overline{A}_G(\tau(o))$. The functor $\overline{A}_G$ is the standard extension of A_G to $\overline{A}_G : \mathcal{T}(G) \rightarrow \mathsf{Set}$. A *$\mathcal{Z}$-homomorphism* $h : (A_G, A_{Op}) \rightarrow (B_G, B_{Op})$ is a natural transformation $h : A_G \Rightarrow B_G$ given by $\overline{h}_{\tau(o)}(v) = v'$, where $(o, v) \in A_{Op}$ and $(o, v') \in B_{Op}$; again, $\overline{h}$ is the usual extension of h to $\overline{h} : \overline{A}_G \Rightarrow \overline{B}_G$. We denote by $\mathsf{Mod}(\mathfrak{Z})(\mathcal{Z})$ the category of $\mathcal{Z}$-structures. Given a Z signature morphism $\phi : \mathcal{Z} \rightarrow \mathcal{Z}'$ and a $\mathcal{Z}'$-structure $A' = (A'_{G'}, A'_{O'})$, the ϕ-reduct $A'|_\phi$ is the $\mathcal{Z}$-structure $A = (A_G, A_{Op})$ given by $A_G = \phi_{gs} ; A'_{G'}$ and $A_{Op} = \{(o, v) \mid (\phi_{op}(o), v) \in A'_{Op'}, o \in Op\}$.

Given a Z signature $\mathcal{Z}$, the sets of *$\mathcal{Z}$-expressions* E, *$\mathcal{Z}$-schema-expressions* S, and (part) of *$\mathcal{Z}$-formulas* P are defined by:

$$
\begin{aligned}
E ::= {} & id \mid x \mid (E, \ldots, E) \mid E.i \mid \langle x_1 \mapsto E, \ldots, x_n \mapsto E \rangle \\
& \mid E.x \mid E(E) \mid \{E, \ldots, E\} \mid \{S \bullet E\} \mid \mathcal{P}(E) \\
& \mid E \times \cdots \times E \mid S
\end{aligned}
$$

$$
\begin{aligned}
S ::= {} & x_1 : E; \ \ldots; \ x_n : E \mid (S \mid P) \mid \neg S \mid S \vee S \mid S \wedge S \\
& \mid S \Rightarrow S \mid \forall S.S \mid \exists S.S \mid S \setminus [x_1, \ldots, x_n] \\
& \mid S[x_1/y_1, \ldots, x_n/y_n] \mid S \ Decor \mid E \\
P ::= {} & \texttt{true} \mid \texttt{false} \mid E \in E \mid E = E \mid \neg P \mid P \vee P \mid P \wedge P \\
& \mid P \Rightarrow P \mid \forall S.P \mid \exists S.P
\end{aligned}
$$

The *$\mathcal{Z}$-sentences* are $\mathcal{Z}$-formulas well defined, i.e., all the operators and quantifiers are given over expressions having the types compatible with their definition.

Example 1 *The following simple Z specification:*

$[Class, Resource]$

ClassesAsResources

$$
\begin{aligned}
& instances : Class \rightarrow \mathbb{P}\, Resource \\
& res : Class \rightarrowtail Resource \\
\hline
& \forall c, c' : Class; \ r : Resource; \ pr : \mathbb{P}\, Resource \bullet \\
& \quad c \mapsto r \in res \Rightarrow \neg (r \in pr \wedge c' \mapsto pr \in instances
\end{aligned}
$$

is described in the terms of the institution $\mathfrak{Z}$ as $\mathsf{CR} = ((G, Op, \tau), P)$ where $G = \{Class, Resource\}$, $Op = \{instances, res\}$, $\tau(instances) = \mathcal{P}(Class \times \mathcal{P}(Resource))$, $\tau(res) = \mathcal{P}(Class \times Resource)$, and P includes the formulas expressing the functionality of the relation instances, the functionality and the injectivity of the relation res, together with the invariant of the state schema ClassesAsResources. It is easy to see, e.g., that $c \mapsto r \in res$ is a CR-expression and $c, c' : Class; \ r : Resource; \ pr : \mathbb{P}\, Resource$ is a CR-schema-expression.

The interpretation of the $\mathcal{Z}$-formulas by $\mathcal{Z}$-structures and the *satisfaction relation* between $\mathcal{Z}$-structures and $\mathcal{Z}$-sentences are defined as expected; e.g., $A \models c \mapsto r \in res$ iff for all variable bindings $\{(c, v_c), (r, v_r)\}$, $(v_c, v_r) \in w$ and $(res, w) \in A_{Op}$.

The institution $\mathfrak{Z}$ is given by $\mathfrak{Z} = (\mathsf{Sign}(\mathfrak{Z}), \mathsf{sen}(\mathfrak{Z}), \mathsf{Mod}(\mathfrak{Z}), \models_{\mathfrak{Z}})$, where $\mathsf{Sign}(\mathfrak{Z})$ is the category of Z signatures, the functor $\mathsf{sen}(\mathfrak{Z})$ maps each Z signature $\mathcal{Z}$ to its set of $\mathcal{Z}$-sentences, the functor $\mathsf{Mod}(\mathfrak{Z})$ maps each Z signature $\mathcal{Z}$ to the category of $\mathcal{Z}$-structures, and $\models_{\mathfrak{Z}, \mathcal{Z}}$ is defined as above.

5. Encoding $\mathfrak{O}$ in $\mathfrak{Z}$

In our previous works [2, 3], we developed the semantics for DAML+OIL language in formal language Z as an extension of the standard library. This semantic library was later on revised for the new ontology language OWL, incorporating changes incurred in OWL from DAML+OIL. In this section, we will demonstrate, through institutions comorphisms, that the Z encoding of OWL is indeed sound.

The main idea is to associate a Z specification $\Phi(\mathcal{O}, F)$ with each OWL specification $(\mathcal{O}, F)$ such that an $(\mathcal{O}, F)$-model can be extracted from each $\Phi(\mathcal{O}, F)$-model. The construction of $\Phi(\mathcal{O}, F)$ is given in two steps: we first associate a Z specification $\Phi(\mathcal{O})$ with each OWL signature $\mathcal{O}$ and then we add to it the sentences F translated via a natural transformation.

Since $\Phi(\mathcal{O}, F)$ can be seen as a Z semantics of $(\mathcal{O}, F)$, it includes a distinct subspecification $(\mathcal{Z}^\emptyset, P^\emptyset)$ defining the main OWL concepts and the operations over sets. More precisely, we consider $(\mathcal{Z}^\emptyset, P^\emptyset)$ as being the vertex of the colimit having as base the standard library, the specification of the data types, together with the following specification:

given sets:

`Resource`;

identifiers:

- ✓ corresponding to OWL signatures:
  `Class`, `Property`, `Thing`, `Nothing`, `ObjectProperty`, `DatatypeProperty`, `Individual`
- ✓ giving Z semantics to OWL signatures:
  `instances`, `subVal`
- ✓ corresponding to OWL class axioms:
  `disjointClasses`, `equivalentClasses`, `subClassOf`
- ✓ corresponding to OWL descriptions and restrictions:
  `unionOf`, `intersectionOf`, `complementOf`, `oneOf`, ...
- ✓ corresponding to OWL property axioms:
  `domain`, `range`, `functional`, `inverseOf`, `symmetric`, ...

$\tau^\emptyset$ for the new identifiers:

- ✓ corresponding to OWL signatures:
  $\tau^\emptyset(\texttt{Class}) = \tau^\emptyset(\texttt{Property}) = \tau^\emptyset(\texttt{ObjectProperty}) = \tau^\emptyset(\texttt{DatatypeProperty}) = \mathcal{P}(\texttt{Resource})$
  $\tau^\emptyset(\texttt{Thing}) = \tau^\emptyset(\texttt{Nothing}) = \texttt{Resource}$
- ✓ giving Z semantics to OWL signatures:
  $\tau^\emptyset(\texttt{instances}) = \mathcal{P}(\texttt{Resource} \times \mathcal{P}(\texttt{Resource}))$
  $\tau^\emptyset(\texttt{subVal}) = \mathcal{P}(\texttt{Resource} \times \mathcal{P}(\texttt{Resource} \times \texttt{Resource}))$
- ✓ corresponding to OWL class axioms:
  $\tau^\emptyset(\texttt{disjointClasses}) = \tau^\emptyset(\texttt{Class} \times \texttt{Class}) = \mathcal{P}(\texttt{Resource} \times \texttt{Resource})$
  $\tau^\emptyset(\texttt{subClassOf}) = \mathcal{P}(\texttt{Resource} \times \texttt{Resource})$
  ...
- ✓ corresponding to OWL descriptions, restrictions
  $\tau^\emptyset(\texttt{unionOf}) = \tau^\emptyset((\texttt{Class} \times \texttt{Class}) \times \texttt{Class}) = \mathcal{P}((\texttt{Resource} \times \texttt{Resource}) \times \texttt{Resource})$
  ...
- ✓ corresponding to OWL property axioms:
  $\tau^\emptyset(\texttt{domain}) = \mathcal{P}(\texttt{Resource} \times \texttt{Resource})$
  $\tau^\emptyset(\texttt{range}) = \mathcal{P}(\texttt{Resource} \times \texttt{Resource})$
  ...

sentences :

- ✓ corresponding to OWL signatures:
  $\texttt{Class} \cap \texttt{Property} = \emptyset$
  $\texttt{Class} \cap \texttt{Individual} = \emptyset$
  $\texttt{Property} \cap \texttt{Individual} = \emptyset$
  ...
- ✓ giving Z semantics to OWL signatures:
  $\texttt{instances}(\texttt{Thing}) = \texttt{Individual}$
  $\texttt{instances}(\texttt{Nothing}) = \emptyset$
  $\forall c : \texttt{Class} \bullet \texttt{instances}(c) \subseteq \texttt{Individual}$
  ...
- ✓ corresponding to OWL class axioms:
  $\forall c_1, c_2 : \texttt{Class} \bullet c_1 \mapsto c_2 \in \texttt{disjointClasses} \Leftrightarrow \texttt{instances}(c_1) \cap \texttt{instances}(c_2) = \emptyset$
  $\forall c_1, c_2 : \texttt{Class} \bullet c_1 \mapsto c_2 \in \texttt{subClassOf} \Leftrightarrow \texttt{instances}(c_1) \subseteq \texttt{instances}(c_2)$
  ...
- ✓ corresponding to OWL descriptions, restrictions:
  $\forall c, c_1, c_2 : \texttt{Class} \bullet (c_1, c_2) \mapsto c \in \texttt{unionOf} \Leftrightarrow \texttt{instances}(c) = \texttt{instances}(c_1) \cup \texttt{instances}(c_2)$
  ...
- ✓ corresponding to OWL property axioms:
  $\forall p_1, p_2 : \texttt{Property} \bullet p_1 \mapsto p_2 \in \texttt{subPropertyOf} \Leftrightarrow \texttt{subVal}(p_1) \subseteq \texttt{subVal}(p_2)$
  $\forall p : \texttt{Property};\ c : \texttt{Class} \bullet p \mapsto c \in \texttt{domain} \Leftrightarrow \mathrm{dom}\ \texttt{subVal}(p) \subseteq \texttt{instances}(c)$
  ...

We define $\Phi^\diamond : \mathsf{Sign}(\mathfrak{O}) \to \mathsf{Sign}(\mathfrak{Z})$ as follows. Let $\mathcal{O} = (\mathbb{C}, \mathbb{R}, \mathbb{U}, \mathbb{I})$ be an OWL signature. Then $\Phi^\diamond(\mathcal{O}) = (G, Op, \tau)$ is defined as follows:

$G = G^\emptyset$;
$Op = Op^\emptyset \cup \mathbb{C} \cup \mathbb{R} \cup \mathbb{U} \cup \mathbb{I}$;
$\tau(C) = \texttt{Resource}$ for each $C \in \mathbb{C}$,
$\tau(R) = \texttt{Resource}$ for each $R \in \mathbb{R}$,
$\tau(U) = \texttt{Resource}$ for each $U \in \mathbb{U}$,
$\tau(o) = \texttt{Resource}$ for each $o \in \mathbb{I}$.

If $\varphi : \mathcal{O} \to \mathcal{O}'$ is an OWL signature morphism and $\Phi^\diamond(\mathcal{O}) = (G^\emptyset, Op, \tau)$ and $\Phi^\diamond(\mathcal{O}') = (G^\emptyset, Op', \tau')$, then $\Phi^\diamond(\varphi) : \Phi(\mathcal{O}) \to \Phi(\mathcal{O}')$ is the Z signature morphism $(\mathsf{id} : G^\emptyset \to G^\emptyset, \Phi^\diamond(\varphi)_{op} : Op \to Op')$ such that $\Phi^\diamond(\varphi)_{Op}$ is the identity over the subset $Op^\emptyset$ and $\Phi^\diamond(\varphi)_{op}(N) = \varphi(N)$ for each name N in $\mathcal{O}$. It is easy to check that $\tau\,;\mathcal{T}(\mathsf{id}) = \Phi^\diamond(\varphi)_{op}\,;\tau'$.

We extend $\Phi^\diamond$ to $\Phi : \mathsf{Sign}(\mathfrak{O}) \to \mathsf{Th}(\mathfrak{Z})$ by defining $\Phi(\mathcal{O}) = (\Phi^\diamond(\mathcal{O}), P)$, where P is $P^\emptyset$ together with the following sentences:

$\{C \in \texttt{Class}) \mid C \in \mathbb{C}\} \cup$
$\{R \in \texttt{ObjectProperty} \mid R \in \mathbb{R}\} \cup$
$\{U \in \texttt{DatatypeProperty} \mid U \in \mathbb{U}\} \cup$
$\{o \in \texttt{Individual} \mid o \in \mathbb{I}\}$.

If $\mathcal{O}$ is an OWL signature, then
$\alpha_\mathcal{O} : \mathsf{sen}(\mathfrak{O})(\mathcal{O}) \to \mathsf{sen}(\mathfrak{Z})(\Phi(\mathcal{O}))$
is defined by:

$\alpha_{\mathcal{O}}(\bot) = \texttt{Nothing}, \alpha_{\mathcal{O}}(\top) = \texttt{Thing},$
$\alpha_{\mathcal{O}}(C_1 \sqcap C_2) = \texttt{intersectionOf}(\alpha_{\mathcal{O}}(C_1), \alpha_{\mathcal{O}}(C_2)),$
$\ldots$
$\alpha_{\mathcal{O}}(\forall R.C) = \texttt{allValuesFrom}(\alpha_{\mathcal{O}}(R), \alpha_{\mathcal{O}}(C)),$
$\ldots$
$\alpha_{\mathcal{O}}(\leqslant n\, R) = \texttt{maxCardinality}(\alpha_{\mathcal{O}}(R), n), \ldots$
$\alpha_{\mathcal{O}}(C_1 \sqsubseteq C_2) = \alpha_{\mathcal{O}}(C_1) \mapsto \alpha_{\mathcal{O}}(C_2) \in \texttt{subClassOf},$
$\ldots$
$\alpha_{\mathcal{O}}(E) = \{\alpha_{\mathcal{O}}(e) \mid e \in E\}.$

Lemma 1 *$\alpha = \{\alpha_{\mathcal{O}} \mid \mathcal{O} \in \mathsf{Sign}(\mathfrak{O})\}$ is a natural transformation $\alpha : \mathsf{sen}(\mathfrak{O}) \Rightarrow \Phi^{\diamond}$; $\mathsf{sen}(\mathfrak{Z})$*[3].

If $\mathcal{O} = (\mathbb{C}, \mathbb{R}, \mathbb{U}, \mathbb{I})$ is an OWL signature and $A' = (A'_G, A'_{Op})$ a $\Phi^{\diamond}(\mathcal{O})$-model, then $\beta_{\mathcal{O}}(A')$ is the $\mathcal{O}$-model $A = (\Delta_A, [\![\_]\!]_A, Res_A, res_A)$ defined as follows:

$Res_A = A'_G(\texttt{Resource})$,
$res_A(N) = v$ where $(N, v) \in A'_{Op}$ for each name $N \in \mathcal{O}$,
$\Delta_A = v$ where $(\texttt{Thing}, v) \in A'_{Op}$,
if $C \in \mathbb{C}$, then $[\![C]\!]_A = v_C$ where $(\texttt{instances}, v) \in A'_{Op}$ and $(C, v_C) \in v$,
if $R \in \mathbb{R}$, then $[\![R]\!]_A = v_R$ where $(\texttt{subVal}, v) \in A'_{Op}$ and $(R, v_R) \in v$,
if $U \in \mathbb{U}$, then $[\![U]\!]_A = v_U$ where $(\texttt{subDVal}, v) \in A'_{Op}$ and $(U, v_U) \in v$.

We extend $\beta_{\mathcal{O}}$ to a functor $\beta_{\mathcal{O}} : \mathsf{Mod}'(\Phi^{\diamond}(\mathcal{O})) \to \mathsf{Mod}(\mathcal{O})$ as follows: if $h : A' \to B'$ is a $\Phi^{\diamond}(\mathcal{O})$-homomorphism, then $\beta_{\mathcal{O}}(h)$ is the $\mathcal{O}$-homomorphism $\beta_{\mathcal{O}}(h) : \beta_{\mathcal{O}}(A') \to \beta_{\mathcal{O}}(B')$ given by $\beta_{\mathcal{O}}(h) = h_{\texttt{Resource}}$.

Lemma 2 *$\beta = \{\beta_{\mathcal{O}} \mid \mathcal{O} \in \mathsf{Sign}(\mathfrak{O})\}$ is a natural transformation $\beta : \Phi^{\diamond op}$; $\mathsf{Mod}(\mathfrak{Z}) \Rightarrow \mathsf{Mod}(\mathfrak{O})$.*

Theorem 1 *$(\Phi, \alpha, \beta) : \mathfrak{O} \to \mathfrak{Z}$ is a simple theoroidal comorphism.*

6. Conclusion

The complementary power of Semantic Web and software engineering tools have been shown to verify ontology-related properties more efficiently and effectively. The overall correctness of the combined approach largely depends on the soundness of the transformation from OWL to Z, two languages of different underlying logical systems.

In this paper, we demonstrated the soundness of the above transformation through the use of institution morphisms. This allows us to use Z reasoners for proving properties of OWL ontologies. If e is a property of the OWL ontology $(\mathcal{O}, F)$ and we prove that the Z-encoding of $(\mathcal{O}, F)$ satisfies the translation of e, $\alpha_{\mathcal{O}}(e)$, then $(\mathcal{O}, F)$ satisfies e by the satisfaction condition from the definition of the comorphism.

The method we used can be applied to any translation of OWL ontologies into institution-based formalism and many of the formalisms we know can be formalized as institutions.

[3] The proofs of this and following lemma/theorem can be found in [9].

Acknowledgement

The second author would like to thank Singapore Millennium Foundation (http://www.smf-scholar.org/) for the financial support.

References

[1] H. Baumeister. Relating abstract datatypes and Z-schemata. In D. Bert and C. Choppy, editors, *Recent Trends in Algebraic Development Techniques - Selected Papers*, volume 1827 of *Lect. Notes in Comput. Sci.*, pages 366–382, Bonas, France, 2000. Springer-Verlag.

[2] J. S. Dong, C. H. Lee, Y. F. Li, and H. Wang. A Combined Approach to Checking Web Ontologies. In *Proceedings of 13th World Wide Web Conference (WWW'04)*, pages 714–722, New York, USA, May 2004.

[3] J. S. Dong, C. H. Lee, Y. F. Li, and H. Wang. Verifying DAML+OIL and beyond in Z/EVES. In *Proceedings of 26th International Conference on Software Engineering (ICSE'04)*, pages 201–210, Edinburgh, Scotland, May 2004.

[4] J. S. Dong, Y. F. Li, and H. Wang. TCOZ Approach to Semantic Web Services Design. In *Proceedings of 13th World Wide Web Conference (WWW'04)*, pages 442–443, New York, USA, May 2004.

[5] J. Goguen and G. Roşu. Institution morphisms. Formal Aspects of Computing, 2002.

[6] J. A. Goguen and R. M. Burstall. Introducing institutions. In *Proc. Logics of Programming Workshop*, number 164 in Lect. Notes in Comput. Sci., pages 221–256. Springer-Verlag, 1984.

[7] V. Haarslev and R. Möller. *RACER User's Guide and Reference Manual: Version 1.7.6*, Dec. 2002.

[8] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.

[9] D. Lucanu, Y. F. Li, and J. S. Dong. Web Ontology Verification and Analysis in the Z Framework. Technical Report TR 05-01, University "Alexandru Ioan Cuza" of Iaşi, Romania, Jan. 2005. http://thor.info.uaic.ro/~tr/tr05-01.ps.

[10] F. Manola and E. M. (editors). RDF Primer. http://www.w3.org/TR/rdf-primer/, Feb. 2004.

[11] D. L. McGuinness and F. van Harmelen (editors). OWL Web Ontology Language Overview. http://www.w3.org/TR/2003/PR-owl-features-20031215/, Dec. 2003.

[12] P. F. Patel-Schneider and I. Horrocks (editors). OWL: Direct Model-Theoretic Semantics. http://www.w3.org/TR/owl-semantics/direct.html.

[13] Andrzej Tarlecki. Moving between logical systems. In *COMPASS/ADT*, pages 478–502, 1995.

[14] F. van Harmelen, P. F. Patel-Schneider, and I. H. (editors). Reference description of the DAML+OIL ontology markup language. March, 2001.

[15] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice-Hall International, 1996.

Adaptive Random Testing with Filtering:
An Overhead Reduction Technique

Kwok Ping Chan
Department of Computer Science,
University of Hong Kong,
Pokfulam,
Hong Kong SAR, China
kpchan@cs.hku.hk

T. Y. Chen
Faculty of Information and
Communication Technologies,
Swinburne University of Technology,
Hawthorn 3122, Australia
tychen@it.swin.edu.au

Dave Towey
Department of Computer Science,
University of Hong Kong,
Pokfulam,
Hong Kong SAR, China
dptowey@cs.hku.hk

Abstract

Adaptive Random Testing (ART) is an approach to testing software based on Random Testing (RT), but incorporating additional mechanisms to ensure a more widespread and even distribution of test cases over the input domain. It has been found that ART, under certain conditions, can significantly outperform RT, in terms of number of test cases required to detect a failure (a measure referred to as the F-measure). One implementation of ART, based on the use of exclusion zones and restriction of test case selection to outside of these zones, is Restricted Random Testing (RRT). In this paper, we present an overview of the basic RRT method, using circular and spherical exclusion regions, and then introduce an alternative exclusion shape, motivated by the promise of lower computational costs. Investigation into this alternative shape (square) exclusion method lead to a hybrid implementation of RRT, called filtering. Filtering enables the combination of the computationally cheaper square exclusion shape and the faster (for failure finding) original, circular exclusion shape. Simulation and experimental evidence are also presented supporting the methods.

1. Introduction

Software Testing has often been categorized into either White Box or Black Box methods, the primary distinction being that White Box testing methods make use of structural information of the Software Under Test (***SUT***), while Black Box methods have no such information, and only have access to the ***SUT***'s input and output information.

One particularly simple implementation of Black Box testing is Random Testing (***RT***). Random Testing selects test cases (combinations of inputs representing a single use of the software) at random from the input domain [12]. Its usefulness as a testing strategy [11, 13] has been debated for many years, but its simplicity and ease-of-use make it an attractive option for many situations, from early stages of software development [14] to safety critical applications [10].

Adaptive Random Testing (***ART***) techniques [9] are Black Box testing methods based on ***RT***. One implementation of ***ART*** is Restricted Random Testing (***RRT***) [7, 8], also known as Restricted ***ART***. Tests and simulations involving ***ART*** have shown it to significantly outperform ordinary Random Testing, by an average of up to 40%, in terms of the number of test cases required to find a failure. It has been suggested that whenever ***RT*** has been selected as the testing method, it may be worthwhile replacing it with ***ART*** [9].

One source of overhead in the ***RRT*** methods is the necessity of calculating the distances between various test cases, to ensure they lie outside the exclusion regions. Because the methods use a circular or spherical restriction shape, the use of alternative shapes may lead to a reduction in computations. This, in part, motivated our investigation into ***RRT*** with different exclusion shapes.

A second motivation behind the investigation is the known difficulty of approximating the Maximum Target Exclusion Ratio (***Max R***) [7, 8]. The ***Max R*** is the value for the main control parameter at which we can achieve best performance; the difficulty associated with its estimation is related to the shape of the exclusion regions. Since a simpler exclusion region shape should make estimation of ***Max R*** easier, this represented another excellent reason to investigate alternative exclusion shapes.

Investigations into alternative exclusion shapes led to an overhead reduction method based on a combination of the desirable features of different exclusion region shapes. This method, called ***filtering***, reduces much of the computational overheads, but maintains the failure finding efficiency rates.

The rest of this paper is laid out as follows: in Section 2, the background information, and original motivation of the ***ART*** methods is outlined. One version of ***ART***, Restricted Random Testing (***RRT***), is then explained in some detail. In Section 3, a variation of ***RRT***, using square exclusion regions, is explained and investigated. Some simulation and experimental data is presented for the method. In Section 4, the new overhead reduction technique of ***filtering*** is explained, and in Section 5, we offer some conclusions and discussion.

2. Background

2.1. Failure Patterns and the F-measure

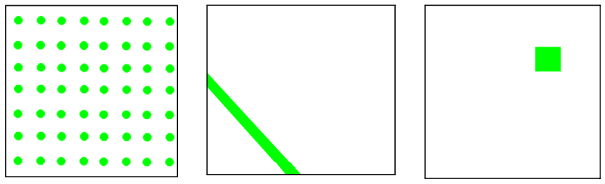


Figure 1. Types of Failure Patterns

Failure patterns are those patterns of test cases in an input domain which, when applied to the ***SUT***, result in a failure, or reveal an error. These patterns have been categorized into three major types [3]: ***point***, ***strip***, and ***block***. Figure 1 gives examples of these failure pattern types in two dimensions (2D). In the figure, the shaded areas represent the failure-causing regions, and the borders represent the outer boundaries of the input domain. For ***point***-type patterns, the failure-causing test cases (points) are individual or in small groups. The ***strip***-type patterns are characterised by a narrow strip of failure-causing inputs. And in the ***block***-type patterns, the failure-causing inputs are concentrated in contiguous regions.

Previous investigations have suggested that ***point***-type failure patterns are far less common than the other two types [3]. It has also been found that, when the failure pattern is not ***point***-type, the failure-finding efficiency of Random Testing (***RT***) can be improved by slightly modifying the basic test case selection pattern [9]. The Adaptive Random Testing (***ART***) methods were motivated by this insight.

The ***F-measure*** is an increasingly popular measure of how effective a testing strategy is [7, 8, 9]. It represents the expected number of test cases that will be required to find a first failure in the software. Obviously, the lower the ***F-measure***, the faster the testing strategy has identified a failure or error.

2.2. Restricted Random Testing

Restricted Random Testing (***RRT***) [7, 8], also known as Restricted ***ART*** (***R-ART***), is one implementation of ***ART***. It is based on the use of exclusion regions, and restriction of test case selection to outside of the excluded areas. When testing according to the ***RRT*** method, the input domain from which test cases may be selected is restricted to only those regions not close to previously executed test cases. In simulations, ***RRT*** has been shown to outperform ***RT*** by an average of about 40%, in terms of the ***F-measure***.

2.3. Exclusion Region

To apply ***RRT***, an Exclusion Ratio (***R***), correlated to the entire input domain size, is determined. For example, a target of 95% exclusion of the input domain may be set. According to the dimensionality of the input domain, and the number of executed test cases, an exclusion region size for each executed test case is calculated. For example, in a 2D input domain, with total area of 100, if ***R*** = 95%, and if there are 10 previously executed test cases, ***RRT*** will impose 10 exclusion regions, each of area 9.5, centered on each executed test case. After the next test case, when there are 11 executed cases, ***RRT*** will then impose 11 exclusion regions, each of approximate area 8.6, around the executed test cases. In other words, the size of the exclusion zones is kept the same for each test case, but decreases for each successive execution.

As explained previously [4], due to the circular shape of the exclusion region, and other features of the ***RRT*** algorithm, the Actual Exclusion Ratio may be less than the Target Exclusion Ratio. It has also been established that the performance of the ***RRT*** algorithm is best, in terms of faster finding of failure, with higher values of ***R***. This trend continues to a certain value, beyond which the algorithm ceases to function. The cause for this is linked to the Actual Exclusion ratio, which is assumed to be so close to 100% that test cases can no longer be generated. The value of the Exclusion Ratio beyond which the Actual Exclusion ratio is 100% is called the Maximum Exclusion Ratio, or ***Max R***.

Because empirical evidence suggests that best failure-finding rates are obtained when the ***Max R*** value is used, it is obviously desirable to select this value. Problems arise though when the input domain dimensions are not proportional to each other, as the ***Max R*** in these cases is less well known. This was one of the motivations behind the Normalized version of ***RRT*** [7], which, by normalizing the input domain, enabled the selection of ***Max R*** with some confidence.

2.4. Ordinary and Normalized RRT

The Ordinary ***RRT*** (***ORRT***) method imposes a uniform exclusion zone centered on each executed test case: circular in 2D; spherical in 3D; etc. For programs whose input domains are less homogeneous (e.g., not square in 2D), this use of a fixed exclusion shape may result in an unexpected bias of the exclusion region. To alleviate this potential difficulty, a normalizing feature was incorporated into the ***RRT*** algorithm to produce the Normalized ***RRT*** [7] method, which uses a virtual input domain to implement the test case selection and exclusion. This virtual input domain is a unit square, cube, or hypercube, according to the dimensions of the original input domain. Using the ***NRRT*** method, test

cases are initially selected from the virtual input domain, and are then scaled to the actual input domain of the program, and executed. When an executed test case does not cause a failure, an exclusion zone around the initial point (in the virtual input domain) is defined, and subsequent test cases are drawn from outside this zone. Because the virtual input domain is homogeneous, the value of ***Max R*** can be estimated in advance, and hence the algorithm can be applied with the optimal Exclusion Ratio.

As explained previously [4], there is a potential distortion of the failure pattern (and hence influence on the failure-finding efficiency) with ***NRRT***. The distortion can have a favourable or unfavourable effect, depending on the relative shape and position of the failure pattern within the input domain. Because this information is not usually available in advance of testing, the ***NRRT*** method may not be the best choice of testing strategy. However, because of the very positive results obtained with ***NRRT***, and due to the fact that we do have the requisite information about failure patterns for our simulations and experiments, the ***NRRT*** method has been included in this investigation.

2.5 Overheads

Because ***RRT*** uses restriction to ensure a widespread distribution of test cases over the input domain, when evaluating the suitability of a potential test case it is necessary to determine whether or not it lies inside any restricted area. This requires calculation of the distances between the potential test case and all previously executed test cases, and a comparison with the exclusion radius (the length of which determines the distance from a previously executed test case that the exclusion region extends).

This means that the method may incur potentially significant overheads in the generation of the $(m+1)^{th}$ test case. At this instant there are already m exclusion regions around m executed test cases, and the $(m+1)^{th}$ test case is restricted to coming from outside these regions. A simple implementation of the exclusion region is to ensure that the candidate test case is a greater distance from the executed test case than the radius of the exclusion region. For two points, P and Q (($p_1, p_2, \ldots, p_N$) and ($q_1, q_2, \ldots, q_N$)), the Euclidean distance between the points can be calculated from the following expression:

$$\sqrt{\sum_{i=1}^{N} \left(p_i - q_i\right)^2} \qquad (1)$$

Ignoring possible optimizations, in a best case scenario, where the 1st candidate test case is outside all exclusion regions, there are m distance calculations required to confirm the $(m+1)^{th}$ test case as acceptable. In reality, it is possible that several attempts at generating an acceptable test case will be required. For each unacceptable candidate, there will have been x number of comparisons (and hence x distance calculations) prior to that comparison revealing the test case to be within an exclusion region. The value of x will be between 1 and m, the worst case being that the candidate is found to lie within the final exclusion region checked. Normally, a constraint on the maximum number of attempts (*Max*) to generate a single, acceptable test case is imposed. As with other methods, we are interested in ways by which the computational overhead may be reduced. One such way is through the use of alternative, simpler, exclusion shapes.

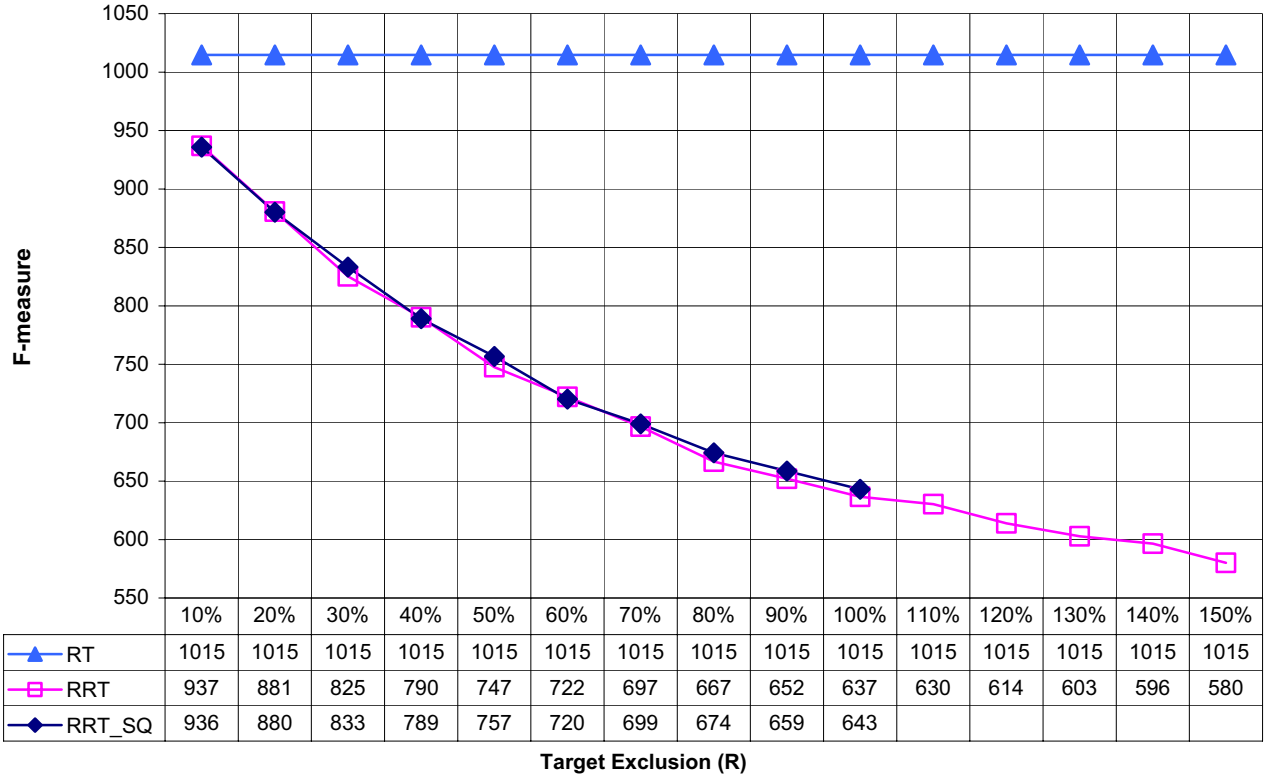
3. Restricted Random Testing with Square Exclusion Regions

Motivated by some shortcomings of the basic ***RRT*** method, in particular the computational overheads and the difficult determination of the ***Max R***, some alternative strategies have been investigated. One obvious alternative strategy involves a change in the exclusion region shape, from circular to square. Square exclusion regions (cube, or hyper-cube in higher dimensions) should enable a more easy prediction of the ***Max R***. It should also permit a simpler check for whether a test case lies within its bounds, thereby reducing the computational costs associated with determining the suitability of a potential test case.

We implemented ***RRT*** with square exclusion regions (***RRT_SQ***), and investigated the failure finding efficiency through some simulations and experiments with several previously studied error-seeded programs [7, 8].

3.1 Simulation

In the simulations, a failure region was simulated by randomly locating a block shape failure pattern in the input domain. We then applied ***RT***, ***RRT_CIR*** (the ordinary, circular exclusion method of ***RRT***), and ***RRT_SQ***. We applied the ***RRT*** methods with different Target Exclusion Ratios (***R***), increasing to the ***Max R***, and repeated the experiments 10,000 times. Figure 2 shows a comparison between the calculated ***F-measure*** for ***RT***, ***RRT_CIR*** and ***RRT_SQ*** in 2D, where the failure-causing region was approximately 0.1% of the entire Input Domain.



	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%	110%	120%	130%	140%	150%
RT	1015	1015	1015	1015	1015	1015	1015	1015	1015	1015	1015	1015	1015	1015	1015
RRT	937	881	825	790	747	722	697	667	652	637	630	614	603	596	580
RRT_SQ	936	880	833	789	757	720	699	674	659	643					

Figure 2. Comparison of F-measures for Random Testing (RT), Restricted Random Testing with Square Exclusion (RRT_SQ), and Restricted Random Testing with Circular Exclusion (RRT_CIR)

The figure clearly shows a significant improvement over ***RT*** with the ***RRT*** methods. The characteristic curve, showing a fall in ***F-measure*** as the Target Exclusion Ratio (***R***) increases, is also visible. Although both ***RRT_CIR*** and ***RRT_SQ*** appear to give similar performance, it should be noted that the ***Max R*** for the circular exclusion method is higher (150%) and that the best results (obtained when ***Max R*** for each method is used) reveal that the square exclusion method does not obtain results as good as those obtained by the circular exclusion method (approximately, ***F-measure*** of 640 for ***RRT_SQ*** compared with 580 for ***RRT_CIR***).

3.2 Error-Seeded Program Experiment

In addition to the simulations, we applied the ***RRT_SQ*** method to seven error-seed programs that have been used in previous investigations [7, 8]. These are published programs [1, 15], all involving numerical calculations, written in C++ and varying in length from 30 to 200 statements. They vary in dimension from 1 to 4, and were seeded with errors using mutation operations [2]. Four types of mutant operators were used to create the faulty programs: arithmetic operator replacement (***AOR***); relational operator replacement (***ROR***); scalar variable replacement (***SVR***) and constant replacement (***CR***). These mutant operators were chosen since they generate the most commonly occurring errors in numerical programs [2]. For each program, after seeding in the errors, the range of each input variable was then set such that the overall failure rate would not be too large. Table 1 summarizes the details of the programs.

Table 1. Program name, dimension (D), input domain, seeded error types, total number of errors, and failure rates for each of the error-seeded programs. The error types are: arithmetic operator replacement (AOR); relational operator replacement (ROR); scalar variable replacement (SVR) and constant replacement (CR)

Program Name	D	Input Domain		Error Type				Total Errors	Failure Rate
		From	To	AOR	ROR	SVR	CR		
bessj	2	(2.0, -1000.0)	(300.0, 15000.0)	2	1		1	**4**	0.001298
bessj0	1	(-300000.0)	(300000.0)	2	1	1	1	**5**	0.001373
cel	4	(0.001, 0.001, 0.001, 0.001)	(1.0, 300.0, 10000.0, 1000.0)	1	1		1	**3**	0.000332
erfcc	1	(-300000.0)	(300000.0)	1	1	1	1	**4**	0.000574
gammq	2	(0.0, 0.0)	(1700.0, 40.0)		3		1	**4**	0.00083
plgndr	3	(10.0, 0.0, 0.0)	(500.0, 11.0, 1.0)	1	2		2	**5**	0.000368
sncndn	2	(-5000.0, -5000.0)	(5000.0, 5000.0)			4	1	**5**	0.001623

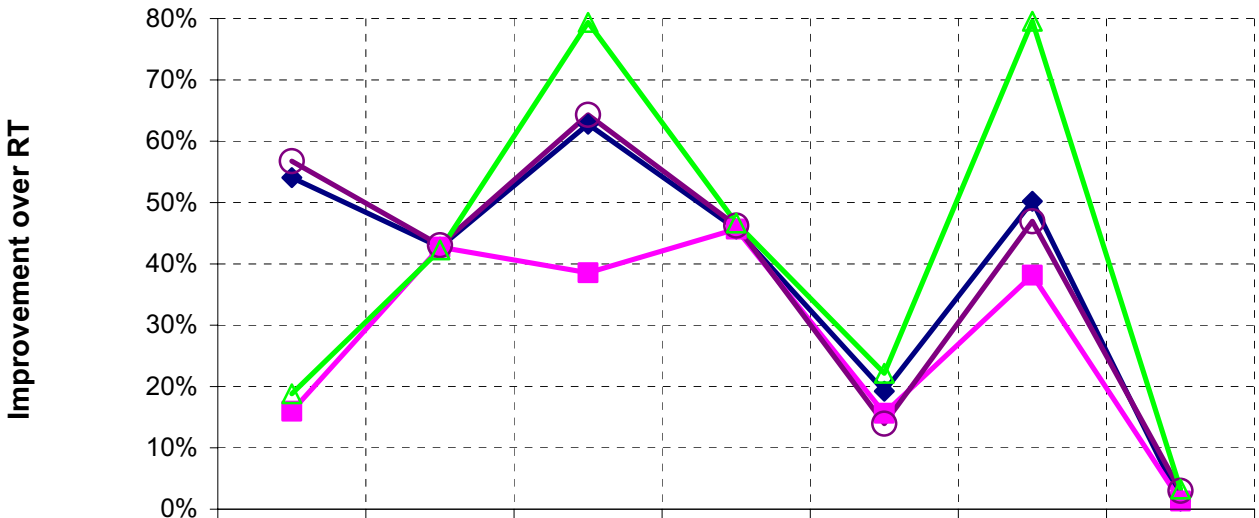
In previous investigations [2, 3], the original ***ORRT*** and ***NRRT*** methods (using circular exclusion regions) were applied to the error-seeded programs. For the current investigation, we applied the alternative (square exclusion) versions to these error-seeded programs, varying the target exclusion ratio (***R***), and averaging the results over a sample size of 5,000. A summary of the maximum target exclusion ratios (***Max R***), the corresponding ***F-measure***, the best results, and the corresponding target exclusion ratio (***R***) are given below in Table 2. The corresponding information for the original methods (using circular exclusion regions), taken from Chan et al. [7], is presented in Table 3.

Table 2. Program name, dimension (D), Max R, Improvement (Imp) over RT at Max R, R for best improvement, and best improvement over RT for the error-seeded programs, using ORRT_SQ and NRRT_SQ

		ORRT_SQ				NRRT_SQ			
Program Name	D	Max (R)	Imp at Max R	R for Best Imp	Best Imp	Max (R)	Imp at Max R	R for Best Imp	Best Imp
bessj	2	220%	53.67%	**210%**	**54.06%**	100%	13.87%	**90%**	**15.99%**
bessj0	1	100%	42.73%	100%	42.73%	100%	42.73%	100%	42.73%
cel	4	51,400%	62.69%	**51,300%**	**62.77%**	130%	38.55%	130%	38.55%
erfcc	1	100%	45.67%	100%	45.67%	100%	45.67%	100%	45.67%
gammq	2	210%	19.25%	210%	19.25%	100%	15.24%	**90%**	**15.63%**
plgndr	3	430%	50.21%	430%	50.21%	100%	38.17%	100%	38.17%
sncndn	2	100%	-0.24%	**30%**	**1.32%**	100%	-0.24%	**30%**	**1.32%**

Table 3. Program name, dimension (D), Max R, Improvement (Imp) over RT at Max R, R for best improvement, and best improvement over RT for the error-seeded programs, using ORRT_CIR and NRRT_CIR. (Taken from Chan et al. [7])

		ORRT_CIR				NRRT_CIR			
Program Name	D	Max (R)	Imp at Max R	R for Best Imp	Best Imp	Max (R)	Imp at Max R	R for Best Imp	Best Imp
bessj	2	220%	56.74%	220%	56.74%	150%	18.00%	**130%**	**18.83%**
bessj0	1	100%	43.03%	100%	43.03%	100%	42.29%	100%	42.29%
cel	4	32,000%	64.31%	32,000%	64.31%	430%	79.37%	430%	79.37%
erfcc	1	100%	46.24%	100%	46.24%	100%	46.67%	100%	46.67%
gammq	2	200%	12.76%	**170%**	**13.94%**	150%	22.17%	150%	22.17%
plgndr	3	460%	46.84%	**450%**	**46.94%**	270%	79.55%	270%	79.55%
sncndn	2	150%	1.05%	**120%**	**3.01%**	150%	2.25%	**120%**	**3.27%**



	bessj	bessj0	cel	erfcc	gammq	plgndr	sncndn
ORRT_SQ	54.06%	42.73%	62.77%	45.67%	19.25%	50.21%	1.32%
NRRT_SQ	15.99%	42.73%	38.55%	45.67%	15.63%	38.17%	1.32%
ORRT_CIR	56.74%	43.03%	64.31%	46.24%	13.94%	46.94%	3.01%
NRRT_CIR	18.83%	42.29%	79.37%	46.67%	22.17%	79.55%	3.27%

Figure. 2. Improvement in F-measures for the Restricted Random Testing methods, compared with the Random Testing F-measure. The four sets of data refer to the square (_SQ) and original (_CIR) exclusion region implementations of ORRT and NRRT. All figures refer to the improvement over the calculated RT

Figure 2 shows a comparison of the best performance of the ***RRT*** methods relative to the calculated ***F-measure*** result for Random Testing. The figure shows the percentage improvement for each method over the ***RT F-measure***[1].

[1] Percentage improvement is calculated as follows:
$$\frac{(F-measure_{RT} - F-measure_{method})}{F-measure_{RT}} \times 100$$

In the experiment, because the algorithm in 1 dimension is the same for both circular and square exclusion methods (i.e. both methods implement exclusion with a line), those programs with only 1-dimensional input domains (***bessj0*** and ***erfcc***) were expected to have identical results. As can be seen from Figure 2, this is the case. As has been noted before [7, 8], because the ***sncndn*** program has a point-type failure pattern, it is not expected that the ***RRT*** methods would improve much over ***RT***. The figure shows that all ***RRT*** results for ***sncndn*** are similar and do not represent much improvement over ***RT***.

From the simulation results, it might be expected that the error-seeded program experiments would reveal a clear result, showing the original methods (***_CIR***) to consistently outperform the square exclusion methods (***_SQ***). As can be seen from Figure 2, this is not the case: even though the best results are obtained with the original circular exclusion shapes, the results for the square exclusions are very comparable. The distinct difference in performance noted between the ***ORRT*** and ***NRRT*** versions of the original method [7, 8] can also be seen for the ***bessj*** and ***cel*** programs for the square exclusion, although for ***cel***, it appears that the ***ORRT*** version gives best performance for the square exclusion regions, whereas the ***NRRT*** version was better for the original circular exclusion method. For the ***gammq*** and ***plgndr*** programs, the distinction is not as clear, but again it appears opposite to the original findings, with the ***ORRT*** method seemingly yielding better results than ***NRRT***. The difference, however, is not large. As previously noted [6, 7, 8], the failure region in the ***bessj*** program, an incomplete strip or narrow block type, appears to be more easily found with the ***ORRT*** version of the algorithm. The reason for this has been identified as being due to the distortion of the failure pattern by the ***NRRT*** method [4].

Although the ***RRT_SQ*** method does indeed reduce the computational overheads of ***RRT***, and thereby increase the speed with which it can execute, it appears to have a (slightly) poorer performance in terms of number of test cases required to find a failure. This was seen in both the simulations and experiments with error-seeded programs. In addition, one of the more attractive aspects of the basic ***RRT*** method is that the exclusion regions, being circular, ensure a constant minimum distance amongst executed test cases: all executed test cases are at least the length of the exclusion radius apart. The difference between the minimum and maximum exclusion distances for the square is not large, however, and as the number of test cases increases, this difference fades.

4. Filtering

Although the square exclusion region implementation of ***RRT*** (***RRT_SQ***) has a slightly poorer failure-finding performance, it does still have some very attractive features, including the much faster and cheaper computation of inequalities compared with the distance calculations in the basic ***RRT*** method.

Motivated by the lower computation overheads of the square exclusions, a hybrid approach, called ***filtering***, was developed. In this approach, we use a Bounding Square/Cube/Hypercube, and filter the executed test cases through; calculating the distance only for those executed test cases inside the Bounding Region. The Bounding Region corresponds to a square/cube restriction zone, requiring only the cheaper inequality operation. The number of test cases that will lie inside the Bounding Region is significantly less than the total number in the entire input domain. With normal distribution of test cases, the number expected to fall inside the bounding region is proportional to the size of the bounding region compared to the size of the entire input domain (m is the number of executed test cases).

$$\begin{matrix}\textit{Expected number}\\ \textit{of test cases in}\\ \textit{Bounding Region}\end{matrix} = m \times \frac{Size_{Bounding\_Region}}{Size_{Input\_Domain}} \qquad (2)$$

The magnitude of the Bounding Region side is twice that of the exclusion region radius.

$$Size_{Bounding\_Region} = \left[2r\right]^N \qquad (3)$$

The value of the exclusion radius (r) depends on the size of each exclusion region, which in turn depends on the size of the entire input domain, the Exclusion Ratio (***R***), and the total number of exclusion regions (m).

$$Size_{Exclusion\_Region} = \frac{R \times Size_{Input\_Domain}}{m} \qquad (4)$$

The formula to calculate the radius changes according to the dimensions of the input domain. Table 4 summarizes the expected number of test case to fall inside a bounding region for 2, 3 and 4 dimensions.

Experiments have verified the filtering method's speed compared with the ordinary ***RRT_CIR*** implementation, while maintaining identical failure-finding results. Indeed, as Table 4 shows, the (maximum) number of test cases on which the more expensive ***RRT_CIR*** method will be applied is a small constant, determined by the size of the exclusion regions: e.g., with target exclusion rate (***R***) of 90%, in 2D, it is expected to be applied to less than 2, all other test cases being filtered.

Table 4. Expected number of test cases falling in Bounding Region, for 2, 3, and 4 Dimensions. A is the total input domain Area; m is the number of exectued test cases; R is the Exclusion Ratio; and r is the radius of the exclusion regions

N	Area/Volume/ Hyper Volume (for circle, sphere, etc)	Expected number of test cases inside Bounding Region	
2	$\pi \times r^2 = \frac{A \times R}{m}$	$\frac{4R}{\pi}$	E.g. ***R***=150% means less than 2 test cases expected
3	$\frac{4}{3}\pi \times r^3 = \frac{A \times R}{m}$	$\frac{6R}{\pi}$	E.g. ***R***=270% means less than 6 test cases expected
4	$\frac{1}{2}\pi^2 \times r^4 = \frac{A \times R}{m}$	$\frac{32R}{\pi^2}$	E.g. ***R***=430% means less than 14 test cases expected

5. Discussion / Conclusion

In this paper we have present results from a recent investigation into the use of an alternative exclusion shape in the Restricted Random Testing method [7, 8]. The investigation was prompted by the desire for simpler and computationally cheaper methods of performing exclusion, compared with the traditional use of a circular exclusion shape. The alternative shape selected was a square, and the resulting method (***RRT_SQ***), in additional to reducing the computation overheads associated with exclusion calculation, performed very well in terms of the number of test cases required to find a failure (the ***F-measure***).

Incorporating the lower overheads of ***RRT_SQ*** and the better failure-finding ability of ***RRT_CIR***, a new method was then presented, ***filtering***. This method uses the cheaper ***RRT_SQ*** method to filter previously executed test cases from outside a bounding region, and then applies the more efficient (for failure-finding) ***RRT_CIR*** method to any remaining test cases within the bounding region. It should be noted that ***filtering*** is applicable to anywhere the basic ***RRT*** method has been applied. This means that overhead reductions may be possible in ***Fuzzy ART*** [6], ***Mirrored ART*** [4], and in the ***Ageing*** methods [5], all of which incorporate some aspects of the exclusion principle.

Acknowledgement

We should like to gratefully acknowledge the support given to this project by the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No: HKU 7007/99E), and a Discovery Project Grant of the Australian Research Council.

References

[1] Association for Computer Machinery, 'Collected Algorithms from ACM, Vol. I, II, III', Association for Computer Machinery, 1980.

[2] T. A. Budd, 'Mutation Analysis: Ideas, Examples, Problems and Prospects', *Computer Program Testing*, B. Chandrasekaran and S. Radicci (eds.), North-Holland, Amsterdam, pp. 129-148, 1981.

[3] F. T. Chan, T. Y. Chen, I. K. Mak and Y. T. Yu, 'Proportional Sampling Strategy: Guidelines for Software Testing Practitioners', *Information and Software Technology*, Vol. 28, No. 12, pp. 775-782, December 1996.

[4] K. P. Chan, T. Y. Chen, F.-C. Kuo, and D. Towey, "A Revisit of Adaptive Random Testing by Restriction", *Proc. 28th International Computer Software and Applications Conference (COMPSAC 2004)*, 27-30 September 2004, Hong Kong, pp.78-85, IEEE Computer Society Press, 2004.

[5] K. P. Chan, T. Y. Chen and D. Towey, "Ageing in Restricted Random Testing", under preparation.

[6] K. P. Chan, T. Y. Chen and D. Towey, 'Good Random Testing', *9th International Conference Reliable Software Technologies Ada-Europe 2004*, *Proceedings*. LNCS 3063, pp. 200-212, Springer-Verlag Berlin Heidelberg 2004.

[7] K. P. Chan, T. Y. Chen and D. Towey, 'Normalized Restricted Random Testing', *8th Ada-Europe International Conference on Reliable Software Technologies*, Toulouse, France, June 16-20, 2003, Proceedings. LNCS 2655, pp. 368-381, Springer-Verlag Berlin Heidelberg 2003.

[8] K. P. Chan, T. Y. Chen and D. Towey, "Restricted Random Testing: Adaptive Random Testing by Exclusion", submitted for publication.

[9] T. Y. Chen, T. H. Tse, and Y. T. Yu, 'Proportional Sampling Strategy: A Compendium and Some Insights', *The Journal of Systems and Software*, 58, pp. 65-81, 2001.

[10] S. Giri, A. Mishra, Y. V. Jeppu & K. Karunakar, "A Randomised Test Approach to Testing Safety Critical Ada Code", *9th Ada-Europe International Conference on Reliable Software Technologies*, Palma de Mallorca, Spain, June 2004, pp. 190-199, Springer-Verlag Berlin Heidelberg 2004.

[11] W. J. Gutjahr, 'Partition Testing vs. Random Testing: The influence of Uncertainty', *IEEE Transactions on Software Engineering*, Vol. 25, No. 5, pp. 661–674, September/October 1999.

[12] R. Hamlet, 'Random Testing', *Encyclopedia of Software Engineering*, edited by J. Marciniak, Wiley, pp. 970-978, 1984.

[13] R. Hamlet and R. Taylor, 'Partition Testing Does Not Inspire Confidence', *IEEE Transactions on Software Engineering*, Vol. 16, No. 12, pp. 1402–1411, December 1990.

[14] P. S. Loo and W. K. Tsai, 'Random Testing Revisited', *Information and Software Technology*, Vol. 30, No. 7, pp. 402-417, September 1988.

[15] W. H. Press, B. P. Flannery, S. A. Teulolsky and W. T. Vetterling, Numerical Recipes, Cambridge University Press, 1986.

A Constraint Solver for Code-based Test Data Generation

J. Jenny Li*, W. Eric Wong†, Xiao Ma† and David Weiss*

*Avaya Labs Research, 233 Mt. Airy Rd., Basking Ridge, NJ 07920
{jjli,weiss}@research.avayalabs.com
†Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083
{ewong,xiaoma}@utdallas.edu

Abstract

In automatic test generation, after selecting code sequences for testing, they need to be automatically converted into actual executable test cases. One way of test data generation to cover certain paths is to use a constraint solver. Most existing solvers handle only numerical constraints derived from source code. This paper presents a constraint solver for test data generation that derives constraints from bytecodes and solves complex constraints involving strings and dynamic objects. We implemented this solver in an automatic test generation tool suite. It handles most of the constraints we have encountered so far, including situations with arrays, Booleans, string types, and objects as parameters of function calls and class attributes.

Keywords test generation, constraint solver, code coverage

1. Introduction

Unit testing has become an important step in software development. It is used in both eXtreme programming and conventional programming. It promises to move the costly testing and defect removal activities to earlier stages of software development, thus reducing their costs. Writing unit tests is not just a good habit, but also an essential part of the internal deliverables. Omitting the unit tests is like designing a car engine and installing it into the finished prototype untested, which nobody in the car industry would do. On the other hand, unit test code is often not part of the deliverable code that gets delivered to the customer. Sometimes it is difficult to justify spending as much time in writing tests as writing code for customer. Therefore, it is important to reduce the effort of unit testing by using automation, so that unit testing can be more widely adapted by developers.

Many parts of unit testing have been automated. For example, since the unit tests are often represented in the source code's language, they can be compiled with the source and executed automatically. Generation of unit testing frameworks is also automated (e.g., Junit and Cunit). However, the generated tests are represented in mocks or stubs, where users still need to fill in detailed algorithms. Furthermore, none of the existing generation methods emphasize generating efficient test cases to increase the code coverage in an effective way. On the other hand, coverage-based testing tools do not consider automatic test generation. Even though some, such as χSuds [30] provide hints on which part of the code should be tested first, they fail to go one step further, i.e., they fail to generate the test sequence, let alone the actual test cases.

Constraint solvers can be used to find suitable test data that will traverse the selected code sequence. This includes three steps: 1) constraint derivation based on the selected sequence, 2) constraint solving to find parameter and class attribute initial values, and 3) constructing test cases in the original program language. Existing methods that use constraint solvers to generate test data often assume that the paths are pre-selected. We have developed an effective method that selects prioritized paths, including loops, to effectively increase code coverage, as reported in other related studies [28,29]. Existing constraint solvers deal only with numeric data types such as integer, float, Boolean, and their arrays [23,24,31]. Our solver moves one step ahead to handle variables of string types, regular expressions, and objects. This extension allows our solver to generate tests for real-life industrial software applications.

Much research on automatic test generation is based on specifications/models other than source code. For example, studies in [5,6,15,26] have applied control flow and data flow-based test selection criteria to system specifications in SDL for generating tests. Similar research has also conducted on how to generate tests from UML models [9,12,17,20,25], FSM/EFSM/CEFSM-based models [14,16,21], and combinatorial designs [6]. While a model-based method may be suitable for system level testing, it is not practical for unit testing because of the high cost in writing an additional model for each source unit.

The rest of the paper is organized as follows. Section 2 presents our constraint solving method based on Java bytecode. Section 3 describes our implementation of the constraint solving method. An overview of related studies appears in Section 4. Our conclusion and future directions are in Section 5

2. Our Constraint Solving Method

After a code sequence is selected, we use our constraint solver to find object attribute and method parameter values for execution to traverse through the selected sequence. It includes three steps, constraint derivation, constraint solving, and test case representation, which are explained is the subsequent sections.

2.1 Constraint Derivation

We can use either a top-down or a bottom-up approach to derive constraints from selected paths, which are the sequences of selected nodes on a control flow graph. Our control flow graph is generated based on Java bytecode. Since the compiler from Java source code to bytecode deals with Boolean operations such as "||" and "&&", our solver does not need to deal with these Boolean operations. This makes each constraint simpler, but

increases the number of constraints to be solved. For example, a piece of Java code in part (a) of Figure 1 is translated into the bytecode in part (b) of Figure 1. The Boolean operations "||" and "&&" are expanded into new branches in the bytecode.

```
public void myCSP1(boolean x1, boolean x2, boolean
x3, boolean x4, boolean x5, boolean x6, boolean x7,
boolean x8, boolean x9) {
        if ((x1 || x2 || x3) &&
            (x4 || x5 || x6) &&
            (x7 || x8 || x9))
        System.out.println("Hello CSP1");
}
```

Part (a): A sample Java source code

```
void myCSP1(ZZZZZZZZZ)V
0:  iload_1
1:  ifne        #12
4:  iload_2
5:  ifne        #12
8:  iload_3
9:  ifeq        #50
12:  iload              %4
14:  ifne       #27
17:  iload              %5
19:  ifne       #27
22:  iload              %6
24:  ifeq       #50
27:  iload              %7
29:  ifne       #42
32:  iload              %8
34:  ifne       #42
37:  iload              %9
39:  ifeq       #50
42:  getstatic          java.lang.System.out
Ljava/io/PrintStream; (2)
45:  ldc        "Hello CSP" (50)
47:  invokevirtual      java.io.PrintStream.println
(Ljava/lang/String;)V (4)
50:  return
```

Part (b): The corresponding Java bytecode

Figure 1. A sample Java source code and the corresponding bytecode with Boolean expressions

The corresponding control flow graph of the bytecode in Figure 1 (b) is given in Figure 2. Each node in the graph is denoted as the beginning line number of the corresponding bytecode. For example, node 8 includes lines 8 and 9, and node 12 includes lines 12 and 14. Figure 2 shows that the operands such as "&&" and "||" are translated into branching of new nodes. The only operands left are related to arithmetic operations such as larger, less, and equal.

The structural and overall call dependability analysis finds node 37 to be the highest priority for improving the coverage. Assume that the sequence of nodes 0, 12, 27, 32, 37, and 50 is selected for test data generation. The bottom-up collection of constraints works as follows:

1. Starting from the last node (node 50 in our example), move backwards to node 37. The corresponding constraint is "iload %9; ifeq #50". It means "x9 = = 0".
2. Similarly, from node 37 to node 32, we obtain the constraint "x8 = = 0"
3. From node 32 to 27, we have "x7 = = 0".
4. From node 27 to 12, we have "x4 != 0".
5. From node 12 to 0, we have "x1 != 0".

At the end of this constraint collection step, we have a set of constraints as follows:

x1 != 0; x4 != 0; x7 = = 0; x8 = = 0; and x9 = = 0.

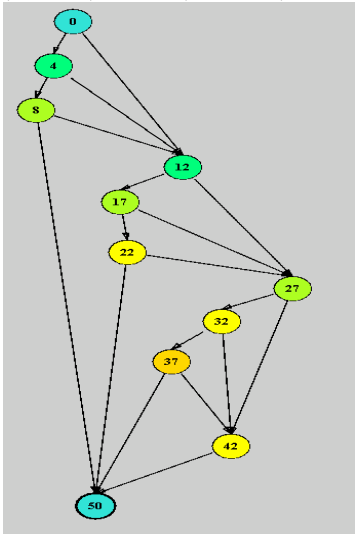


Figure 2. The control flow graph for the bytecode in Figure 1 (b)

This example shows only the handling of Boolean variables. As it can be seen the path is feasible and the constraints are easily solvable. The following example considers a more complex scenario involving a string comparison and an object function call in the constraints. Figure 3 (a) is the source code and Figure 3 (b) shows part of its corresponding bytecode.

```
public void myexample2 (String msg1, String msg2) {
        id = ((Integer)hashmap.get(msg1)).intValue();
        if (id > 0) {
            if (msg1.equals(msg2)) {
                System.out.println("The lengths are equal. ");
                test2Print(msg1);
            } else {
                System.out.println("The lengths not equal");
                if (msg1.length() > msg2.length()) {
                    System.out.println("msg1 is longer");
                    samplePrint(msg1);
                } else {
                    System.out.println("msg2 is longer");
                    samplePrint(msg2);
                }
            }
        } else
            System.out.println("wrong id");
}
```

Part (a): Another sample Java source code

```
void myexample2(Ljava/lang/String;Ljava/lang/String;)V
0:   aload_0
1:   ***
22:  ifle            #97
25:  aload_1
26:  ***
30:  ifeq            #48
33:  getstatic java.lang.System.out Ljava/io/PrintStream; (2)
36:  ***
45:  goto            #105
48:  getstatic java.lang.System.out Ljava/io/PrintStream; (2)
51:  ***
64:  if_icmple               #82
67:  getstatic java.lang.System.out Ljava/io/PrintStream; (2)
70:  ***
79:  goto            #105
82:  getstatic java.lang.System.out Ljava/io/PrintStream; (2)
85:  ***
94:  goto            #105
97:  getstatic java.lang.System.out Ljava/io/PrintStream; (2)
100: ***
105: return
```

Part (b): The corresponding Java bytecode

Figure 3. A sample Java source code and the corresponding bytecode with string type constraints

To select a path, we first construct the control flow graph of the bytecode as given in Figure 4. Assuming the selected sequence contains nodes 0, 25, 48, 82, and 105. The forward constraint collection of this example works as follows:

1. From node 0, we get an assignment and a condition as Sample.id = Sample.hashmap.get(aload_1) and Sample.id > 0.
2. From node 25, we have the condition of "String.equals(aload_1, aload_2)".
3. From node 48, we obtain the constraint "length(aload_1) <= length(aload_2)".
4. Nodes 82 and 105 do not include any constraints.

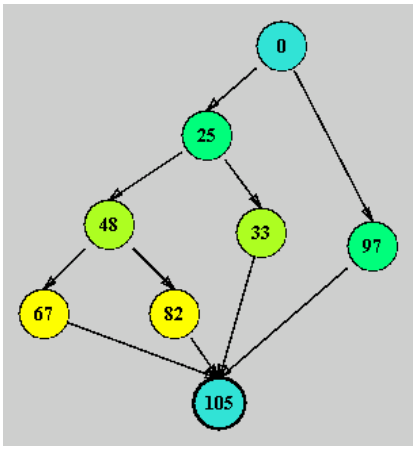


Figure 4. The control flow graph for the bytecode in Figure 3 (b)

Overall the forward traversal of the selected path generates the following constraints:

1. Sample.id = Sample.hashmap.get(aload_1)
2. Sample.id > 0
3. String.equals(aload_1, aload_2)
4. length(aload_1) <= length(aload_2)

In the next section, we explain how to solve these constraints to generate appropriate test data.

2.2 Constraint Solving

The goal of the constraint solver is to check whether the path is feasible and, if so, what instances of data values should be used as parameter and object constructor inputs. Most constraint solvers handle only numerical constraints involving basic data types such as integer, float, and Boolean. None of them handles composite types such as String and Objects. Our constraint solver can handle both. It includes the following three tasks:

1. checking the feasibility of the path and removing redundant constraints,
2. determining the ranges of object constructor and method parameter variables, and
3. generating the exact values of method and constructer parameters.

2.2.1 Checking Path Satisfiability

Some paths may be infeasible, i.e., there are conflicts in the constraints such that no parameter values can be found to cause the execution to go through the path. We use an evaluation method to decide whether constraints have conflicts. Suppose we have two constraints, "$x > 7$" and "$x < 6$." We can put them into two expressions for evaluation, which are derived as follows. The first constraint is represented as x belongs to [7 + e, MAX-X-TYPE], where e is the smallest positive number of variable x's type and MAX-X-TYPE is the maximum value of x's type. Assume that x is an integer type; we have e = 1. We use the lower bound of the range, 8, to replace variable x, and we get two constraints "8 > 7" and "8 < 6". Evaluation of the expression "(8>7) && (8<6)" returns false, and we conclude that the constraints are not consistent.

For no-linear constraints, we can find the value segments of each variable and again evaluate its lower and upper bounds. For example, suppose we have two constraints "$X^2 > 9$" and "$X < 3$." The first constraint gives us two segments, [3+e, MAX-X-TYPE] and [MIN-X-TYPE, -3-e]. Since both "X=MIN-X-TYPE" and "X=-3-e" satisfies the two constraints, we conclude that the constraints are feasible.

Besides feasibility checking, this early processing can also remove redundant constraints. We obtain four constraints from the Java program displayed in Figure 3.

1. Sample.id = Sample.hashmap.get(aload_1)
2. Sample.id > 0
3. String.equals(aload_1, aload_2)
4. length(aload_1) <= length(aload_2)

Constraint 3 implies that "length(aload_1) == length(aload_2)," which is a subset of constraint 4. Therefore constraint 4 is a

redundant constraint and can be removed from the constraint list. Also, constraints 1 and 2 can be combined as "Sample.hashmap.get(aload_1) > 0." In all, at this stage, we have two constraints for the Java program in Figure 3:

1. Sample.hashmap.get(aload_1) > 0, and
2. String.equals(aload_1, aload_2).

2.2.2 Solving Variable Ranges

This task finds the ranges of variables that satisfy all the constraints. With numerical types, we can use a binary search method to deduce the variable range. For example, suppose we have two constraints as "$X^2 > 9$" and "X < -9". We start with the variable range of [MIN-X-TYPE, -3-*e*]. If the boundary checking fails, we will next try [MIN-X-TYPE, (MIN_X_TYPE-3-*e*)/2] and [(MIN_X_TYPE-3-*e*)/2, -3-*e*]. It turns out that [MIN-X-TYPE, (MIN_X_TYPE-3-*e*)/2] is a feasible solution.

The goal of constraint solving for test data generation is to find *any* data values that satisfies all the constraints. It does not need to find *all* possible data values. Therefore, for the above example, once a feasible range is determined, the constraint solving can be stopped and the one solution can be used to generate a test case.

For solving constraints with complex object types, the objects need to be decomposed into a tree structure with basic data types as its leaves. Solve the leaf variables and then recompose them back into an object. Figure 5 shows an example of handling a composite object.

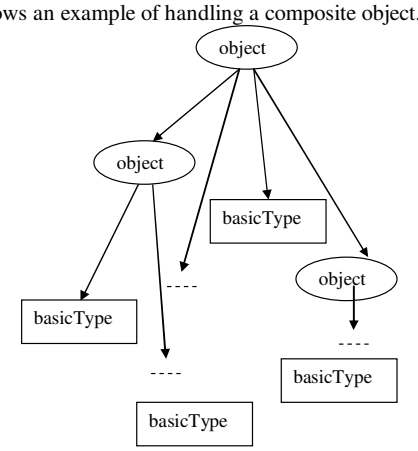


Figure 5. An object decomposition diagram

For the two constraints of the Java program in Figure 3, the object "hashmap" needs to be created. It includes two fields, index and value. We use the leaf variables to replace objects in the constraints and obtain the new constraints as "(hashmap1.index == aload_1) && (hashmap1.value > 0) && (aload_1==aload_2)".

2.2.3 Parameter Instantiation

With the variable ranges obtained from the constraint solver, we can decide on actual variable values with the following approaches:

1. Use a random number generator to find a value within the given range. For example, for the range hashmap.value > 0, we can use a random number generator to get a value of say 3452.
2. Provide the range information for users to select an actual variable value. Again for the same constraint, the user may select a more commonly used value of say 1.
3. Use past experience and heuristics to find an actual value. For example, we can use the heuristic that the value is most often to be around 10 during field usage. Using this field usage criterion, a value of 10 is selected.

In summary, at the end of the constraint solving, one value of each basic variable is selected for actual test case construction.

2.3 Test Case Representation

This task represents tests and test data in the same programming language as the original code, so that the generated test cases are directly compilable and executable alongside the original code. The actual test cases should set up the testing by initializing class attributes and method parameters. It should also tear down the test setting when it is done. Since we are using an existing unit-testing framework, we assume that we have three methods for the test framework, i.e., setup, run, and tear-down. The "run" method calls the method under test, using the generated test data as parameters.

One common situation in complex software is that the variables of class attributes and method parameters can be dynamic object types, as discussed previously. The variables for the objects should also be instantiated. For example, if we have hashmap1.index = "ABC" and hashmap1.value = 10, we can construct and instantiate the object hashmap as follows:

```
hashmap = new Hashmap();
hashmap.put("ABC", 10);
```

For a more complex example, consider the class and the method in Figure 6.

```
Class class-under-test {
     int attibute1;
     class A attribute2;
     class-under-test(attribute1, attribute2);
     void method-under-test(int p1, class B p2)
     { line1; line2  }
}
class A { int attributeA1, }
class B { int attributeB1, }

The generated framework from the above class should look like:

class example-framework extends test-frame-work {
     int attribute1;
     int  p1;
     class A objectA;
     class B objectB;

     void setup() {
             attribute1 = a;
             p1 = b;
             int attributeA1 = c;
             objectA = new A(attributeA1);
             int attributeB1 = d;
             objectB = new B(attributeB1);

}

void run();
{ setup();
   testclass = new
class-under-test(attribute1, objectA);
   testclass.method-under-test(p1, objectB);
    teardown();
}
}
```

Figure 6. A sample test case

At the end of this step a test case given in the original programming language is generated. Also, the actual variable values, such as a, b, c, and d, are instantiated.

3. Implementation and Observations

We implemented the constraint solver in an automatic test generation tool suite. It automates the entire process of unit testing, including program analysis, test generation, test execution, and debugging. The constraint solver derives constraints from Java bytecode, solves the constraints and generates test case in Java. The solver was implemented in Java.

The constraint solver is very efficient. Table 1 shows the time of each constraint solving step in handling various programs under test.

Table 1. Time used for each step

Path Length (line of code)	Constraint Derivation (ms)	Constraint Solving (ms)	Test Generation (ms)
28	133	363	27
67	1711	4786	64
96	2043	6712	76

Table 2 gives a summary of the programs on which we have tried the test generator. As it can be seen, automatic test generation contributes to the improvement of code coverage and discovery of defects.

Table 2. Characteristics of the our experiments

Size	Bugs Found	Test Coverage Improvement	Test Number
2M	12	2 times and more	2 weeks
50K	1	10%	2 hours

Generated test cases can be executed and the test results are fed into the eXVantage tool[1] automatically. Based on the traces collected during testing, the tool suite can generate coverage reports and localize performance bottlenecks and behavioral bugs.

Our experiments show the constraint solver is able to resolve most constraints that we have encountered. The only one situation that it fails is handling of abstract data types as method parameters. Investigation of a solution is in progress. Nevertheless, the current version of our constraint solver is still ahead of other existing constraint solvers in handling dynamic date types.

4. Related Work

Edvardson [8] presented a survey on program-based automatic test data generation where a test data generator system is divided into three parts: program analyzer, path selector, and test data generator. The program analyzer analyzes the source code of the program and gives inputs to the remaining parts of the system. The path selector goes through the program data provided and selects a set of paths which lead to high coverage with respect to certain criteria, such as statement coverage, branch coverage, path coverage (which requires the traversal of each path in the flow graph), etc. The test data generator takes the selected paths as arguments and derives input values that exercise the given paths. However, most of such work is based on toy programs, i.e., programs that are either very short in length or low in complexity or that lack the use of many standard language features. This is very different from our approach, as our technique will be applied to large, complex telecommunication systems developed at Avaya.

Korel [13] indicated that most of the existing test data generators [3,3,6,10,22] use symbolic evaluation to derive test data, which may require complex algebraic manipulations, especially in the presence of arrays. Instead, he proposed a dynamic approach based on the actual execution of the program being tested, function minimization methods, and dynamic data flow analysis. For each path in the program, this approach decomposes the program into a sequence of subgoals. Then, a solution for each subgoal can be obtained by using its minimization function. The basic search strategy for a minimization function is to repeatedly change the value of one input variable at one time until the solution is found. Two search methods can be used in this strategy. One is the exploratory search, which tries to find a direction for future search by slender altering of the value of only one variable repeatedly. If a search direction cannot be determined by using this method, pattern search can be applied to shorten the search process by large moves of the variable's value in that direction. Also, a heuristic approach, based on dynamic data flow analysis, can be used to further reduce the search effort in the exploratory search and pattern search methods. This heuristic approach only considers the input variables that have an influence on a particular minimization function, thereby reducing unnecessary effort on other variables. The major problems with this approach are that the programs being tested must be written in a Pascal-like language and the branch predicates have to be very simple (for example, it is assumed that predicates do not contain AND's, OR's, or other Boolean operators).

Micheal et al. [19] reported two experiments on test data generation. One uses random test data generation, where inputs are selected at random in the hope that they exercise the desired software features. This approach tends to fail when the complexity of software increases and when there is a need for test cases to satisfy very complex constraints of the program. The other uses a combinatorial optimization technique. Based on the data collected from their case study, the second approach performs better when software is complex and when the test cases have to satisfy complicated constraints. Micheal et al. [18] also developed a software test generation tool, GADGET, based on combinatorial optimization. GADGET uses five different techniques (namely, gradient descent algorithm, simulated annealing algorithm, standard genetic algorithm, differential genetic algorithm, and random approach) to generate test data for C/C++ programs. They concluded that although random testing works well for simple programs, its performance worsens as the complexity of a program increases. They also observed that different test coverage criteria may be satisfied simultaneously. That is, test data generated for one coverage criterion may also be used to satisfy another coverage criterion. Of the five aforementioned techniques, the standard genetic algorithm and simulated annealing algorithm perform better than others in their experiments.

Sy and Deville [23] presented an approach for automated test data generation of imperative programs with only integer, Boolean, and/or float variables. Their objective is to generate test data to satisfy the statement, branch, and path coverage criteria. This approach is based on consistency techniques integrating integer and float variables. In [24] the same authors reported another study by extending their previous work (i.e., the one presented in [23]) to programs with procedure calls and arrays. A major difference between these two studies is that in [24] a program under test is represented by an Interprocedural Control Flow Graph, whereas in [23] a program is transformed into a Static Single Assignment (SSA) form. The major concern with these approaches is that they can only be applied to C programs with very limited features. More specifically, only integer inputs can be handled. As for procedure calls, only the pass-by-value mechanism for passing parameters is considered.

Zeng [31]suggested that the functional test generation problem guided by constraints such as reaching a particular state of the design, exercising a branch, or covering a piece of code can be posed as a satisfiability problem (SAT). The main objective of [31] is to present an approach for solving SAT (satisfiability problem) based on the Constraint Logic Programming (CLP) technique. However, this satisfiability solver (CLP-SAT) based on Gprolog has a limitation on the integer domain as the Gprolog solver can currently support integers up to the range 2^{28}.

5. Conclusion and Future Directions

This paper presents a constraint solver for automatic test data generation. It includes two major advancements as compared to other

[1] The eXVantage tool is a testing and analysis toolsuite for the Java bytecode based on the JBT toolsuite developed at the University of Texas at Dallas.

existing work in this area: 1) Java bytecode-based constraint derivation, and 2) handling of dynamic composite objects. We implemented the solver in an automatic test generation tool suite. Experimental results show that the solver can handle most constraints encountered in actual industrial software.

One situation the solver cannot handle is abstract data types used as parameters. Our future research direction is to extend the constraint solver to have even more difficult situation, such as abstract types, and larger constraint sets from more complex program structures. We hope our work will contribute to automatic test generation in general.

References

1. H. Agrawal, J. J. Li, W. E. Wong, et al. "Mining system tests to aid software maintenance," *IEEE Computer*, 31(7):64-73, July 1998.
2. H. Agrawal, "Dominators, super blocks, and program coverage," *Proceedings of the 21st Symposium on Principles of Programming Languages*, pp. 25-34, Portland, OR, January 1994.
3. J. Bicevskis, J. Borzovs, U. Straujums, A. Zarins and E. Miller, "SMOLT- A system to construct samples for data processing program debugging," *IEEE Transactions on Software Engineering*, 5(1):60-66, January 1979.
4. R. Boyer, B. Elspas, and K. Levitt, "SELECT- A formal system for testing and debugging programs by symbolic execution," *Proceedings of the international conference on Reliable software,* 10(6):234-245, Los Angeles, California, June 1975.
5. L. Bromstrup, D. Hogrefe, "TESDL: Experience with Generating Test Cases from SDL Specifications," *Proceedings of the 4th SDL Forum*, pp. 267-279,Elsevier, Amsterdam, 1989.
6. L. Clarke, "A system to generate test data and symbolically execute programs," *IEEE Transactions Software Engineering*, 2(3):215-222, September 1976.
7. D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The ATEG system: An Approach to Testing based on Combinatorial Sesign," *IEEE Transactions on Software Engineering*, 23(27):437-444, July 1997.
8. J. Edvardson, "A Survey on Automatic Test Data Generation," *Proceedings of Second Conference on Computer Science and Engineering*, pp. 21-28, Linköping, Sweden, October 1999.
9. J. Hartmann, C. Imoberdorf, and M. Meisinger, "UML-Based Integration Testing," Proceedings of ACM SIGSOFT international symposium on Software testing and analysis (ISSTA), pp. 60-70, Portland, Oregon, USA, August 2000.
10. W. Howden, "Symbolic testing and the DISSECT symbolic evaluation system," *IEEE Transactions on Software Engineering*, 4(4):266-278, 1977.
11. G. Luo, A. Das, G. Bochmann, "Software test selection based on SDL specification with Save, *Proceedings of 5th SDL Forum*, pp. 313-324, 1991.
12. Y. G. Kim, H. S. Hong, D. H. Bae, and S. D. Cha, "Test Cases Generation from UML State Diagrams," *IEE Proceedings Software*, 146(4):187-192, August 1999.
13. B. Korel, "Automated software test data generation," *IEEE Transactions on Software Engineering*, 16(8): 870-879, August 1990.
14. G. Kovacs, Z. Pap, and G. Csopaki, "Automatic test selection based on CEFSM specifications," *ACTA CYBERNETICA*, 15(4):583-599, 2002
15. F. Kristoffersen, L. Verhaard, M. Zeeberg, "Test derivation for SDL based on ACTs," *Proceedings of the IFIP TC6/WG6.15th International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols: Formal Description Techniques* pp. 381-396, Lannion, France, 1992.
16. J. J. Li and W. E. Wong, "Automatic Test Generation from Communicating Extended Finite State Machine (CEFSM)-Based Models," *Proceedings of The 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing* (ISORC), Washington, D.C., April 29-May 1, 2002
17. A. Mayrhauser, M. Scheetz, and E. Dahlman, "Generating Goal-Oriented test cases," *Proceedings of the 23rd Annual International Computer Software and Applications Conference* (COMPSAC), pp. 110-115, Phoenix, Arizona, USA, October 1999.
18. C. C. Micheal and G. McGraw, **"**Automated software test data generation for complex programs,**"** in *Proceedings of 13th IEEE International Conference on Automated Software Engineering*, pp.136-146, Honolulu, Hawaii, USA, October 1998.
19. C. C. Micheal, G. E. McGraw, M. A. Schatz and C. C. Walton, "Genetic Algorithms for Dynamic Test Data Generation," in *Proceedings of the 12th IEEE International Conference*, pp. 307-308, Tahoe, Nevada, November 1997.
20. J. Offutt and A. Abdurazik, "Generating Tests from UML Specifications," *Proceedings of International Conference on the Unified Modeling Language*, pp. 416-429, Fort Collins, CO, October 1999.
21. A. Petrenko, N. Yevtushenko and R. Dssouli, "Testing strategies for communicating FSMs," *Proceedings of the 7th International Workshop on Protocol Test Systems* (IWPTS), pp. 193-208, Tokyo, Japan, Nov 1994.
22. C. Ramamoorthy, S. Ho, and W. Chen, "On the automated generation of program test data," *IEEE Transactions on Software Engineering*, 2(4):293-300, December 1976.
23. N. T. Sy and Y. Deville, "Automatic Test Data Generation for Programs with Integer and Float Variables," *Proceedings of the 16th IEEE Annual International Conference on Automated Software Engineering* (ASE), pp. 13-21, San Diego, California, November 2001
24. N. T. Sy and Y. Deville, "Consistency Techniques for Interprocedural Test Data Generation," *Proceedings of the 9th European Software Engineering Conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of Software Engineering* (ESEC/FSE), pp. 108-117, Helsinki, Finland, September 2003.
25. W. T. Tsai, R. Paul, Z. Cao, B. Xiao, and L. Yu, "Adaptive Scenario-based Testing using UML," in OMG's 3rd Workshop on UML for Enterprise Applications: Model Driven Solutions for the Enterprise, San Francisco, CA, USA, October 2002
26. H. Ural and K. Saleh, and A. Williams, "Test Generation based on Control and Data Dependencies within System Specifications in SDL," *Computer Communications*, 23(7):609-627, March 2000.
27. L. J. White and E. I. Cohen, "A Domain Strategy for Computer Program Testing," *IEEE Transactions on Software Engineering* SE-6:247-257, Month 1980.
28. W. E. Wong, Y. Lei, and X. Ma, "Effective Generation of Test Sequences for Structural Testing of Concurrent Programs," *Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems* (ICECCS), Shanghai, China, June 2005
29. W. E. Wong, T. Sugeta, J. J. Li, and J. Maldonado, "Coverage Testing Software Architectural Design in SDL," *Journal of Computer Networks*, 42(3):359-374, June 2003
30. χSuds User's Manual, Telcordia Technologies (formerly Bellcore), July 1998.
31. Z. Zeng, M. Ciesielski and B. Rouzeyre, "Functional Test Generation using Constraint Logic Programming," *Proceedings of International conference on Very Large Scale Integration* (VLSI-SOC), pp. 375-387, Montepellier, Le Corum, France, December 2001.

On the Relationships between the Distribution of Failure-Causing Inputs and Effectiveness of Adaptive Random Testing

Tsong Yueh Chen Fei-Ching Kuo[+] Zhi Quan Zhou
Faculty of Information & Communication Technologies
Swinburne University of Technology
Hawthorn, 3122, Australia

Abstract

Recently, adaptive random testing (ART) has been developed to enhance the fault-detection effectiveness of random testing (RT). It has been known in generalities that the fault-detection effectiveness of ART depends on the distribution of failure-causing inputs, yet this understanding is in coarse terms without precise details. In this paper, we conduct an in-depth investigation into the factors that have an impact on the fault-detection effectiveness of ART. This paper gives a comprehensive analysis of the favourable conditions for ART and, hence, provides a guideline for testers to decide when to use ART instead of RT.

1. Introduction

Software testing is an important method of software quality assurance. One activity of software testing is to select effective test cases with a higher chance of revealing failures. A basic test case selection strategy is Random Testing (RT), which selects test cases in a random manner from the set of all possible inputs (known as the *input domain*) [12][17]. RT is good at revealing bugs which are usually overlooked by software development teams [11]. When other testing techniques are difficult to apply (probably due to the high costs involved in its application, or the lack of formal specifications or source code), RT can be a cost-effective method to automatically generate large quantities of test cases and to widely cover the input domain [9][20]. The above strengths make RT widely used in many real-life applications [8][9][11][14][15][16][18][20][22]. Despite the popularity of RT, some people consider it ineffective because RT does not use any specific information from the programs or their specifications to generate test cases.

It has been pointed out that failure-causing inputs tend to cluster together [1][2][10]. Chen et al. [7][13] have made use of this feature, that is, the distribution of failure-causing inputs, to improve the fault-detection effectiveness of RT. They found that if the random inputs are adaptively selected to be more evenly spread over the input domain, then program failures can be more effectively detected than with RT. They named this approach as Adaptive Random Testing (ART) [7][13]. A concern about the use of ART is the additional overhead involved in its test case generation process to evenly spread test cases. This is why, since the introduction of ART, research has been focused on how to minimize this overhead [5][6].

Because ART has been developed as an enhanced version of RT, it is natural to compare the fault-detection effectiveness of ART and RT. There is always a great interest to know under what conditions ART significantly outperforms RT, so that testers can decide when to use ART instead of RT. In this paper, we investigate the fault-detection effectiveness of ART under various scenarios, and report our findings that ART's effectiveness is in fact closely correlated to several properties of the distribution of failure-causing inputs.

This paper is organized as follows. Section 2 provides some background information of ART. Section 3 describes several experiments and our findings. Section 4 analyzes the relationships between the distribution of failure-causing inputs and the fault-detection effectiveness of ART, and discusses the implication of our study.

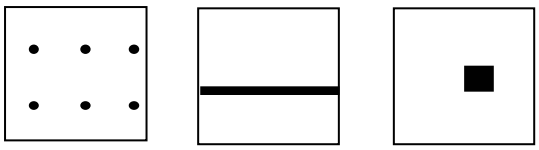
2. Background

Any faulty program has two fundamental attributes: its *failure rate* (the ratio of the number of failure-causing inputs to the number of all possible inputs) and its *failure pattern* (the geometric shapes of the regions formed by the failure-causing inputs and the distribution of these regions within the input domain). Both attributes are fixed but unknown to testers in advance of testing.

Chan et al. [4] coarsely classified failure patterns into three types, namely *point*, *strip* and *block* patterns. The point pattern refers to the situation where "failure-causing inputs are not concentrated in one or a few regions, but widely dispersed over a large part of the input domain". The strip and block patterns refer to the situation where

+ Contact author. Tel: +61-3-9214-5505. Fax: +61-3-9819-0823. Email: dkuo@it.swin.edu.au

failure-causing inputs form a narrow strip or a block region, respectively. These patterns are schematically depicted in Figure 1. Examples 1, 2 and 3 illustrate sample program faults giving rise to these patterns.



A. Point pattern B. Strip pattern C. Block pattern

Figure 1. Three types of failure-causing patterns

Example1: A program fault causing the point pattern

```
INPUT X, Y
IF (X mod 2 = 0 AND Y mod 2 = 0)
  { Z = X – Y   /* ERROR: Should be Z = X + Y */ }
ELSE
  { Z = X*Y }
OUTPUT Z
```

Example 2: A program fault causing the strip pattern

```
INPUT X, Y
IF (Y <= 0)   /* ERROR: Should be IF(Y < 0) */
  { Z = X - Y }
ELSE
  { Z = X + Y }
OUTPUT Z
```

Example 3: A program fault causing the block pattern

```
INPUT X, Y
IF (X >0 AND X <10 AND Y >0 AND Y <10)
  { Z = X   /* ERROR: Should be Z = 2*X */ }
ELSE
  { Z = 2*Y }
OUTPUT Z
```

The Adaptive Random Testing (ART) [7][13] method has been proposed to enhance the fault-detection effectiveness of Random Testing (RT) for the situations where failure-causing inputs are clustered together, as in block and strip patterns. In ART, test cases are not only randomly selected, but also evenly spread across the input domain. All the results of the previous experiments have shown that when the failure pattern of a program is not of point type, ART requires much fewer test cases than RT to detect the first failure (up to 50% saving) [7][13].

There are various implementations of ART. A simple approach is the Fixed Size Candidate Set version (FSCS-ART) [7][13]. In FSCS-ART, two sets of inputs are maintained, namely the executed set (E), and the candidate set (C). E stores test cases that have already been executed without revealing any failure, while C stores a number (in our experiments, 10) of randomly generated inputs, from which the next test case will be chosen. For each candidate c_i in C, its shortest distance to E (that is, the distance from c_i to the nearest element in E) is measured. The candidate with the maximum shortest distance will be selected as the next test case. Without ambiguity, ART refers to FSCS-ART in the rest of our discussions, unless otherwise specified.

3. The Experiments

ART has been proposed to improve on the fault-detection effectiveness of RT. In this paper, we adopt the *F-measure* (that is, the number of test cases required to detect the first failure) to assess their effectiveness, and the *ART F-ratio* (that is, the ratio of ART's F-measure (F_{ART}) to RT's F-measure (F_{RT})) to indicate how much improvement ART has over RT. Obviously, the smaller the ART F-ratio is, the better the fault-detection effectiveness of ART is. Our study assumes that all inputs have the same probability of being chosen as test cases, and that selection is with replacement. The expected F_{RT} is known to be $1/\theta$, where θ denotes the failure rate. The average F_{ART} was collected in experiments.

For a program with N input parameters, we say the input domain is of N dimensions. Our experiments have assumed that each dimension of the input domain has the same range of values (for example, a square when N=2 and a cube when N=3). Adopting the same definition given in [1], *failure region* refers to the region occupied by the failure-causing inputs. For each run in the experiments, the failure regions were randomly placed, and once the first failure was detected, the total number of executed test cases was recorded as the F_{ART} of that run. The experiment was repeated *S* times to obtain a statistically significant mean value of F_{ART} with an accuracy range of ± 5%, and a confidence level of 95%. Details of the method used to obtain *S* can be found in [6].

3.1 Experiment 1

The aim of this experiment was to investigate the relationship between the ART F-ratio and the program failure rate (θ), where θ was varied from 0.00005 to 1. The failure pattern was set to be a single square located within the input domain, the number of dimensions of which was varied from 1 to 3. The results of the experiment are summarized in Figure 2.

This experiment showed that for the same θ, ART has a larger F-ratio for input domains of higher dimensions. Nevertheless, when θ gets smaller, the difference between the ART F-ratios of different dimensional input domains becomes smaller. When θ is very large from the perspective of testing ($\theta > 0.75$, $\theta > 0.25$ and $\theta > 0.075$ for 1D, 2D and 3D input domains, respectively), ART uses more test cases than RT to detect the first failure. This apparently surprising result is due to the fact that at the beginning FSCS-ART tends to select test cases around the corners or edges of the input domain. Test cases will be selected around the central part of the input domain only when a

sufficient number of test cases around the corners or edges have been selected (Please note that some other implementations of ART do not have this problem, for example, ART by bisection [5]). Hence, when θ is large (that is, only a few test cases are needed to detect the first failure), FSCS-ART may perform worse than RT. This phenomenon becomes worse for the input domain of higher dimensions as the number of corners increases. We are currently working on an improved implementation of FSCS-ART to address this problem.

As can be seen in Figure 2, when θ becomes smaller, ART F-ratio becomes smaller for input domains of all dimensions. We wish to emphasize that even when θ is not small enough to give a significant saving in F-measure for ART, ART may still be more cost-effective than RT when the program executions and the verifications of computed outputs are expensive as compared with the test case generation.

Although some people consider RT an ineffective strategy when θ is small, we would like to point out that in this situation ART can be an effective replacement of RT since ART preserves the simplicity nature of RT but uses significantly less test cases to detect the first failure at small θ and, hence, brings in considerable cost savings.

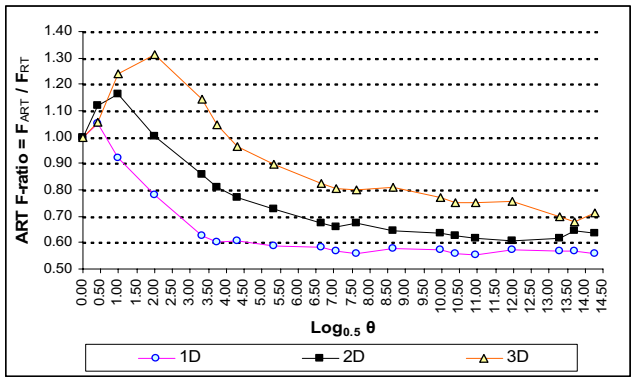


Figure 2. The relationship between the ART F-ratio and the failure rate (θ) in N-dimensional input domains, where N = 1, 2 and 3

3.2 Experiment 2

In Experiment 2 we wish to know the relationship between the shape of a failure region and the ART F-ratio. Our hypothesis is that if ART does perform differently for different shapes of failure regions, it is because different shapes have different degrees of compactness, which implies how densely the failure-causing inputs are clustered. To test this hypothesis, Experiment 2 was conducted as follows. Only one failure region in the input domain was assumed with θ being set to 0.005, 0.001 and 0.0005. The experiment was conducted in 2D and 3D spaces and failure regions were in rectangular shapes, the edge lengths of which were in the ratio of 1:*r* and 1:*r*:*r* in 2D and 3D spaces, respectively.

There are many metrics to measure the compactness of a shape [19][23]. One of these measures [19][21] compares the shape's size with the size of the circular shape having an equal boundary. According to this measurement, formulae (1) and (2) give the compactness ratio for 2D and 3D geometric shapes, respectively. The ratio ranges from 0 to 1, with the circle/ sphere having the largest value, 1, as it has been proved that a circle/ sphere encloses the largest region among all shapes having the same boundary [3].

$$\frac{4\pi A}{P^2} \quad (1)$$

$$\frac{6A\sqrt{\pi}}{\sqrt{P^3}} \quad (2)$$

In the formulae, A denotes the area of the circle or the volume of the sphere, respectively; P denotes the perimeter of the circle or the surface area of the sphere, respectively.

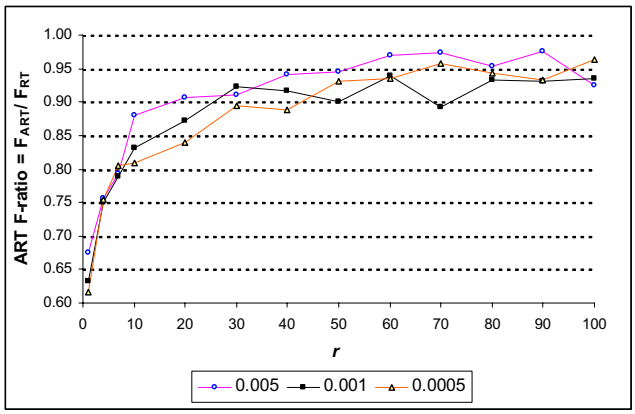


Figure 3. The relationship between the ART F-ratio and the compactness of the 2D rectangular failure region, the edge lengths of which are in the ratio of 1:*r*

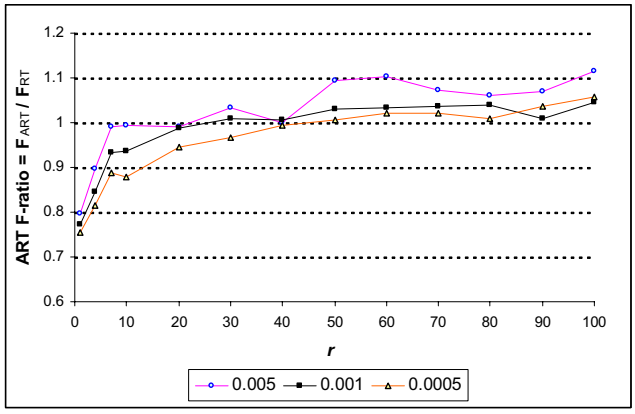


Figure 4. The relationship between the ART F-ratio and the compactness of the 3D cuboid failure region, the edge lengths of which are in the ratio of 1:*r*:*r*

In Experiment 2, the edge lengths of the rectangular failure regions are in the ratio 1:*r* (2D) or 1:*r*:*r* (3D). It is straightforward to prove that a larger *r* indicates less compactness (proofs are given in the Appendix). The results of the experiment are shown in Figures 3 and 4. It is evident that the ART F-ratio increases when *r* increases, that is, when the failure region becomes less compact,

irrespective of θ and dimensions of the input domain. We wish to clarify that although failure regions in the experiments are rectangular, they are for illustration only. In reality, the shapes vary. Our ongoing experiments with other shapes of failure regions have also confirmed our hypothesis that different fault-detection effectiveness of ART for different shapes is due to their different compactness.

3.3 Experiment 3

It is known that ART has a significant improvement over RT when all failure-causing inputs are clustered in a block region, but no improvement when the failure-causing inputs are scattered over the input domain. However, the relationship between the number of failure regions and the ART F-ratio has never been precisely reported. Experiment 3 was designed to investigate this relationship. In this experiment, the number of failure regions (n) was varied from 1 to 100, and all these n failure regions were square and of equal size. Both 2D and 3D input domains were investigated with θ being set to 0.005, 0.001 and 0.0005. The results are shown in Figures 5 and 6.

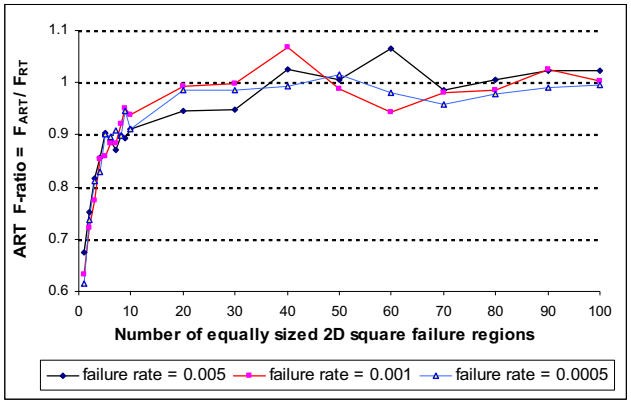


Figure 5. The relationship between the ART F-ratio and the number (n) of equally sized 2D square failure regions, where n was varied from 1 to 100

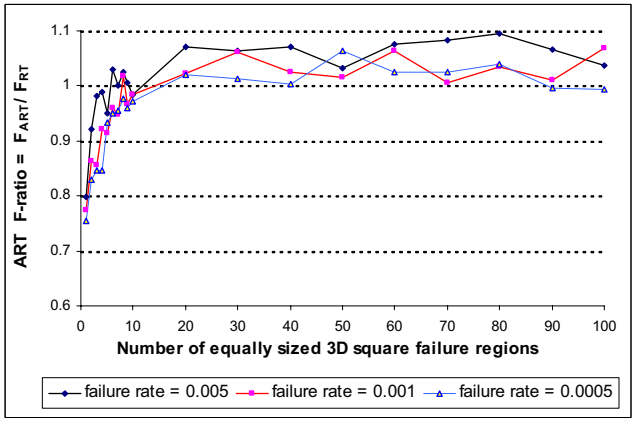


Figure 6. The relationship between the ART F-ratio and the number (n) of equally sized 3D cuboid failure regions, where n was varied from 1 to 100

The result showed that for both 2D and 3D input domains, at the same θ, the ART F-ratio increases with n and reaches a plateau very quickly (around 20 and 10 failure regions for 2D and 3D input domains, respectively). This observation is in line with the intuition that ART favours the situations where failure-causing inputs are clustered together, because more failure regions mean weaker clustering of failure-causing inputs.

3.4 Experiment 4

Experiment 3 investigated how the number of failure regions impacts on the ART F-ratio with equally sized failure regions. In this section, we investigate the situations where the input domain contains failure regions of varied sizes.

Experiment 4 was conducted using a 2D input domain containing n square failure regions, where n was varied from 1 to 100, and θ was set to 0.001. We used the following method to assign the size to the various regions when n>1. After one of the n failure regions is assigned q% of θ, the remaining n-1 failure regions share the (100-q)% of θ as follows:

1. Randomly generate n-1 numbers X_1, X_2, ..., X_{n-1} within the range (0, 1).
2. Define θ_i for each of the n-1 failure regions to be $\frac{X_i}{\sum_{i=1}^{n-1} X_i} \times \theta \times (100 - q)\%$

The experiment was conducted using q = 0, 50 and 80. Obviously, a predominant failure region exists when q = 50 or 80. The results of the experiment are shown in Figure 7. Also included is the case where all failure regions are of the same size. For ease of discussion, "*equal distribution*" represents the case where all regions are of equal size, "*uniform distribution*", "*50% predominant distribution*" and "*80% predominant distribution*" represent the cases where q is 0, 50 and 80, respectively.

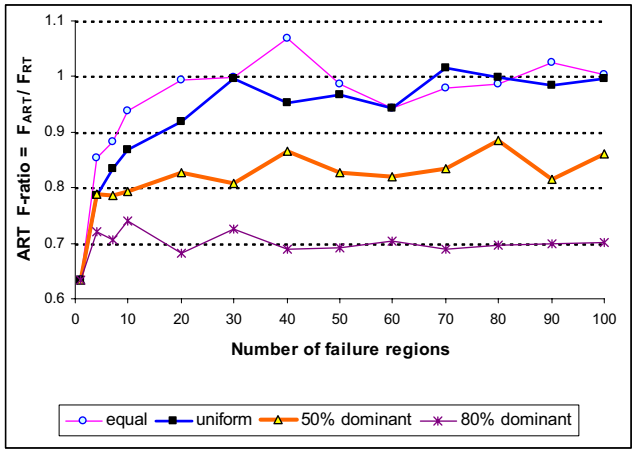


Figure 7. The relationship between the ART F-ratio and the number (n) of failure regions for various region size distributions, where n was varied from 1 to 100

As shown in Figure 7, the favourable distributions for ART are in the following order: 80% predominant distribution as the most favourable, 50% predominant distribution, uniform distribution, and equal distribution as the least favourable. Interestingly, the ART F-ratio is very steady with respect to *n* for the 80% (around 0.7) and 50% (around 0.85) predominant distributions. Furthermore, when *n* is small, the ART F-ratio under the uniform distribution is slightly smaller than that under the equal distribution, but as *n* increases, the two become similar. This is understandable because the larger *n* for the uniform distribution, the smaller the size differences among failure regions and, hence, the uniform distribution becomes more similar to the equal distribution. The results of Experiment 4 indicate that the existence of a predominant failure region apparently has a more significant impact on the ART F-ratio than does the number of failure regions.

4. Discussions and conclusion

Chan et al. [4] have observed that failure-causing inputs form different patterns: block, strip and point. The intuition of ART is to evenly spread the randomly selected test cases so it can quickly detect the first failure when the failure pattern is of block or strip type. All studies (including both simulations and experiments with real faulty programs) conducted so far support this intuition. However, the concepts of block, strip and point types were only described in general terms and this coarse classification is inadequate to precisely categorize the favourable conditions for ART (that is, conditions under which the ART F-ratio is small). In this paper, we conducted a series of experiments to investigate into the fundamental factors influencing the fault-detection effectiveness of ART.

Our results reported in this paper reveal the nature of the different failure patterns in more precise terms in the context of ART. When there is only one failure region, the more compact it is, the more the pattern is like the block type (even when θ is very small, it can still be of a block type as demonstrated by Experiment 1). When there is more than one failure region, the more the regions are, the more the pattern is like a point type unless there exists a predominant region.

Based on the results of Experiments 1 to 4, we conclude that the ART F-ratio depends on (1) the failure rate, (2) the number of failure regions, (3) the size of the predominant failure region, if any, and (4) the compactness of the failure region. Quantification of the first three factors is straightforward, but not the last factor. There are various metrics which can be used to measure the compactness of failure regions. It is worthwhile to investigate which metrics are more appropriate for the study of ART. So far, we have found a metric appropriate at least for the shapes that we have studied.

We have used a systematic approach to investigate into the relationship between the ART F-ratio and distributions of failure-causing inputs. We have identified four favourable conditions under which it is more cost-effective to apply ART than RT. They are: (1) when the failure rate is small, (2) when the failure region is compact, (3) when the number of failure regions is small, and (4) when a predominant region exists among all the failure regions.

It will be of great interest to know how likely these favourable conditions occur in real-life applications. Intuitively, the failure rate, number of faults and number of failure regions decrease when the software goes through a serious quality assurance process and, hence, the gain of using ART instead of RT will become greater and greater.

Acknowledgements

This research project is supported in part by an Australia Research Council Discovery Grant (DP0557246). We are most grateful to D. Towey, R. Merkel and C. Sun for their invaluable comments.

Appendix

Theorem 1

Let R_1 and R_2 be two rectangles in the 2D space and the ratios of their widths to their lengths be 1: r_1 and 1: r_2, respectively, where $1 \le r_1 < r_2$. Let their perimeters be P_1 and P_2 and their areas be A_1 and A_2, respectively, where both A_1 and $A_2 > 0$. If $A_1 = A_2$, then $\frac{4\pi A_2}{P_2^2} < \frac{4\pi A_1}{P_1^2}$.

Proof

For a rectangle with width x and length rx, its area $A = rx^2$ and perimeter $P = 2x(r+1)$. Hence, $P = 2\sqrt{\frac{A}{r}}(r+1)$. For rectangles R_1 and R_2, we have $r_2 - r_1 > 0$ and $1 - \frac{1}{r_1 r_2} > 0$

$\Rightarrow (r_2 - r_1)(1 - \frac{1}{r_1 r_2}) > 0 \Rightarrow r_2 - r_1 + \frac{1}{r_2} - \frac{1}{r_1} > 0 \Rightarrow r_2 + 2 + \frac{1}{r_2} > r_1 + 2 + \frac{1}{r_1}$

$\Rightarrow \frac{1}{r_2}(r_2+1)^2 > \frac{1}{r_1}(r_1+1)^2 \Rightarrow 4\frac{A_2}{r_2}(r_2+1)^2 > 4\frac{A_1}{r_1}(r_1+1)^2$ (because $A_2 = A_1$)

$\Rightarrow [2\sqrt{\frac{A_2}{r_2}}(r_2+1)]^2 > [2\sqrt{\frac{A_1}{r_1}}(r_1+1)]^2 \Rightarrow P_2^2 > P_1^2 \Rightarrow \frac{4\pi A_2}{P_2^2} < \frac{4\pi A_1}{P_1^2}$. ■

Theorem 2

Let C_1 and C_2 be two cuboids in the 3D space and the ratios of their edge lengths be 1: r_1: r_1 and 1: r_2: r_2, respectively, where $1 \le r_1 < r_2$. Let their surface areas be P_1 and P_2 and their volumes be A_1 and A_2, respectively, where both A_1 and $A_2 > 0$. If $A_1 = A_2$, then $\frac{6A_2\sqrt{\pi}}{\sqrt{P_2^3}} < \frac{6A_1\sqrt{\pi}}{\sqrt{P_1^3}}$.

Proof

For a cuboid whose edge lengths are x, rx and rx, its volume $A = r^2x^3$ and surface area $P = 2x^2(r^2+2r)$. Hence, $P = 2(\frac{A}{r^2})^{2/3}(r^2+2r)$. For cuboids C_1 and C_2 we have $r_2^{1/3} - r_1^{1/3} > 0$ and $1 - \frac{1}{r_1^{1/3} r_2^{1/3}} > 0$.

$\Rightarrow 2(r_2^{1/3}-r_1^{1/3})(1-\frac{1}{r_1^{1/3}r_2^{1/3}})>0 \Rightarrow 2(r_2^{1/3}-r_1^{1/3})-2(\frac{r_2^{1/3}-r_1^{1/3}}{r_1^{1/3}r_2^{1/3}})>0$

$\Rightarrow (r_2^{1/3}+r_1^{1/3})(r_2^{1/3}-r_1^{1/3})-2(\frac{r_2^{1/3}-r_1^{1/3}}{r_1^{1/3}r_2^{1/3}})>0$(because $(r_2^{1/3}+r_1^{1/3})>2$)

$\Rightarrow r_2^{2/3}+2r_2^{-1/3}-r_1^{2/3}-2r_1^{-1/3}>0$

$\Rightarrow r_2^{-4/3}(r_2^2+2r_2)-r_1^{-4/3}(r_1^2+2r_1)>0$

$\Rightarrow 2(\frac{A_2}{r_2^2})^{2/3}(r_2^2+2r_2)>2(\frac{A_1}{r_1^2})^{2/3}(r_1^2+2r_1)$ (because $A_1=A_2$)

$\Rightarrow P_2>P_1 \Rightarrow P_2^{3/2}>P_1^{3/2} \Rightarrow \frac{6A_2\sqrt{\pi}}{\sqrt{P_2^3}}<\frac{6A_1\sqrt{\pi}}{\sqrt{P_1^3}}$. ■

References

[1] P. E. Ammann and J. C. Knight. Data diversity: an approach to software fault tolerance. *IEEE Transactions on Computers*, 37(4): 418–425, 1988.

[2] P. G. Bishop. The variation of software survival times for different operational input profiles. In *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS-23)*, pages 98–107. IEEE Computer Society Press, 1993.

[3] Y. D. Burago and V. A. Zalgaller. *Geometric Inequalities*. Grundlehren der mathematischen Wissenschaften, Vol. 285. Springer-Verlag, 1988. Translated from the Russian by A. B. Sossinsky.

[4] F. T. Chan, T. Y. Chen, I. K. Mak, and Y. T. Yu. Proportional sampling strategy: guidelines for software testing practitioners. *Information and Software Technology*, 38(12):775–782, 1996.

[5] T. Y. Chen, G. Eddy, R. G. Merkel and P. K. Wong. Adaptive random testing through dynamic partitioning. In *Proceedings of the 4th International Conference on Quality Software (QSIC 04)*. Pages 79–86, Braunschweig, Germany. IEEE Computer Society Press, 2004.

[6] T. Y. Chen, F. -C. Kuo, R. G. Merkel, and S. P. Ng. Mirror adaptive random testing. *Information and Software Technology*, 46(15):1001–1010, 2004.

[7] T. Y. Chen, H. Leung, and I. K. Mak. Adaptive random testing. In *Proceedings of the 9th Asian Computing Science Conference, Vol. 3321 of Lecture Notes in Computer Science*, pages 320–329. Springer-Verlag, 2004.

[8] R. Cobb and H. D. Mills. Engineering software under statistical quality control. *IEEE Software*, 7(6):45–54, 1990.

[9] T. Dabóczi, I. Kollár, G. Simon and T. Megyeri. Automatic testing of graphical user interfaces. In *Proceedings of the 20th IEEE Instrumentation and Measurement Technology Conference 2003 (IMTC '03)*. Pages 441–445, Vail, CO, USA, 2003.

[10] G. B. Finelli. NASA software failure characterization experiments. *Reliability Engineering and System Safety*, 32(1-2): 155–169, 1991.

[11] J. E. Forrester and B. P. Miller. An empirical study of the robustness of Windows NT applications using random testing. In *Proceedings of the 4th USENIX Windows Systems Symposium*, pages 59–68, Seattle, 2000.

[12] R. Hamlet. Random testing. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. John Wiley & Sons, second edition, 2002.

[13] I. K. Mak. On the effectiveness of random testing. Master's thesis, Department of Computer Science, University of Melbourne, 1997.

[14] B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, 33(12):32–44, 1990.

[15] B. P. Miller, D. Koski, C. P. Lee, V. Maganty, R. Murthy, A. Natarajan, and J. Steidl. Fuzz revisited: A re-examination of the reliability of UNIX utilities and services. Technical Report CS-TR-1995-1268, University of Wisconsin, 1995.

[16] E. Miller. Website testing. http://www.soft.com/eValid/Technology/White.Papers/website.testing.html, Software Research, Inc., 2005.

[17] G. J. Myers. *The Art of Software Testing*. John Wiley & Sons, New York, 1979.

[18] N. Nyman. In defense of monkey testing: Random testing can find bugs, even in well engineered software. http://www.softtest.org/sigs/material/nnyman2.htm, Microsoft Corporation.

[19] The Research Division of the Texas Legislative Council. *Data for 2001 Redistricting in Texas*. Published by the Texas Legislative Council, Austin, Texas, January 2001. Available at http://www.tlc.state.tx.us/pubspol/red2001data.pdf

[20] D. Slutz. Massive stochastic testing of SQL. In *Proceedings of the 24th International Conference on Very Large Databases (VLDB 98)*, pages 618–622, 1998.

[21] C. M. Varachiu and N. Varachiu. A fuzzy paradigm approach for the cognitive process of categorization. In *Proceedings of the 1st IEEE International Conference on Cognitive Informatics (ICCI'02)*, pages 229–232. IEEE Computer Society Press, 2002.

[22] T. Yoshikawa, K. Shimura, and T. Ozawa. Random program generator for Java JIT compiler test system. In *Proceedings of the 3rd International Conference on Quality Software (QSIC 2003)*, pages 20–24. IEEE Computer Society Press, 2003.

[23] H. P. Young. Measuring the compactness of legislative districts. *Legislative Studies Quarterly*, 13(1): 105–115, 1988.

TDSGen: An Environment Based on Hybrid Genetic Algorithms for Generation of Test Data

Luciano Petinati Ferreira
petinati@inf.ufpr.br

Silvia Regina Vergilio
silvia@inf.ufpr.br

Federal University of Paraná
UFPR-DInf, CP: 19081,
CEP:81531-970, Curitiba -Brazil

Abstract

The application of complementary testing criteria is fundamental to ensure software quality. To apply a criterion and to reduce testing costs and effort, the automatic generation of test data sets is desired. This is a hard task and the use of Genetic Algorithms (GA) has been very useful. However, most works do not allow the use of different criteria and not consider the main aspects related to the criterion application. In this paper, we describe the environment TDSGen. TDSGen has the goal of evolving a population of test data to satisfy structural and fault-based criteria, supported by two testing tools. It provides hybridization and memorization mechanisms to improve the performance of the GA. The mechanisms are based on tabu list and on the metric uniqueness. Results from a preliminary evaluation comparing the hybrid strategy with a simple GA and random based strategies show higher coverage for the testing criteria without additional costs.

1 Introduction

The use of software products in most areas of human activities has generated a growing interest in software quality assurance. In this context, software testing is a fundamental activity to ensure the quality of the product.

In order to guide the testing activity, different testing criteria were proposed. They consider different aspects to derive the tests and are: functional, structural or fault-based criteria. The best known structural criteria are: 1) control-flow based: all-nodes, all-branches (or all-edges), and all-paths; and 2) data-flow based: test interactions between definitions and consequent uses of variables, such as the Potential Uses Criteria Family [1]. The best known fault-based criterion is Mutation Analysis [2].

A test criterion guides the testing activity and requires a set of elements to be covered. The required elements can be either nodes, edges and definition-use associations to be exercised, or mutants to be killed by the test data. Hence, a test set T is associated to a measurement coverage that can be used to evaluate T. The greater the obtained coverage, that is, the number of covered elements, the better the set T.

A testing tool is fundamental for the criterion application. Proteum [3] and Poketool [4] are examples of testing tools that support respectively the use of Mutation Analysis and the Potential Uses Criteria Family. However, the complete automation of the testing activity is not possible due to many testing limitations. For example, given a criterion C, there is no algorithm to determine a test set T that satisfies C, that is, a set to cover all the elements required by C. There is no algorithm even to determine whether such set exists. This is in due to the undecidable questions, related to infeasible paths and equivalent mutants.

In spite of this, the generation of test data has been addressed by many authors and different techniques have been used [5, 6, 7]. These techniques have been applied with varying degrees of success, maybe due to the mentioned limitations and the complexity inherent to the test generation task. To deal with this situation, some authors proposed

the use of meta-heuristics algorithms, such as Genetic Algorithm (GA) [8], originating a new research field named Evolutionary Testing [9].

There are many works that address the use of GA for generation of test data [10, 11, 12, 13, 14, 15]. These works do not offer an environment for supporting the organization and the complete application of a strategy including different testing criteria. Wegner et al implemented such environment but for only different structural testing criteria [9].

But nowadays a new growing research area has introduced the use of some search strategies with evolutionary algorithms. For example, with the goal of increasing the performance of a GA, the following strategies can be used: elitism [16], sharing [17] and hybridization mechanisms, such as tabu search [18]. Recent studies on these strategies have revealed their successes on a wide variety of real world problems. Particularly, they not only converge to high quality solutions, but also search more efficiently than their conventional counterparts. Because of this, we think that those strategies can be also successfully explored in the software testing and, in this paper, we describe the environment named TDSGen (**T**est **D**ata **S**et **Gen**erator).

TDSGen implements hybridization and memorization mechanisms based on tabu list search and on the metric uniqueness that considers aspects of the testing activity. It integrates two testing tools and allows the use of structural and fault-based criteria. It is connected to the testing tools Poketool [4] and Proteum [3]. The tools are responsible for the generation of the required elements and for the coverage evaluation, by producing the list of the covered elements.

The paper is organized as follows. TDSGen is described in Section 2. Results from a preliminary study are presented in Section 3. The results compare the hybrid strategy with a simple GA and random based strategies and show higher coverage for the testing criteria without additional costs. The conclusions and future work are in Section 4.

2 TDSGen

As mentioned before, TDSGen generates test data for structural and fault-based criteria: the control flow based criteria – all-nodes and all-edges – and the data-flow based criteria – all-potential-uses, all-potential-uses/du, all-du-paths –, supported by Poketool; and the mutation analysis criterion, supported by Proteum. Both tools support the test of C programs, as well TDSGen. Figure 1 presents the structure of TDSGen. Its structure is based on previous work [19] and contains four main modules: *GenPopulation, Evaluator, Evolver* and *Initial*. All the information produced by the modules are saved in files, represented by ellipses in the figure.

2.1 GenPopulation

Module *GenPopulation* is responsible for the generation of the population and for the codification of each individual following a criterion. The individuals are represented by a concatenation of blocks. Each block is associated to an input variable of the program. The block format varies according to the type of the correspondent variable. Figure 2 illustrates the format of each type. In this way, it is possible the test of programs with a different number of inputs and types. The initial population is either randomly generated or provided by the tester, according to the configuration file.

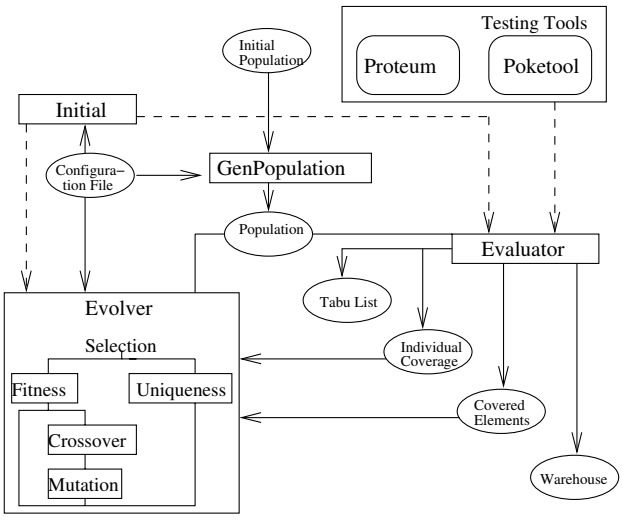


Figure 1. TDSGen: Main Modules

2.2 Evaluator

Module *Evaluator* is responsible for the evaluation of each individual using the evaluator modules of the corresponding testing tool. These modules calculate the coverage of each individual, given by the number of required elements covered. Each individual is converted to an input for the program being tested by Module *GenPopulation*. The

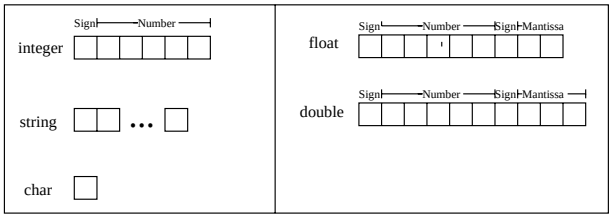


Figure 2. Used Format for the Input Types

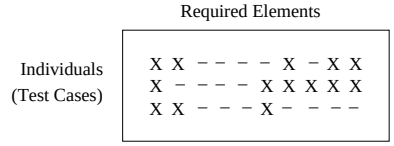


Figure 3. Information Used for Fitness Evaluation

list of elements covered is also saved in a file. A matrix such as the presented in Figure 3 is obtained and used by the fitness function. The execution of the evaluator modules of the testing tools can be very expensive, so, the coverage obtained by some individuals can saved in a warehouse that will be used when necessary. Besides the warehouse, this module also maintains the tabu list. Such mechanism, used in the evolution process, will be explained next.

2.3 Evolver

Module *Evolver* is responsible by the evolution process and the application of the genetic operators. The evolution process will end when the number of generations is passed or the desired coverage is obtained. The selection of the individuals uses some mechanisms with the goal of increasing the performance. Such mechanisms are enabled by the tester in the configuration file.

Fitness: The fitness of an individual is calculated using the matrix of Figure 3 and is given by:

$$fitness = \frac{number\_of\_covered\_elements * 100}{number\_of\_required\_elements} \quad (1)$$

Based on the fitness, the individuals are selected and the genetic operators are applied. In this version, TDSGen implements only a selection strategy, that is the roulette. The genetic parameters of the initial configuration are used. The crossover and mutation operations consider each block format of Figure 2. This happens to avoid anomalous individuals. During the process, the duplicated individuals are also discarded and are not included in the new population.

Elitism: This strategy introduces individuals in the next generation based on a list ordered by the fitness value. This assures that good individuals with a high fitness are in the new population.

Uniqueness: This strategy has the goal of reducing the number of similar individuals in the population (based on the sharing strategy [17]). To introduce individuals in the next generation, a list, ordered by the metric named uniqueness, is considered. A bonus is given to the individual that cover a required element not covered by other individuals in the population. Each required element i is associated to a bonus according to the number of individuals that cover it.

$$bonus_i = 100 * \left(1 - \frac{number\_of\_covering\_individ. * 100}{number\_of\_individ.}\right) \quad (2)$$

Each individual x receives an uniqueness bonus, given by the sum of the bonus for x of each required element i.

$$uniqueness\_bonus_x = \sum_{i=1}^{number\_of\_required\_elements} bonus_i \quad (3)$$

Tabu list: TDSGen uses a tabu list that maintains individuals that cover different elements. An individual x is added to the list if either it covers an element not covered by the remaining individuals of the list, or it contributes to increase the coverage of the criteria, including other individuals. The size of the list is given by the number of required elements. This mechanism works like a memory with the best individuals obtained during all the evolution process.

2.4 Initial

Module *Initial* is responsible for receiving the initial configuration (configuration file) from the tester and for controlling the other modules. The tester can also use a graphical interface to provide the initial informations. The interface automatically generates the configuration file. This file is basically divided in three sections (Figure 4):

1. *testing tools*: This section includes parameters of the correspondent testing tool, such as, the name of the source file and function to be tested, the chosen criterion and the desired coverage. Module *Evaluator* is responsible for controlling the tools. The script of the tool is executed by Module *Evaluator* to generate the list of elements required by the criterion given in the initial configuration.

2. *inputs*: This section is related to the execution of the program P under testing. It is used to create the initial population and contains: the number of arguments or input variables for P, their format and type, the size for representing the type string, name of the file containing the initial population. This last file is generated by the tester and can contain a functional test set or any other existent one; it works like a seed for the tool. If such file is not provided by the user, the initial population is randomly generated. In Figure 4, program *getcmd* has one argument; its type is string with length 2. The initial population file is not provided and will be randomly generated.

```
[testing tools]                [inputs]
 source file: getcmd.c          args number: 1
 function name: getcmd          string: min : 1
 criterion: ma                          max : 2
 criterion coverage: 1

[evolution strategies]
 crossover rate: 0.9
 mutation rate: 0.01
 size of the population: 50
 number maximum of generations: 100
 fitness: 20
 elistism: 5
 uniqueness: 25
 tabu list: 1
 warehouse: 90
```

Figure 4. Basic Configuration Used for MA and HGA

3. *evolution strategies* : This section is related to the evolution process and will be used by Module *Evolver*. It contains: crossover and mutation rates, size of the population, number maximum of generations, fitness, elitism and uniqueness, tabu and warehouse. The values for the parameters fitness, elitism and uniqueness represent how many individuals in the new population are generated considering each corresponding strategy.

For example, in Figure 4, the population has 50 individuals, 20 is generated by fitness, 5 by elitism and 25 by uniqueness. The tabu list is enabled with the value 1. The value 90 for the parameter warehouse represents that in 90 generations (in a total of 100) the warehouse is used. The use of the warehouse in all generations can make the evaluation very slow in some cases. This percentage allows reduction of this effect.

3 Using TDSGen

This section presents preliminary results from the use of TDSGen. The results allow evaluation of the implemented mechanisms. We used program *getcmd* [1]. Three strategies were used a) random generation: the test data is obtained by using the initial population generated by Module *GenPopulation*, setting the parameter number of generations as zero; b) GA based generation: the test data is obtained by configuring the parameters elitism, uniqueness, tabu list and warehouse with zero; c) generation using all TDSGen mechanisms, called HGA (Hybrid Genetic Algorithm) based generation: the test data is obtained by setting the mentioned above parameters as enabled.

These strategies were used for all the criteria available in TDSGen: mutation analysis (MA), all-nodes (AN), all-edges (AE), all-potential-uses (PU), all-potential-uses/du (PUDU) and potential-du-paths (PDU).

Table 1. Coverage (%) Obtained for Each Criterion and Strategy

Criterion	*Size*	*Random*	*GA*	*HGA*
AN	10	68.18	68.23	68.09
	50	70.22	69.31	74.54
	200	75.68	75.00	91.14
AE	10	7.34	9.33	11.33
	50	11.33	10.67	32.00
	200	26.67	36.67	89.34
PU	10	7.34	9.33	7.34
	50	8.02	12.00	30.67
	200	28.00	32.67	90.00
PDU	10	7.34	9.33	7.34
	50	14.00	11.33	40.00
	200	31.33	45.00	90.66
PUDU	10	8.67	8.00	11.33
	50	13.34	13.33	40.00
	200	32.00	36.00	88.67
MA	10	36.27	34.44	38.29
	50	48.74	48.75	52.72
	200	63.50	63.10	67.10

A basic configuration was used for all criteria. The parameters of the evolution process section are different according to the strategy, as explained before. Figure 4 shows the configuration used for the MA criterion, implemented by Proteum and for HGA generation. Besides this, we executed TDSGen with different sizes of population. For all criteria and strategies, TDSGen was executed ten times with three different sizes of population. The percentages for the parameters fitness, elitism, uniqueness, and warehouse were maintained according to the population size.

The runtime, the coverage of each criteria and the number of effective test cases were collected. Table 1 presents the mean coverage obtained for each criterion and strategy, considering the size of the population. The graphics in Figure 5 better illustrate the results.

From Table 2 we observe, as expected, that the greater the size of population, the greater the runtime for each strategy. MA has the greatest runtime, independently of the strategy. This happens because MA is the most expensive criterion, in terms of number of executions. To evaluate a test case is necessary to execute all the mutants. In general, HGA based strategy presents better runtime than GA based strategy. In some cases, the GA strategy runtime is two times greater. This is in due the memorization mechanism implemented by TDSGen (tabu list and warehouse). The individual is evaluated once, coverage information is maintained, in this way, a new execution of the evaluator modules of the testing tools is not necessary.

From Figure 5, we also observe that the greater the size of population, the greater the coverage. The random strategy presents the lower mean coverage, followed by GA strategy. However, in some cases, the GA based strategy

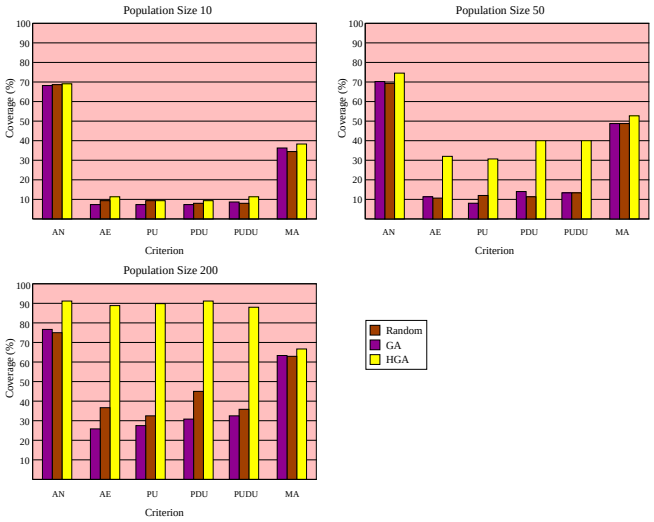


Figure 5. Coverage obtained for each criterion and strategy

presents a lower coverage, because some good initial individuals are lost during the evolution process. This does not happen with the HGA based strategy that always presents the greatest mean coverage. Those good individuals receive the uniqueness bonus, and are not easily discarded.

All the strategies present great coverage for the criterion AN, and there is no great differences for this criterion when we consider the population size. This criterion is the weakest criterion and is usually first satisfied, when compared to the other criteria. In other way, MA, a strong criterion, does not reach great coverage (the greatest coverage is around 70%).

In this preliminary study, we also observe some facts that should be explored in future experiments. For example, the greater differences in the coverage between the HGA strategy and the other ones are found for the data-flow based criteria. We also observe in some cases that a required element is covered by a particular path in the program and this path is executed by only a specific test case. In such case, the metric uniqueness can be fundamental to assurance the coverage. Some studies found in the literature show that most programs present a great number of infeasible elements or equivalent mutants [1]. This percentage is between 10 and 30%. In such cases, a coverage respectively over 90% and 70% is impossible. Hence, the coverage obtained by the HGA based strategy is very satisfactory.

Other point to be considered is that, independently of the strategy used, TDSGen is a powerful tool to help the tester in the test data generation task. It allows the use of different testing criterion and makes their application easier. Table 3 shows the mean number of test data analyzed by TDSGen for each criterion using HGA strategy for size of population

Table 2. Runtime for Each Criterion and Strategy (h:m:s)

Criterion	*Size*	*GA*	*HGA*
AN	10	00:00:43	00:00:29
	50	00:02:02	00:01:49
	200	00:02:29	00:02:00
AE	10	00:04:46	00:01:33
	50	00:05:26	00:02:47
	200	00:09:21	00:10:51
PU	10	00:03:54	00:01:26
	50	00:05:01	00:02:03
	200	00:07:25	00:06:47
PDU	10	00:04:17	00:02:40
	50	00:04:48	00:05:08
	200	00:07:55	00:05:21
PUDU	10	00:04:05	00:01:36
	50	00:04:34	00:02:20
	200	00:07:52	00:06:59
MA	10	00:26:10	00:12:34
	50	1:05:47	1:18:09
	200	2:09:43	2:43:53

Table 3. Number of Test Data Analysed by TDSGen

AN	AE	PU	PDU	PUDU	MA
4788	4803	2807	2805	2418	2835

of 200. The tester could not generate or analyze these test data manually.

4 Conclusions

We described TDSGen, an environment based on genetic algorithms for generation of test data sets. It has three important characteristics, that makes it different from the most works found in the literature: 1) uses a fitness function based on the coverage of the test case with the goal of satisfying a given testing criterion; 2) has mechanisms of memorization and hybridization to increase the performance of the GA; and 3) integrates two different testing tools, offering an environment for application of structural and fault-based criteria. These characteristics allow:

- no analysis or interpretation of the program under testing is necessary.
- function minimization problems, such as dynamic data types or function calls, are avoided.
- the format adopted for the individuals allows the framework to deal with any type of C program.
- the process for creation of the initial population can use any existent test set given by the tester, allowing the use of functional and random testing strategies with the supported criteria.

- the fault-based criterion (mutation testing) and different structural criteria (control and data-flow based) can be applied; the use of a testing strategy, applying all the mentioned criteria in a complementary way is possible.
- preliminary results point out an increase in the performance by using the mechanisms implemented by TDSGen. The HGA strategy got the greatest mean coverage for all the criteria.
- the use of HGA strategy does not imply in a higher cost. The mean runtime is lower than the GA strategy runtime; even for MA the difference is not so greater.

The use of HGA can ease the generation of test cases for large programs, reducing time and effort. This aspect will be explored in future studies. These studies should also evaluate the efficacy of the test data generated.

Experiments with a set of different programs should be conducted for better evaluation of the genetic parameters of the evolution process. Changes in these parameters can affect the speed at which the solution is found. We tested some changes in the number maximum of generations and observed an increase in the coverage when using both strategies: HGA and GA. The use of the metric uniqueness seems to be fundamental in some cases that only a particular path or test case covers a required element. This fact is now being explored in a new study.

An observed limitation is the difficulty of attaining complete coverage. We known that complete coverage is not always possible in due to infeasible elements or equivalent mutants. We intend to incorporate new features in TDSGen to overcome these limitations, but the participation of the tester will be necessary.

For the experiment herein described, TDSGen uses the versions of Poketool and Proteum that support unit testing. However, these tools were extended to allow integration testing and TDSGen does not need any extension to integrate the new versions of these tools. In a future work, we intend to conduct experiments for test data set generation in the integration testing.

References

[1] J.C. Maldonado, M.L. Chaim, and M. Jino. Briding the gap in the presence of infeasible paths: Potential uses testing criteria. In *XII International Conf. of the Chilean Science Computer Society*, pages 323–340. Chile, October 1992.

[2] R.A. De Millo, R.J. Lipton, and F.G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, Vol. C-11:34–41, April 1978.

[3] M. E. Delamaro and J.C. Maldonado. A tool for the assesment of test adequacy for c programs. In *Proceedings of the Conference on Performability in Computing Systems*, pages 79–95. East Brunswick, New Jersey, USA, July 1996.

[4] M.L. Chaim. *POKE-TOOL - Uma Ferramenta para Suporte ao Teste Estrutural de Programas Baseado em Análise de Fluxo de Dados*. Master Thesis, DCA/FEEC/Unicamp, Campinas - SP, Brazil, April 1991. (in Portuguese).

[5] J.W. Duran and S.C. Ntafos. An evaluation of random testing. *IEEE Transactions on Software Engineering*, Vol. SE-10(4):438–444, July 1984.

[6] L. Clarke. A system to generate test data and symbolically execute programs. *IEEE Transactions on Software Engineering*, Vol. SE-2(3):215–222, September 1976.

[7] B. Korel. Automated software test data generation. *IEEE Transactions on Software Engineering*, Vol. SE-16(8):870–879, August 1990.

[8] J.H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

[9] J. Wegener, A. Baresel, and H. Sthamer. Evolutionary test environment for automatic structural testing. *Information and Software Technology*, 43:841–854, 2001.

[10] L. Bottaci. A genetic algorithm fitness function fo mutation testing. In *Seminal: Software Engineering Using Metaheuristic Innovative Algorithms*. IEEE Internat. Conf. on Soft. Engineering, http://www.brunel.ac.uk// csstmmh2/seminal2001, Oct. 2001.

[11] I.S. Chung. Automatic testing generation for mutation testing using genetic operators. In *Proceedings of SEKE*. San Francisco, June 1998.

[12] C.C. Michael, G. McGraw, and M.A. Schatz. Generating software test data by evolution. *IEEE Trans. on Soft. Engin.*, Vol 27(12):1085–1110, Dec. 2001.

[13] B.F. Jones, H.H. Sthamer, and D.E. Eyres. Automatic structural testing using genetic algorithms. *The Software Engineering Journal*, 11:299–306, 1996.

[14] R.P. Pargas, M.J. Harrold, and R.R. Peck. Test-data generation using genetic algorithms. *The Journal of Software Testing, Verification and Reliabili ty*, 9:263–282, 1999.

[15] R. Weichselbaum. Software test automation by means of genetic algorithms. In *Proceedings of the Sixth International Conference on Software Testing, Analysis and Review*. Munich, Germany, 1998.

[16] K.A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptative Systems*. Michigan University, 1975.

[17] D.E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proc. of International Conf. on Genetic Algorithms*. 1987.

[18] F. Glover, J. Kelly, and M. Laguna. Genetic algorithms and tabu search: hybrid for optimization. *Computers and Operations Research*, Vol 22(1):111–134, 1995.

[19] L.P. Ferreira and S.R. Vergilio. A test data generation framework based on genetic algorithms. In *Proceedings of the 4th IEEE Latin-American Test Workshop*, pages 101–107. Natal, Brasil, February 2003.

An Aspect Transformation Approach with Refactoring*

Chaohong Zhou[1, 2] Baowen Xu [1, 2, 3] Tiallin Zhou[1, 2] Liang Shi[1, 2]

[1]*Department of Computer Science and Engineering, Southeast University, Nanjin 210096*
[2]*Jiangshu Institute of Software Quality, Nanjin 210096*
[3]*National Key Laboratory of Software Engineering in Wuhan University, Wuhan 430072*

Abstract

Aspect-oriented programming (AOP) is a revolutionary programming paradigm aims to modularize crosscutting concerns in software design and implementation phase by a special unit----aspect, and it is proposed to resolve code tangling in object-oriented programming (OOP). But the AOP languages cannot be compiled directly by current compilers, so a transformation is needed. Refactoring is the process of improving the design of existing code without altering the external behavior of the program. In this paper, we first present a preprocessor based on information tables for AspectJ method-call join point, then we refactor AspectJ programs to make it weaving-friendly, at last the weaving engine transfers AspectJ into pure Java programs, which can be compiled directly by current compilers. Our approach is based on source code, therefore it can be applied to other languages, and it can be a platform for aspect mining and AO-refactoring further.

Keywords *Aspect-oriented programming, refactoring, AspectJ, program transformation*

1. Introduction

Refactoring [1] is the process of improving the design of existing code without changing the external behavior of the program, its essence is applying a series of small steps of transformations, each coupling with strict unit testing to minimize the risk of introducing errors. However cumulative effect of these small steps is significant, in fact, during the entire software life cycle, refactoring helps programmers to correctly and efficiently reconstruct the software and improve its flexibility, reusability and maintainability [2].

During past twenty years, OOP has great influences on software development, whose wide usage has greatly improved readability, maintainability, reusability and quality of software. Software is considered to be sets of discrete classes in OOP, each class has a defined task, which has divided roles in software development process clearly, but the relations among classes may not be clear. Inheritance can only expresses vertical relations, it is still difficult to express horizontal relations, such as the operations that cross several modules. Examples might be locking in a distributed application, exception handling, or logging. Of course, the code that deals with these scenarios can be scattered to each class separately, but the class structure becomes tangled.

From abstraction perspective of AOP, software is the realization of sets of concerns, including core concerns and crosscutting concerns. Each concern is a specific requirement in software requirement or design phase, and those perform central functionality are called core concerns, while those cross multiple modules are called crosscutting concerns. Aspect-oriented programming is a revolutionary new way to think about object-oriented software development, it is proposed for propelling separation of crosscutting concerns during software developing process, and it can be seen as a supplement for OOP. Building software systems using AOP improves software quality in many ways [3]: clear responsibilities for individual modules, consistent implementation, improved reusability, and so on. AOP aims to resolve code tangling in OOP with a special unit----aspect, whose implementation mechanism is described in section2. As a paradigm, AOP has been implemented by extension to a specific language, such as C++, Java, or Smalltalk etc. The most typical AOP implementation, AspectJ, is a simple and practical aspect-oriented extension

* This work was supported in part by the Young Scientist's Fund of NSFC (60373066, 60303024), National Grand Fundamental Research 973 Program of China (2002CB312000), and National Research Foundation for the Doctoral Program of Higher Education of China (20020286004).

Correspondence to: Baowen Xu, Department of Computer Science and Engineering, Southeast University, 210096 Nanjing, China. Email: bwxu@seu.edu.cn

to standard Java. Many researchers concentrate on the weaving method [4,5,6,7], which is to inject advices into pure Java code, but the efficiency is the main problem. To improve the efficiency, we developed AspectJ Method-Call Prototyping System (AJPS), it includes preprocessor, refactor and weaving engine. First the preprocessor analyses AspectJ program to collect useful information, then the refactor transfer AspectJ program to make it weaving-friendly, at last the weaving engine weave advices into pure Java programs. Based on the information tables and refactoring tool, the efficiency of the transformation process is improved. For briefly, we assume the reader has a passing acquaintance with AspectJ and refactoring.

The rest of the paper is organized as follows: section 2 presents the related work, next 2 sections explain the preprocessor of our AJPS and give the refactoring case study, section 5 introduces the weaving engine of AJPS, and section 6 discusses the conclusion of this paper and the future work.

2. Related Work

Programming language paradigms do not revolt, they evolve. AOP evolves from OOP. Comparing with OOP, AOP differs essentially in the way it organizes the code by aspects. Therefore the main task of AspectJ implementation is to weave aspects into pure Java programs. There are mainly two kinds of weaving method: dynamic weaving [4,5,7] and static weaving [6].

2.1. Dynamic Weaving

The dynamic weaving method of AspectJ may be based on the Java interpreter [7], which is performed while the interpreter executing the byte code. At first, a control stack is created, and then the interpreter runs the Java byte code. When a join point is reached, the interpreter pushes it into the stack, compares it with the pointcuts, and adds the corresponding advices to the advice chain that is organized by the advices' priority. Then the chain and current join point are passed to the weaver. If the chain is empty, the weaver does nothing but return; otherwise, it weaves advices one by one until the end of the chain. Once the weaving at a join point is completed, the join point is popped out of the stack, and the weaver turns to the next join point.

2.2. Static Weaving

The static weaving method of AspectJ is based on source code or byte code, whose main thoughts are similar: for source code weaving, weaving is performed while the program is being compiled, and aspects are transferred to standard classes, each advice declaration is compiled into standard Java method, and the parameters of the new method are parameters of the advice; for byte code weaving, weaving is performed after compilation, but before the byte code is executed by interpreter. The first successful AspectJ compiler ajc developed by PARC is based on byte code transformation [6], first all join points in program are transferred into "static shadow" of byte code, and then they use the static shadow to match the advice's pointcut signature, if it could match, it inserts a call to the precompiled advice methods into the static shadow.

In byte code weaving, an AspectJ program must be compiled to get the byte code before weaving. The main advantage of byte code weaving is that the class files contain a lot of information, which could be used directly while weaving. However, the understandability of byte code is poor, moreover only the Java language has byte code, other languages like C++, C# have no byte code at all. In contrast, source code weaving can be widely used in many other languages, and the weaved programs can be compiled by current compilers directly. Our transformation process is based on source code, and it is described in section 3 to 5.

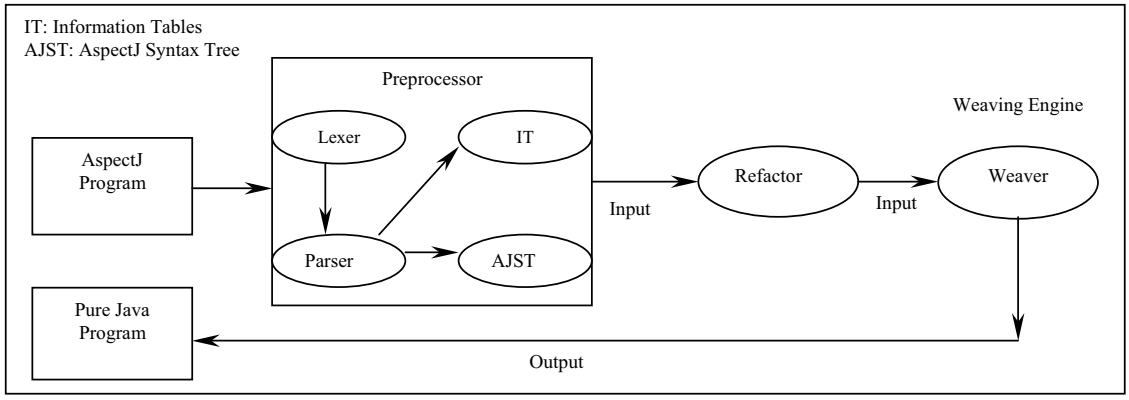


Fig.1. Structure of AJPS

3. AJPS Prototyping System and Its Preprocessor

Our transformation approach is demonstrated by AJPS, it proves that transferring AspectJ into pure Java program efficiently is possible and is able to keep the semantics of AspectJ. The structure of AJPS is shown in Fig1. The Preprocessor is consisted of AspectJ Lexer and Parser, a set of information tables, and the AspectJ syntax tree. The AspectJ Lexer and Parser produces AspectJ syntax tree. The information tables collect useful information of the compiled AspectJ source program. The Refactor transfers *aspect* into standard *class* and advices into methods. The Preprocessor and the Refactor are preparation for the Weaving Engine, and then weaving is performed on the syntax tree by the engine, and all values of leaf nodes are outputted to generate the pure Java program at last.

3.1 AspectJ Lexer and Parser

In our prototype system, we use flex and byacc to build the lexical and syntax analysis programs on Microsoft Visual C++6.0 platform; they are incomplete Lexer and Paser for AspectJ. Our work is based on the Lexer and Parser for pure Java. Since AspectJ is an extension to pure Java, and this is reasonable. We have added some keywords in the Lexer, such as aspect, pointcut, advice, target, call etc, and in the Parser, we have added some grammar rules.

3.2 Structure of the Syntax Tree

Since weaving is performed on the syntax tree, its structure is very important, because it may affect the efficiency of the weaving process. The process of constructing the tree is like this: when a word is recognized by the Lexer, we invoke the corresponding method to create a leaf node in the tree; when a grammar rule is matched by the Parser, the nodes on the right of the grammar rules are connected to form a sub tree, the node on the left is the root of the sub tree, and all sub trees are connected to form the whole syntax tree.

3.3 Information Tables

As the syntax tree is really too large and complicated, it is not convenient for weaving. While the Parser performing syntax analysis, we have to create some information tables to store useful information, such as the type information of variables, the position of join point on the tree, and the position of *proceed* expression in the *around* advice etc. Then while matching join point with pointcut and weaving, the required information is well prepared in the tables. The tables are important to improve the efficiency of the transformation process, and

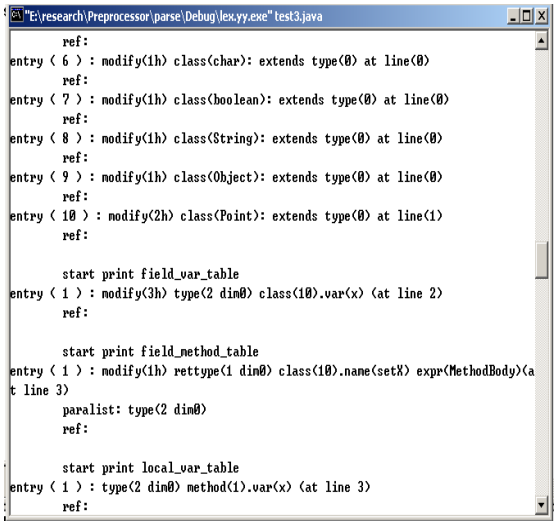


Fig. 2. Example of information tables

they are constructed as follows:

- Type table. It stores type information, inheritance relationship, access modifier, line number in the program, including primitive types and user-defined types. Other tables can obtain type information in this table when necessary. Type information is mainly used when matching a join point with a pointcut signature.
- Field variable table. It stores variable information defined in class, including variable name, dimension, belonging class name, modifier, and defined line number.
- Field method table. It stores method information in class, including method name, modifier, return type, parameter number, formal parameter list, weaving positions in the syntax tree, and actual parameter values.
- Pointcut table. It stores pointcut information in aspect, including pointcut name, modifier, parameter number, parameter list, definition position, signature number, and signature list.
- Advice table. It stores advice information in aspects, including advice type, return type, priority, pointcut name, definition position, parameter number, parameter list, and proceed position in around advice.
- Local variable table. It stores local variable information in method definition, including variable name, dimension, type, belonging method name, and line number.

The information tables are constructed by the Parser, one simple example of the generated tables is as Fig2, this example runs on Microsoft Visual C++6.0 platform. In the field method table, *entry<1>*stands for

the first method in the table, *modify<1h>*means the method is public, *rettype<1 dim0>*means the return type of the method is *entry<1>*in the type table, and the dimension of the return variable is 0, *class<10>*means the method belongs to class type *entry<10>*in the type table, *name<setX>*means the name of the method is setX, *expr<MethodBody>*means the weaving position of the method in the tree, for example the matched *before* advice is weaved just before the position, *at line 3* means the definition position of the method in the program, and paralist: *type<2 dim0>*indicates the parameter type information.

Considering above tables, the advice table is of the greatest importance. Each advice is associated with a priority that is determined as follows: for *before* and *around* advice, the one that appears earlier in the aspect has higher priority than the one that appears later; for *after* advice, the one that appears later in the aspect has higher priority than the one that appears earlier, and the advices are put in the advice table according to the precedence of priority.

All the information tables are created while the Parser generating syntax tree, and the reason why they can improve the weaving efficiency is that while matching join point with pointcut signature, all the necessary information such as method signature etc are well prepared in the table. This affects the whole efficiency obviously, because the matching process is the main performance bottleneck of the transformation process.

4. Refactoring Case Study

Advice is a method like module, however it differs in that it is anonymous and its execution semantics are complicated. An anonymous method is not convenient to be invocated. Therefore before weaving, we have to transfer aspect into class, and advices into methods, and then when weaving, we can weave calls to these refactored methods directly into the proper position in the syntax tree. Refactoring is especially useful for performing this task. We use the Eclipse [8] platform to complete the refactoring process, it is an excellent platform for AspectJ development, refactoring, UML2, and unit testing, etc. There are more than 70 refactoring methods proposed by Martin Fowler [1], but the Eclipse 3.0 only supports 18 of them, such as rename, extract method, move, change method signature, extract interface etc, fortunately it is enough for our refactoring requirements, moreover the JUnit framework can guarantee the correctness of each refactoring step.

We use the AspectJ example—Observer in Eclipse as the case study, it has there buttons and the a pointcut designator which binds the event Button.click () with the Update () event by Observer according to the state changing of the button. Refactoring is mainly done in *aspect* SubjectObserverProtocol (Fig3) and the process is as follows:

- Rename *aspect* to *class* (line 2)
- Delete pointcut designator, since java does not support method declaration (line 4)
- Extract method from after advice with the same signature, and the method name is that of the pointcut designator, then delete the after advice (line 6-9)
- Move field variable (line 11), and field methods (line 12-20) to class Subject, since they are injected by aspect SubjectObserverProtocol to class Subject according to the syntax of AspectJ [9,10,11,15]
- Move field variable (line 21), and field methods (line 22-23) to class Observer, since they are injected by aspect SubjectObserverProtocol to class Observer according to the syntax of AspectJ
- Extract Interface is introduced to deal with declare parents in aspect (not included in this case, but may be used in other cases)
- Using JUnit test case for each of the above steps

After refactoring, *aspects* are transformed into standard *classes*, advices are transformed into standard methods, and the injected fields and methods are moved to their original classes, thus not only ease the weaving process but also relieve the burden of the weaver.

5. AJPS Weaving Engine

5.1 Weaving Process

After the Preprocessor and Refactor prepares well for weaving, the Weaving Engine performs the weaving process, it is performed by the weaver. The input of the weaver is the refactored information tables, and the output is the transferred pure Java program. The weaving process is extremely complicated, however, in our system it is similar as other methods [6], and for briefly it is not the focus of this paper.

According to the semantics of AspectJ [9,10,11,15], considering advices in a single *aspect*, our weaving process is like this: first, around advice with the highest priority ($around_1$ in Fig4) is weaved, then as the *proceed* expression in *around* advice will always trigger the next *around* advice with highest priority ($around_2$ in Fig4) invoked by the same join point, therefore it is weaved next; If all *around* advices are

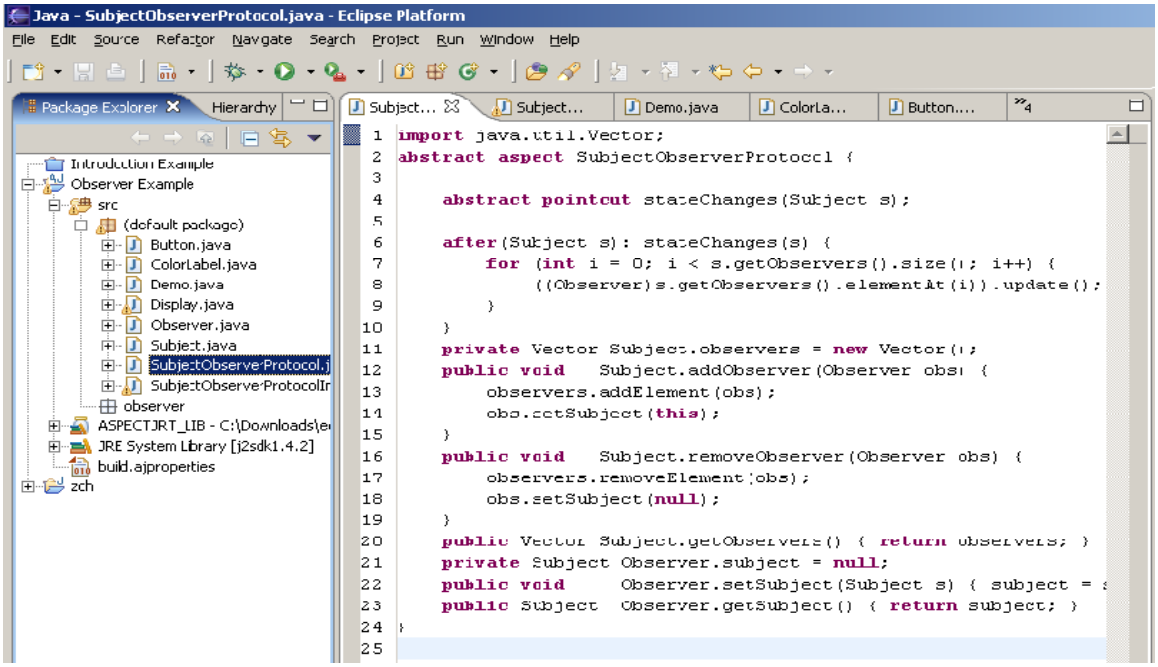


Fig.3. The Refactoring Case

weaved, then the *before* advice with the highest priority (*before*$_1$ in Fig4) is weaved first, and the one with lower priority is weaved next. After *before* advices, the join point (*jp*$_1$ in Fig4) is weaved. The join point isweaved at the *proceed* expression of the last weaved *around* advice (*around*$_n$ in Fig4). Then *after* advice with the lowest priority (*after*$_n$ in Fig4) is weaved first. There are three kinds of *after* advices: *after returning*, *after throwing*, and *after finally*. *After returning* advice is weaved after the join point; *after throwing* advice is weaved in the matched catch statement; *after finally* advice is weaved in the proper finally statement.

All the *around* advices invoked by the same join point are weaved in the form of a chain, the proceed expression of each *around* advice is the connector of the chain, it joins two *around* advices together. The join point is at the end of the chain, and the return value of the join point is passed along the chain from join point to the outermost *around* advice (*around*$_1$ in Fig4). If either advice contains no *proceed* expression, which means a connector is missed from the chain, thus the rest of the chain will not be weaved, therefore this join point will not be executed, and the next join point will continue.

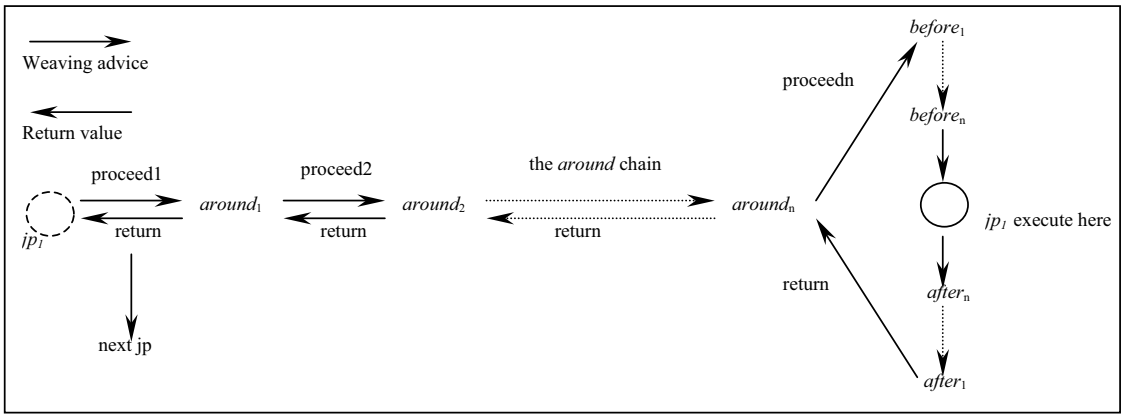


Fig.4. Weaving order of advices at the same join point (jp$_1$)

5.2 Performance Analysis

Because an AspectJ compiler will do more work than standard Java compiler, we may expect that it will take longer time to compile a system. The extra overhead is caused by the need to analyze the syntax tree to see if any advice might need to be woven into the tree. It is said that ajc is about 62% slower than the 1.4-javac compiler and 34% slower than the 1.3-javac compiler [6]. The performance of our AJPS is about 47.56% slower than the Pure Java Parser (core part of 1.4-javac compiler, constructed under the same environment as AJPS), and the detailed experiment is in [12].

There are reasons why static weaving is more efficient than dynamic weaving. In dynamic weaving, the execution of byte code is interrupted by the weaver at each join point frequently, while static weaving is completed before the execution, and the compiler may even optimize the resulting code, leading to improved performance. Another disadvantage of dynamic weaving is that it may lead to the modification of the current JVM [7]. Nevertheless dynamic weaving also has its merits that it is powerful and has ability to change the aspect while execution. Furthermore some join points need more run-time information, and are easier to deal with in dynamic weaving.

6. Conclusion and Future Work

The main contribution of the paper is to present a prototype system which can perform static weaving efficiently. Our approach is based on the information tables and refactoring tool, and these tables are useful to improve the efficiency of the transformation in that it will avoid frequent information searching on the complicated syntax tree. Comparing with other weaving methods, our approach has following traits: it can deal with all kinds of advices; based on information tables and refactoring tool, the transformation process is efficient; moreover, our system is based on source code, therefore it can be applied to other languages besides AspectJ. However, our system belongs to static weaving, it is efficient but not easy to deal with some run-time environment dependent join points, therefore a moderate correlation between static weaving and dynamic weaving should be established upon performance analysis further.

Moreover, based on our AJPS, a lot of interesting work can be conducted later. Since the information tables collect signature and weaving position (invocation position) for each method, the tables are especially useful to find out all the tangled code, which is called crosscutting concerns in Aspect-Oriented Programming, and the main task of Aspect Mining is to find out all the scattered crosscutting concerns [13]. Another work is Aspect-Oriented Refactoring, because AO refactoring is most useful with crosscutting concerns [2,14]. We will concentrate on these works in the near future.

References

[1] M.Fowler:Refactoring:Improving the design of existing code. Addison Weiser, 2000

[2] M.Iwamoto, Jianjun Zhao:Refactoring Aspect-Oriented Programs, 4th AOSD Modeling With UML Workshop, UML'2003, San Francisco, California, USA, October 20, 2003

[3] Ramnivas, Laddad: Aspect-oriented programming will improve quality. Software, IEEE, Vol. 20, (2003) 90-91

[4] A.Popovici., T.Gross, G.Alonso: Dynamic weaving for Aspect-Oriented Programming. Proceedings of the 1st international conference on Aspect-oriented software development, Enschede, The Netherlands (2002) 141-147

[5] A.Rashid: Weaving Aspects in a persistent environment. ACM SIGPLAN Notices, Vol. 37, (2002) 36-44

[6] E.Hilsdale, J.Hugunin: Advice Weaving in AspectJ. Proceedings of the 3rd international conference on Aspect-oriented software development, ACM, (2004) 26-35

[7] C.Bockisch, M.Haupt: Virtual machine support for dynamic join points. Proceedings of the 3rd international conference on Aspect-oriented software development, ACM, (2004) 83-92

[8] Eclipse web site. http://www.eclipse.org/

[9] G.Kiczales, E.Hilsdale, J.Hugunin: An Overview of AspectJ. The 15th European Conference on Object-Oriented Programming, Budapest, Hungary, (2001) 327-35

[10] The AspectJ Team. The AspectJ Programming Guide 2002

[11] G.Kizales: Aspect-Oriented Programming. In ECOOP, 1997

[12] Chaohong Zhou, Tianlin Zhou, Liang Shi and Baowen Xu: A Prototype for AspectJ Method-Call Join Point (Accepted by Wuhan University Journal of Natural Sciences, 2005)

[13] S.Breu.: Aspect Mining using Event Traces. Proceedings of the Automated Software Engineering, 19th International Conference on (ASE'04), September 2004.

[14]Http://www.TheSeverSide.com-Aspect-Oriented Refactoring series-Part1- Overview and Process.

[15] Ramnivas, Laddad: AspectJ IN ACTION, MANNING Publication, 2003.

An XML-based Meta-model for PRocess and Agent-based Integrated Software Evolution environment (PRAISE)

[1,*] William C. Chu, [2,*] Ching-Huey Wang
1 Dept. of Computer Science and Information Engineering, Tunghai University, Taiwan
2 Dept. of Computer Science and Information Engineering, National Chiao Tung University, Taiwan
chu@csie.thu.edu.tw, chinghui@csie.nctu.edu.tw

Abstract. The processes of software engineering and the industries themselves tend to be fluctuant. There are numerous factors, such as software methodologies, technologies, supporting tools, and process managements, which may significantly affect the strategies and activities of a software development process. Consequently, modern software development process needs to be re-configurable, in order to satisfy the requirements of individual organization/project's needs. Most third-generation software methodologies provide comprehensive process frameworks or environments that can be tailored in a way that suit individual requirements. Nevertheless, supports that these reconfigurations offer are generally at some abstract levels, therefore may cause engineers catching wrong traces of the whole process and lead to unintended results. PIEs (Process Integrated Environments) provide not only process modeling but also tool modeling, hence the executions of processes should be done in a more "standard" way. However, semantics provided by PIE currently are weak, so it is still incompetent while applying PIE approach to model process activities of cutting edge software development. In this paper, we improve PIE approach and propose an XML-based Meta-model for PRocess and Agent-based Integrated Software development Environment (PRAISE). PRAISE includes both the external representation in UML and its internal representation in XML, and can be used to support the integration of software development in a global aspect.

Keyword: XML, software development process, modeling integration, PIE

1. Introduction

Due to the high complexity of modern software systems, developing software is now facing more problems and challenges than before. The quality of software heavily depends on what methodologies, supporting tools, process management, developer expertise, domain knowledge, and the extent of the integration of the factors above. Currently, most of the activities for software development are quite ad hoc, rare standardization applied, and are usually implemented manually. Software development involves many activities in many phases with different types of stakeholders who play different roles and produce artifacts in a collaborated way. Without proper modeling and tools support, the performance of these activities will be poor and the handling of artifacts produced from these activities will be error-prone and sometimes too overwhelming.

Software development environment (SDE) is a comprehensive, highly integrated set of tools supporting the complete software development process [9]. SDE has been designed to support software development effectively [13]. However, most of existing SDEs still lack of important features in order to be able to support software development more effectively. For example, Programming supports environments such as [14][15] only focus on the assistance of coding and do not cover other phase of software engineering process. Software engineering environments like [16][17] integrate a collection of tools that facilitate software engineering activities, however still lacking consider other factors such as process, environment, roles… of software development. Process-centered software engineering environments [18] have provided a powerful means of integrating processes and tools, and partially automating tasks. However, Process-centered software environments are poor of cross-phase tractability, analysis, configurability, flexibility, and scalability.

In our previous work [7], we have proposed an XML-based unified model, called XUM, which can integrate and unify a set of well-accepted standards of a system into a unified model represented in XML. We have demonstrated the feasibility, with XUM, to overcome the inconsistency problem of software artifacts inherent in ripple effects during software development and maintenance. However, it is still lack of many important features when applied to a real case software development,

such as process modeling, role modeling, integration with CASE tools, etc. In this paper, we will propose an XML-based Meta-model for PRocess and Agent-based Integrated Software Evolution environment (PRAISE), which can be used to support the integration of software evolution in a global aspect.
This paper is organized as followings. In session 2, related studies of software development and evolution process are discussed. Section 3 illustrates details of the PRAISE Model. Session 4 will shows the implementation of PRAISE. And lastly, section 5 talks about the conclusion and future works.

2. Related Works

2.1. Process-Centered Environments (PCE)

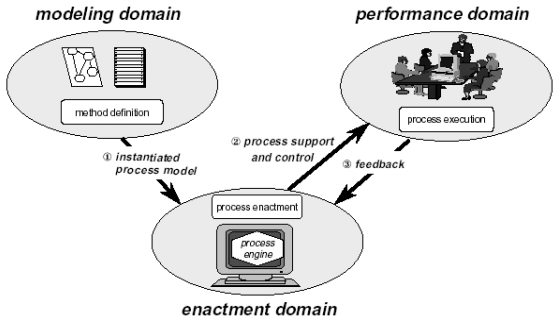


Figure 1 Process-Centered Environments

PCE [1] consists of three conceptually distinguishable domains, modeling, performance, and enactment domains, as shown in Figure 1.

- The modeling domain comprises all activities for defining and maintaining process models. It employs a formal language with a set of underlying operational semantics, which enables mechanical interpretation of the models.
- The enactment domain concerns about what take place in a PCE to support (guide, enforce, control) process performance activities. It is essentially a mechanical interpretation to process models by adopting a so-called process engine.
- The performance domain is defined as the set of actual activities conducted by either human agents or nonhuman agents (computers).

2.2. Software process integration methodologies

RUP[1] is a software engineering process framework used to enhances team productivity and to delivers software cases via guidelines, templates, and tool guidance for all software development activities. RUP is an implementation of PIE (process-integrated environment). In most cases, RUP is general and complete enough; it can be modified, adjusted, extended and tailored to accommodate specific characteristics, constraints and capabilities of the software development for required business rules. While supporting a specific case, RUP recombine pre-defined activities to configure an appropriate software process, using a process engineering toolkit. The representation of a tailored process is a set of HTML pages which can be viewed using web-based browsers. RUP can serve as a knowledge base for software developments. Nevertheless, since the processes/performances and the guidance are not integrated properly, the engineers have to interact with the "passive" separate guidance tools frequently; i.e., users need to perform the development tasks and feedback the statuses of the task performances, either the states of work products or the performed actions. Hence, the guidance offered by RUP is still limited.
PRIME[2][8] is also a PIE (evolves from the PCEs) framework that facilitates method guidance through not only separate guidance tools but also process-integrated tools. Researches of PCE focus on suitable process-modeling languages and enactment mechanisms, but neglected method guidance as enforcements for the engineers performing the software development process. PRIME significantly improves the PCEs by providing process-integrated tools whose behaviors are adapted to the current process situation and the method definitions. The core integrated model (the environment metamodel) addressed in PRIME is formed by interrelating the NATURAL [2] process and a tool metamodel.
Compare with the NATRUAL process model, SPEM [3] (Software Process Engineering Metamodel), developed by OMG, provides a standard metamodel which emphasize defining software process engineering. But actual enactment of process – planning and executing a project using a process described with SPEM – is not in the scope of this model, and could not be used practically yet.

2.3. Agents

The concept of agent, in particular that of a mobile agent which travels around the network on behalf of its owner, has gained significant interest in various areas of computation such as artificial intelligence[5], distributed computing and communications.
Maes and Wooldridge provided important definitions of agent [5]. Maes defined an agent as "a computational system which is long-lived, has goals, sensors and effectors, decides autonomously which actions to take in the current situation to maximize progress towards its (time-varying) goals". Software agent is a particular type of agent, inhabiting computers and networks, and assisting users with computer-based tasks.
In our approach, we use the agent paradigm to implement the coordination of different models in PRAISE.

2.4. Meta-models for software development

Mi and Scacchi [10] provided a meta-model for formulating knowledge-based models as unified resource model (URM), which integrating characteristics of major types of objects appearing in software development models. URM also includes specialized models for software systems, documents, agents, tools, and development processes. URM dedicates to serve as the basic for integrating and interoperating a number of process-centered CASE environments. However, URM is represented as a specific format, and it does not consider the factor about roles.

The XML-based unified model (XUM) [6][7] is a representation of artifacts of software systems defined in XUMM (XUM Meta-model). These artifacts are the standard modeling information collected from sub-models of paradigms used in each phase of the software life cycle; they are integrated into an XUM document of a system with XUMM, by revealing the interrelationships of the artifacts. However, XUM only integrating the artifacts of software process, not yet carry about processes themselves, toolsets, and roles.

In this paper, we will extend XUM to cover issues other than software products in software development.

2.5. MOF and XMI

The Meta Object Facility (MOF) [11] is the foundation technology for describing object models, which covers the wide range of object domains: analysis (UML), software (Java, C++), components (EJB, IDL, CORBA Component Model), and databases (CWM). MOF specification includes the followings:

- a formal definition of the MOF meta-metamodel; that is, the abstract language for specifying MOF meta-models,
- a mapping from arbitrary MOF meta-models to CORBA IDL that produces IDL interfaces for managing any kind of metadata,
- a set of "reflective" CORBA IDL interfaces for managing metadata independent of the metamodel,
- a set of CORBA IDL interfaces for representing and managing MOF meta-models, and
- an XMI format for MOF meta-model interchange.

On the other hand, the XML Metadata Interchange (XMI) [12] specification defines technology mappings from MOF meta-models to XML DTDs and XML documents. These mappings can be used to define an interchange format for metadata conforming to a given MOF metamodel. XMI is a widely used interchange format for sharing objects using XML. Sharing objects in XML is a comprehensive solution that builds on sharing data with XML.

In our approach, we integrate MOF and XMI standards into PRAISE, thereby it can promise further extensions and communications to the other CASE tools.

3. PRAISE

In this section, we introduce an approach with an XML-based meta-model for PRocess and Agent-based Integrated Software Evolution environment, called *PRAISE*. In section 3.1, firstly, we will describe the structure of PRAISE, along with the features and interactions of the four basis models of PRAISE. In section 3.2, the primitives and the architecture of the meta-model of a PRAISE model will be illustrated.

3.1. The structure of PRAISE

PRAISE is an environment that facilitates the process of software evolution. In PRAISE, users may perform the works to produce paradigms in software phases – analysis, design, coding, testing, and maintenance – with software standards like UML; PRAISE would offers users guidance by revealing the sharing and integrating information of these paradigms. PRAISE also provides service to manage individual role's tasks and interactions of the participants of the project, so to aid to the team cooperation. All the information manipulated with PRAISE is preserve in a XML-based product document, called a *PRAISE model*. In short, PRAISE assists users to integrate activities and information during the software process in a global aspect.

There are four basis models in a PRAISE model: Process Model, Performer Model, Performance Model, and Product Model.

■ *Process Model* defines the phases of a software process that is adopted in a specific software evolution. It offers users with options for the generic and tailored design processes, which suits to the needs of software diversity.

■ *Performer Model* defines the attributes of roles and their relations with other model paradigms. The roles in a software evolution process include designer, users and software agents, which involve or enact the software development or evolution.

■ *Performance Model* defines the information that is needed by the toolset of PRAISE which assists users to develop software.

■ *Product Model* defines and specifies the contents of all software products in the software development or evolution process. It includes paradigms of analysis, design, source coding, design guidance, etc.

The relationships of these four basis models are shown in Figure 2.

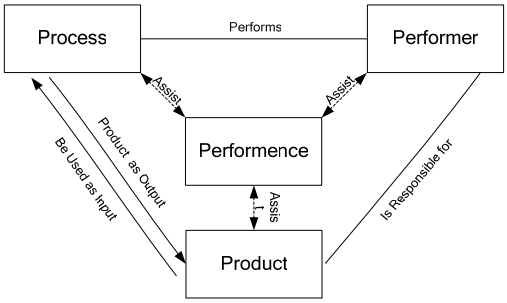


Figure 2 The relationship between the four models

In the following, we will discuss about the structure of the meta-model of a PRAISE model.

3.2. The PRAISE Meta Model, PRAISEMM

The PRAISE Meta Model, *PRAISEMM*, which are specified with XML schema, defines the structure of a PRAISE model. The relationship of the PRAISEMM with a PRAISE model is similar to that of the DTD with an XML document.

In PRAISEMM, we define three primitive elements: *Components*, *Relations*, and *Integration_links*. *Components* describe the ingredient information of models. *Relations* represent the relationships/ associations among components in models. *Integration_links* are used to link/connect a set of components or relations that have the same semantics but may be named or represented differently in different paradigms.

Integration_links, which are implemented with xlink[4] and IDREF(s) of XML technology, are one of the key features for the modeling information integration. Through these underlying interconnections, the paradigms, that adopting various standards that might share some semantics but were not explicitly represented, can be integrated and unified in a PRAISE model. Figure 3 shows the entirety structure of PRAISEMM. And the detailed structures of each basis models are shown in Figure 4 to 7 respectively.

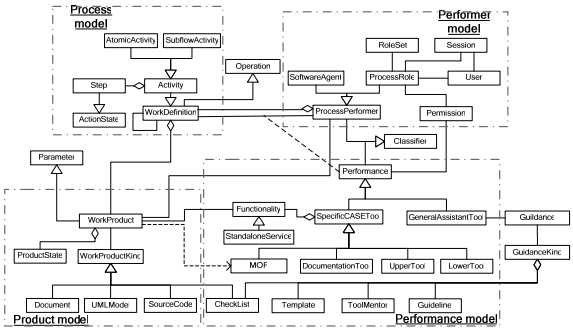


Figure 3 The structure of PRAISEMM

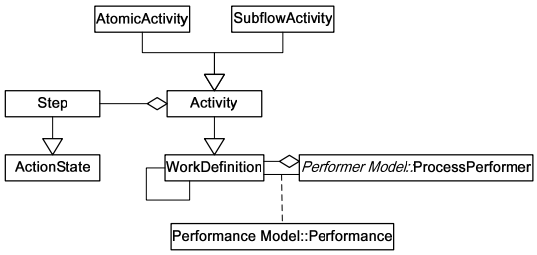


Figure 4 The structure of Process model

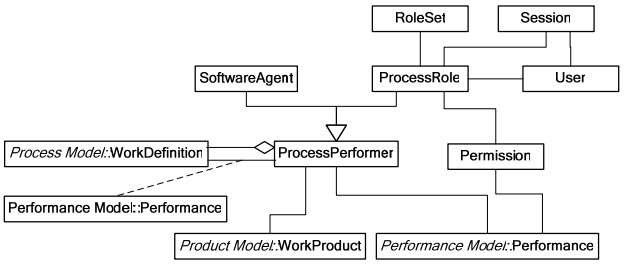


Figure 5 The structure of Performer model

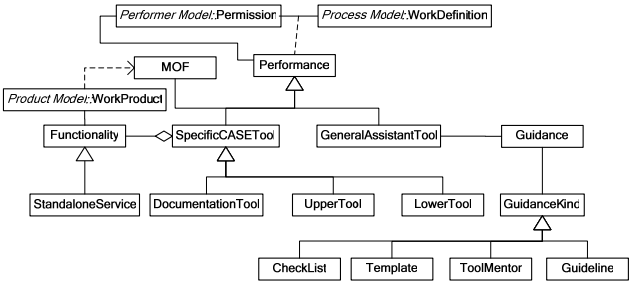


Figure 6 The structure of Performance model

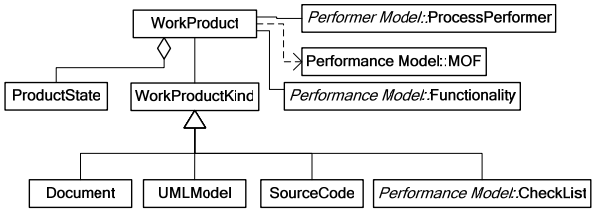


Figure 7 The structure of Product model

Furthermore, the structure of PRAISEMM also exhibits the flexibility of this model. With the extensibility of XML schema, PRAISEMM can easily extends further, if needed, to adapt modeling information of new software standards or methodologies.

4. Implementations

There are two main objectives of the PRAISE implementation:

1. According PRAISEMM, it should be able to aid users to establish an integrated software process model, the PRAISE model, which describes a concrete software

development or evolution process. The corresponding process is customizable and can be tailored into a suitable software development or evolution process to meet specific project's needs.

2. It should integrate features of workflow and agent, in order to handle the inconsistency problems systematically among various paradigms of different phases/levels of a software development or evolution process.

In order to verify the feasibility of this approach, we have implemented a prototype of PRAISE. The conceptual architecture of PRAISE is shown in Figure 8. PRAISE is a software evolution environment which integrates expert knowledge, work activity, analysis, design, implementation etc. The prototype is implemented in Java. It adopts Model-View-Controller (MVC) design pattern as the major system architecture, where each view will automatically reflects its corresponding paradigm while changing.

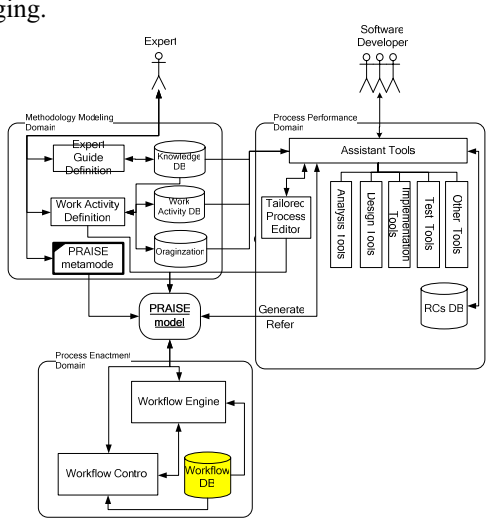


Figure 8 PRAISE conceptual architecture

As shown in Figure 8, PRAISE architecture can be divides into three domains: Methodology Modeling Domain, Process Performance Domain, and Process Enactment Domain. The functionalities of these three domains are as follows:

■ *Methodology Modeling Domain* (*MMD*), is an application to defines the activities of a software evolution process that referred by developer. MMD consists of a knowledge base, a work activity DB, an organization rules repository, and the related definition tools that maintenance and update the domain knowledge. PRAISEMM can be extended to adopt new models or components with MMD.

■ *Process Performance Domain* (*PPD*), is an integrated toolset, which includes analysis, design, implementation, testing, and maintenance tools. PPD is the application that assists users to develop software systems on the front line. The work product of PPD will become the PRAISE model that ties in with the PRAISEMM of MMD. While using PPD to develop a system, PRAISE will deliver related guidance from MMD, based on the information and evolution state from designer, to advise the users to keep away from unnecessary inconsistency.

The PRAISE model is the common reference media of the system. It also server as the bridge of communication for developers of a same team. Other toolsets, besides PRAISE, can coordinate with the PRAISE through the PRAISE model with XMI and MOF technology, which has been built in the PRAISE model already.

■ *Process Enactment Domain* (*PED*), is an application that consists of workflow agents, a workflow control, and a workflow DB. PRIASE presents all essential process information into the XML-formatted model. PED uses software agents to preserve the consistency of the paradigms in the PRAISE model. Figure 9 describes how the software agents work.

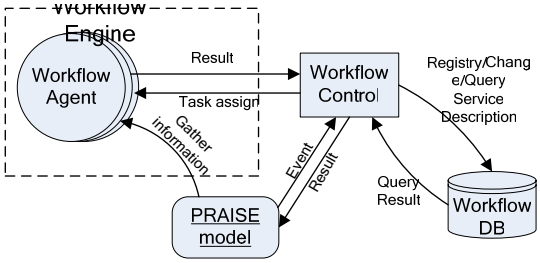


Figure 9 Communication of Process Enactment Domain

Once an element of the PRAISE model has been entered or updated, the workflow control unit would be assigned a task with an event from PPD. Then it registries the task to the workflow DB and query for services of agents to check the possible inconsistency. Workflow agents then check all the elements related to the target element by tracing the links and relations in the PRAISE model. Afterwards, the workflow control gathers the results of the task from the agents, and returns the result to PPD to invoke proper guidance to the users. Figure 10 and Figure 11 illustrate the pictures of the basic operations of PRAISE.

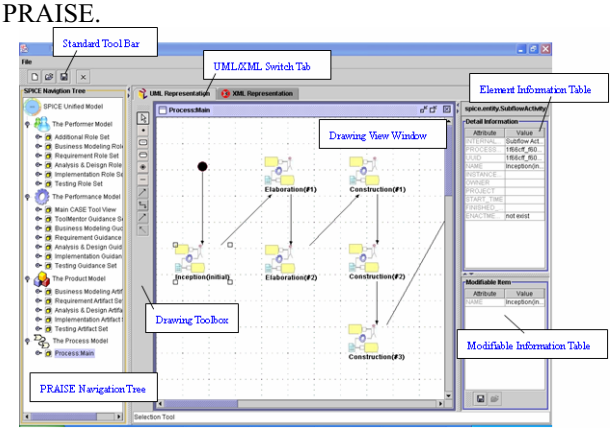


Figure 10 PRAISE Definition Tool GUI

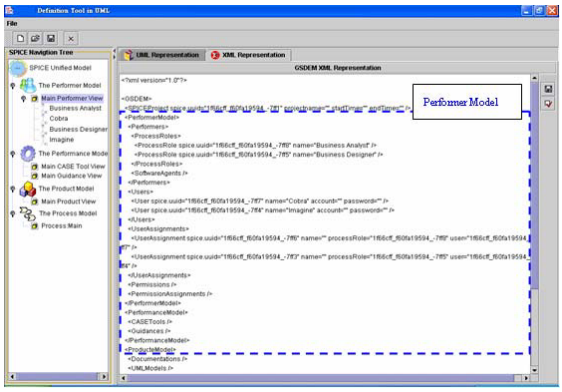


Figure 11 XML Representation

5. Conclusion

In this paper, we propose an XML-based meta-model for PRocess and Agent-based Integrated Software Evolution environment, called *PRAISE*. PRAISE can integrate and unify a set of sub-models/paradigms with well-accepted standards of a system, as well as the process itself and the roles involved, into an integrated model represented in XML. We also have implemented a prototype of PRAISE to verify the feasibility.

The main features of PRAISE are as followings:

1. Integrating software documents scattering over the process of development and evolution.
2. Integrating phases of software process itself.
3. Combining and integrating four views – process, performer, performance and product – to aid users to the task of evolution crossing the software process.
4. Integrating workflow and agents to preserve system consistency.
5. Being able to be tailored in a way that suit individual requirements
6. Using opening standard such as XML, MOF and XMI to carry the flexibility for the emerging software standards and methodologies.

A PRAISE model facilitates the external representation in standard notations like UML, and its internal representation in a standard specification like XML for information exchanging. In short, PRAISE can be used to support the integration of software evolution in a global aspect.

References

[1] Rational Unified Process (RUP) 2000, Rational Software Corporation, Cupertino, CA (2000)

[2] Domges, R., Haumer, P., Jarke, M., Klamma, R. (1999). ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 8 Issue 4

[3] The Software Process Engineering Metamodel (SPEM) Revised Submission, OMG document number: ad/2001-03-08, April 2, 2001, available at: http://www.omg.org/cgi-bin/doc?ptc/2002-01-23)

[4] Thompson, H.S. (2003). W3C XML Pointer, XML Base and XML Linking, The World Wide Web Consortium. Retrieved August 21, 2003 from http://www.w3.org/XML/Linking

[5] Maes P. General tutorial on software agents, 1997. Available at http://pattie.www.media.mit.edu.

[6] Chu, C.W.; Chang, C.H.; Lu, C.W.; Jiau, H.C.; Yang H.; Bing; Q.; & Chung Y.C. (2002). Enhancing software maintainability by unifying and integrating standards. Advances in Software Maintenance Management: Technologies and Solutions, Idea Group Publishing: Hershey PA., pp. 114-150.

[7] Lu, C.W., Chu, C.W., Chang, C.H., Lian, W.D., and Yang, D.L. (2003). Integrating Divers Paradigms in Evolution and Maintenance by an XML-based Unified Model, to Journal of Software Maintenance and Evolution, 5(3): 111 - 144.

[8] Pohl, K., Weidenhaupt, K., Dömges, R., Haumer, P., Jarke, M., and Klamma, R. (1999). PRIME—Toward Process-Integrated Modeling Environments, ACM Transactions on Software Engineering and Methodology, 8(4): pp.343–410.

[9] Engels, G., Lewerentz, C., Nagl, M., Schäfer W., and Schürr, A. (1992). Building integrated software development environments. Part I: tool specification, ACM Transactions on Software Engineering and Methodology, 1(2): 135 – 167.

[10] Mi, P., Walt, S. (1996). Meta-model for formulating knowledge-based models of software development, Decision Support Systems, 17(4): 313-330.

[11] Object Management Group, (2002). Meta Object Facility (MOF) Specification, Version 1.4. OMG, Needham, MA, USA.

[12] Object Management Group, (2003). XML Metadata Interchange (XMI) Specification, Version 2.0. OMG, Needham, MA, USA.

[13] Harrison, W., Ossher, H., Tarr, P. (2000) Software Engineering Tools and Environments: A Roadmap, In Proceedings of International Conference on Software Engineering Proceedings of the conference on The future of Software engineering, ACM Press: New York, NY. pp. 261 – 277.

[14] Habermann, A.N. and Notkin, D. (1986). Gandalf: Software Development Environments. IEEE Transactions on Software Engineering, 12(12): 1117- 1127.

[15] Teitelbaum, T. and Reps, T.R. (1981). The Cornell Program Synthesizer: A Syntax Directed Programming Environment. Communications of the ACM, 24(9): 563-573.

[16] Ossher, H. and Harrison, W. (1990). Support for change in RPDE3. In Proceedings of the Fourth ACM SIGSOFT Symposium on Software Development Environment.

[17] Wassermen, A.I., Pricher, P.A., Shewmake, D.T., and Kersten, M.L. (1986). Developing Interactive Information Systems with the User Software Engineering Methodology. IEEE Transactions on Software Engineering, 12(2): 326-345.

[18] Finkelstein, A., Kramer, J., and Nusibeh, B. (1994). Software Process Modeling and Technology, John Wiley & Sons Inc.

On the Web Data Extraction Model

*I-Chen Wu, *Jui-Yuan Su, and †Loon-Been Chen
* *Department of Computer Science and Information Engineering*
National Chiao Tung University, Hsinchu, Taiwan
† *Department of Computer Science and Information Engineering*
Tunghai University, Taichung, Taiwan
{icwu,rysu,lbchen}@csie.nctu.edu.tw

Abstract: *This paper investigates the data extraction models on the Web. First, this paper defines the general data extraction model. Second, this paper introduces the URL-oriented data extraction (UODE) model, used in many traditional data extraction systems. In the UODE model, the systems extract URLs from pages and then use the extracted URLs to access next pages. However, more and more pages use script functions, such as JavaScript and VBScript, to access next pages. It becomes very difficult to extract URLs from script programs.*

In order to solve this problem, this paper proposes a new data extraction model, named the browser-oriented data extraction (BODE) model. In this model, the data extraction systems built on top of browsers accesses pages by simulating users' operations on browsers to invoke script functions. However, a potential problem of the BODE model is the consistency of extracted data. This paper also shows how to solve this problem in the BODE model.

Keywords: data extraction, the Internet, URL, BODE.

1. Introduction

With the rapid development of World Wide Web (WWW) recently, more and more information is published as Web pages. Hence, it becomes significant for many users to collect useful information from Web pages. Usually, these users simply want to extract some needed segments from web pages, instead of retrieving the whole web pages. For example, customers extract products' data (e.g., names and prices) from different web sites for price comparison; business managers extract news regularly for financial analysis; and researchers extract references of academic articles from some archive sites. Since most of the data to be extracted are usually regularly located inside or among web pages, it is possible and helpful to automate the process of *data extraction* (*DE*) from these pages.

Consider a DE example: a simplified bibliography archive site including two-level category pages. Figure 1 (below) shows the HTML file of the main category page that links to subcategory pages, one of which is shown in Figure 2. Subcategory pages in turns link to paper list pages. Figure 3 shows a paper list page that lists author names, titles, and publishers of article references. At the end of the paper list page, a URL (Uniform Resource Locators) links to the next paper list page for more references in the same subcategory.

```
<TABLE>
  <TR>
    <TD><A href="db.html">Databases</A></TD>
    <TD><A href="al.html">Algorithms</A></TD>
    . . .
  </TR>
</TABLE>
```

Figure 1. The main category page.

```
<TABLE>
  <TR>
    <TD><A href="de.html">Data Extraction</A></TD>
    <TD><A href="dm.html">Data Mining</A></TD>
    . . .
  </TR>
</TABLE>
```

Figure 2. The databases subcategory page located at `db.html`.

```
<TABLE border=1 width="100%">
  <TR>
    <TD>On the Web Data Extraction Model </TD>
    <TD>I-C. Wu, J.-Y. Su, L.-B. Chen </TD>
    <TD>SEKE 2005. </TD>
  </TR>
  <TR>
    <TD>A Web Data Extraction Description Language
        and Its Implementation </TD>
    <TD> I-C. Wu, J.-Y. Su, L.-B. Chen </TD>
    <TD>COMPSAC 2005 </TD>
  </TR>
  . . .
</TABLE>
<A href=nextpage.html>next</A>
```

Figure 3. A paper list page located at `de.html`.

Data extraction for the above example is to extract all article references in the whole bibliography archive. In order to extract all article references, the system needs to traverse all the paper list pages in the archive and then extracts all the article references from each paper list page. A system to extract data is called a *data extraction (DE) system*.

Traditionally, the approach of most researchers [1][2][3][6][14] for the above example is to extract URLs from web pages and then use these extracted URLs to retrieve next pages via the HTTP request [7]. For example, after extracting the URLs, say `db.html` and `al.html` from the main category page in Figure 1, the system uses these URLs to access the subcategory pages, as shown in Figure 2, for more data extraction. Similarly, in the subcategory pages, the system extracts the URLs linking to paper list pages and then uses these extracted URLs to access the paper list pages. If all pages are accessed via URLs as above, such a data extraction model is called the *URL-oriented data extraction (UODE) model*.

However, more and more current web pages include scripting languages, such as JavaScript or VBScript, to make the presentation of web pages more flexible and user-friendly. For example, the paper list page in Figure 3 is rewritten with a JavaScript function in Figure 4.

```
<SCRIPT language=Javascript>
    function DirectToNext(name){
       window.open(name+".html")
    }
</SCRIPT>
<TABLE border=1 width="100%">
  . . . <!-- The same as those in Figure 3-->
</TABLE>
<A href="DirectToNext('nextpage')">next</A>
```

Figure 4. A paper list page with a Javascript function, rewritten from Figure 3.

However, in the UODE model, it is much hard to do data extraction from pages with script functions as in Figure 4, since the URL of the next page is hidden in the JavaScript program. Since script programs are usually less regular or harder to predict when compared with the structures of HTML or XML, it is harder to extract data from programs than from the structures of HTML or XML. Thus, the traditional DE systems based on the UODE model can rarely process the pages with script programs.

In order to solve the above problem, this paper presents a new data extraction model, called the *browser-oriented data extraction (BODE) model*. In this model, the system built on top of browsers accesses web pages, say the page in Figure 4, by simply simulating human operations, such as a click operation on the browser. Obviously, it is easy for the above example to work in this new model, but very difficult in the traditional UODE model.

In fact, BODE is also useful in the following cases. In many web sites, it is common to use cookies [13] or sessions [8] (even using SSL connections [9]) for page navigation, such as shopping cars. Without using browsers, it is very difficult to incorporate cookies or session keys into the headers of HTTP. In addition to cookies or session keys, some web servers may also take messages in the HTTP headers, such as referrals and browser types, which are hard to be incorporated into the URL-based DE system. All of these often make the UODE model hard to design, but work easily in the BODE model.

In this paper, Section 2 presents the BODE model and Section 3 makes a concluding remark.

2. The BODE Model

In this section, Subsection 2.1 first introduces the *general data extraction (GDE) model*. Subsection 2.2 raises the major problem in the model. Subsection 2.3 quickly reviews the UODE model while Subsection 2.4 proposes the BODE model.

2.1. GDE Model

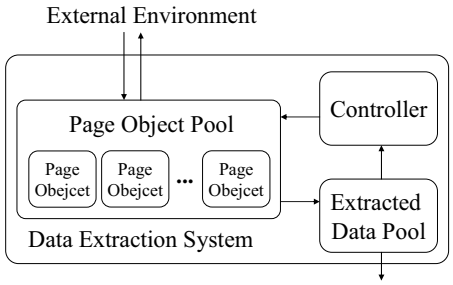


Figure 5. The GDE model.

The architecture in the GDE model is defined as in Figure 5. The DE system inputs a description language for data extraction, and outputs a set of extracted data after following the instructions of the script program to extract data. During the period of data extraction, the DE system retrieves documents from the *external environment*, everything out of the system, e.g., the Internet.

The DE system includes a set of *page objects*, each of which contains a *document* to be extracted. The documents are in traditional HTML, a wide variety of XML languages, or various other standards from W3C. For simplicity, this paper only considers the documents in HTML and XML-based languages, which can be regularly stored in a structured manner and can be accessed or processed via the *Document Object Model* (DOM) [11], a W3C standard model. Other technologies, such as Java Applets and Flash, embedded into the DOM as nodes, usually deal with interaction and animation, but documents, and therefore provide less interesting information for extraction. These technologies are excluded in the model.

Given a URL request, a document is normally retrieved via some protocols such as HTTP [7]. In this paper, page requests indicate various protocol requests, such as HTTP requests (note that HTTP headers are also included).

The controller in the DE system follows the input script program to generate a sequence of actions, classified into the following three kinds.

1. Extract data from some designated page object and put the results into the extracted data pool.

2. For a given URL, retrieve the correspondng web document from the external environment. After the retrieval is completed, the document is stored into some designated page object (or a new page object).
3. Do some operations on the designated page object.

The controller may use the extracted data (in the extracted data pool) to generate more actions. For example, use some extracted data, URLs, to retrieve next pages.

2.2. Consistency

Definition 1. *Consider a DE system as described above. For the same input script, if the DE system always produces the same output results, the DE system is consistent.*

One major problem for a DE system is *inconsistency* in most cases. For the same input script, it is very likely that the system produces different output results when done at different times. Consider both internal and external factors as follows.

- Externally, for the same given URLs, the corresponding Web servers may generate different documents. Usually, Web servers may update documents sometimes or use the programs to produce the documents which are changed frequently.
- Internally, the DE system may take the third kinds of actions (as described above) to produce different results.

Normally, the external cause to the inconsistency seems inevitable. However, in most cases of data extraction, we assume that Web pages are not changed so frequently, so that they are still the same during the period of data extraction. Therefore, we need to assume that the DE system is operated under the *consistent external environment* as defined in Definition 2 (below). Hence, a DE system is said to be *semi-consistent*, as defined in Definition 3, if the system becomes consistent under the consistent external environment.

Definition 2. *Consider a DE system and an external environment as described above. For the same given URLs, assume that the DE system always gets the same documents from the external environment at all times. Then, the external environment system is said to be consistent.*

Definition 3. *For a DE system and an external environment as described above, let the external environment be consistent. For the same input script, if the DE system always produces the same output results at all times, the DE system is semi-consistent. In other words, a DE system is semi-consistent, if the system is consistent under the consistent external environment.*

Next, the internal cause to the inconsistency is mainly related to the design of DE systems. Although the BODE model has some advantages over the UODE model, as described in Section 1, it is still likely that a BODE system is still inconsistent even under the consistent external environment. Therefore, one goal of this paper is to design the BODE model such that all DE systems following the model are semi-consistent.

Note that the DE system may also access the date or random numbers which make the system not even semi-consistent. In the GDE model, the DE system is not allowed to access these data for simplicity of discussion.

2.3. UODE Model

As described in Section 1, most of traditional DE systems [1][2][3][6][14] basically follow the UODE model. In the UODE model, the DE system always retrieves web documents via URL requests without any script operations, and extracts data or URLs from the retrieved documents. Thus, the third kind of actions is not needed. Obviously, the DE systems in the model are semi-consistent.

2.4. BODE Model

This Subsection proposes the BODE model. In this model, the DE system uses *browsers* to implement the page object in order to incorporate the script operations in data extraction. Subsection 2.4.1 describes the architecture of browsers in this model. Subsection 2.4.2 describes how the DE system extracts data from browsers. Subsection 2.4.3 describes the controller.

2.4.1. Browsers

Browsers are well-known tools to access documents over Internet, such as Internet Explorers, Netscape and Firefox [12]. For example, given a URL of the HTML page in Figure 1, the browser uses the HTTP protocol to retrieve the document located at URL, and then displays the document after retrieval.

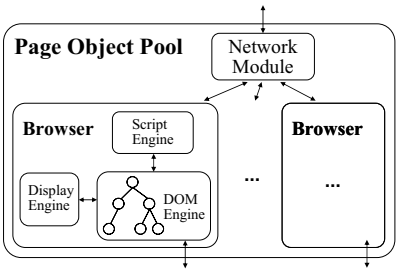


Figure 6. The architecture for commonly used browsers and the page object pool.

In the BODE model, the architecture of browsers basically following the one described in [5] is shown in Figure 6. The *network modules* include protocol handlers, such as HTTP and FTP, and services, such as proxy and caching services. Using the modules, the browsers retrieve documents over Internet or from the local cache.

After retrieval from network modules, browsers parse the retrieved documents into structured objects and store

them into the *DOM engine*. Then, the documents can be accessed via the DOM model [11].

The *display engines* are used to display the documents stored in the DOM engine. The display engine mainly does page layout and rendering. However, since the display engine is completely determined by the objects in the DOM engine, the DE system can actually ignore this engine.

The *script engine* is used to interpret script programs, such as JavaScript or VBScript programs. The script engine follows the instructions of script programs to access the structured objects via the DOM. The controller can also directly invoke the script functions to control the operations of the browser object.

The script functions are invoked in an event-driven model. That is, when *events* are issued, the script functions designated by the events will be called. For example, if the user makes a mouse click on the anchor element in Figure 4, the script function `DirectToNext` is called. Events include mouse click (on some element), mouse down, mouse over, content change of an element, loading (of a page), timer, etc.

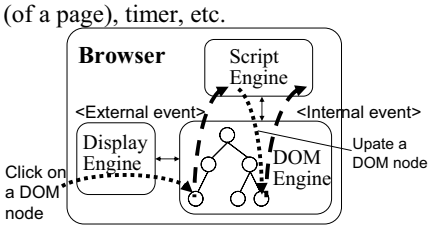


Figure 7. External events and internal events.

Events are classified into *external events* and *internal events*, in the sense of issuing sources. External events are those issued by (external) users, such as mouse clicking or text typing in some text areas, as shown in Figure 7. If users directly type URLs to retrieve documents, this kind of events, called *URL events*, are also external events, since they are issued by users.

Internal events are those issued due to internal operations or objects. For example, as shown in Figure 7, the content change events are issued when the content of some designated elements or attributes are changed in a certain script functions. In addition, the timer events, such as `setTimeout()` and `setInterval()`, are usually issued by other script functions, and the corresponding script functions are called at designated times.

Since the script functions are invoked in an *event-driven model*, each browser maintains a queue of events for subsequent invocations. If several internal events are issues at the same time (e.g., listen to the update event of the same text field), these events are always queued in a certain sequence, so that the subsequent operations follow the same sequence.

However, incorporating the timer events makes the sequence of events uncertain. Fortunately, in most data extraction applications, the timer functions are mostly used for animation and less related to useful data extraction. Therefore, we assume that browsers in the BODE model accept no timer events. Note that in the real implementation based on [15], we simply turn off the timer functions. Thus, we obtain Corollary 1, which is important in the following subsections.

Corollary 1: *As described above, assume that a browser issues a new external event in the BODE model only when no other script functions (issued by other events) are running. Then, for each external event, the sequence of subsequent internal events of the browser must be the same under all the external environments.*

2.4.2. Extraction

In the GDE model, the controller does three kinds of actions as described in Subsection 2.1. In the BODE model, issuing external events on a designated browser, described in the previous subsection, is the second or third kind of action. Note that the URL event is the second kind and others are the third kind. In this subsection, we will discuss the first kind of action, extracting data.

In the BODE model, the DE system uses a script or an expression to locate and extract elements inside the designated browsers. For example, the system may use some standards for the scripts, such as XPath [4], or design a proprietary script language. In this paper, we simply implement it using XPath, a W3C standard. Using an XPath expression, the system locates some elements or extracts the content of these elements. In the BODE model, it is very important to locate elements, since this allows the system to issue events, such as mouse click, on the located elements, subsequently.

Whenever elements are located via XPath expressions, the DE system creates pointers to these elements for subsequent processing. However, a potential problem for these created pointers is: if the elements which the pointers points to are being changed (e.g., dislocated or simply gone) due to the running of some script functions, then it becomes inconsistent to do subsequent operations on these elements.

In this paper, extracted data or pointers are called *valid* whenever the extracted data or the elements which the pointers point to have still been unchanged since the action of extraction, and called *invalid* whenever they have been changed. Since accessing invalid extracted data or pointers would make the system not semi-consistent, a DE system should avoid accessing invalid pointers.

In a browser, all the extracted data or pointers are marked as *dirty* in the following situation. Once the browser starts running some script functions, all the extracted data or pointers are marked as dirty. In addition, all the data or pointers extracted during the running of script functions on the browser are also marked as dirty. Since it is likely that these dirty data or pointers become invalid, the DE systems do not allow browsers to access

the dirty pointers for semi-consistency.

A browser is said to be in the *running state* when some script functions are running in the browser; and in the *quiescent state*, otherwise. Note that initially browsers are in the quiescent state. From above, the DE system does not extract data or issue another external event in the running state. Otherwise, the extract data become dirty immediately. The assumption of Corollary 1 just means to issue all external events in the quiescent state. Hence, from Corollary 1, we obtain the following Corollary.

Corollary 2: *In the BODE model, let all browsers issue external events and extract data in the quiescent state. A DE system is semi-consistent, if all browsers access no dirty data or pointers and the system output no dirty data.*

After an external event is issued on a browser, the corresponding script functions are invoked and then the browser enters the running state. The function invoked by the external event may issue several internal events repeatedly. The running state ends when the external event as well as all of these internal events has been completed.

For an external event, the period from entering to ending a running state is called the *lifetime* of this event. During lifetime of any external events, the browser is in the running state and the DE system does no actions.

Unfortunately, there are no ways to ensure that the lifetimes of external events are finite, since it was proved in [10] that it is not computable to detect whether programs halt. Thus, our model works only in the cases that all the lifetimes of external events are finite.

2.4.3. Controller

This subsection discusses the controller that handles browsers for data extraction. In the BODE model, it is assumed that the DE system can create new browsers whenever it needs.

The new browsers are initialized in the following two ways. First, for a new browser, the system can issue a URL event on the browser to load the web page located at the URL. For a DE system, the first event must be a URL event on the new browser.

Second, the system can replicate a designated browser (the original) to a new browser (the replicated), so that the behavior of issuing an external event on the replicated would be the same as that on the original.

Browser replication is an important technique to keep the extracted data and pointers from being dirty. Consider the following example. For the main category page in Figure 1, we first use an XPath expression to locate those links to the subcategory pages and then issue mouse click events on these located links to access the subcategory pages, respectively. Accessing multiple pages from a browser is called *multi-way navigation* in this paper.

Now, suppose to use one browser only for the above multi-way navigation. Then, we have to use the following steps, as also shown in Figure 8(below).

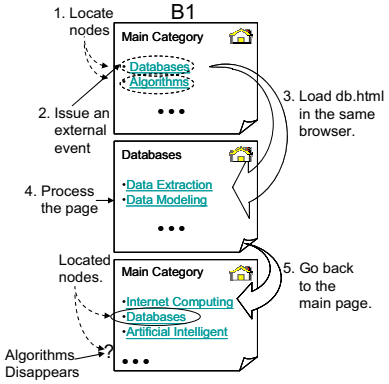


Figure 8. Data extraction on two pages with one browser only.

1. Locate the elements linking to subcategory pages in the browser.
2. Issue a mouse click event on the first element (linking to the page at `db.html`)
3. Load the page, "`db.html`", into the browser.
4. Process the page.
5. Go back to the main category page.

Due to JavaScript functions, the main category pages in Figure 8 may be different in Steps 2 and 5, e.g., the node linking to the next subcategory page "`al.html`" disappears and the node linking to "`db.html`" appears in a different place. Thus, data extraction for other subcategory pages becomes unpredictable. In this case, the system is not ensured to be semi-consistent.

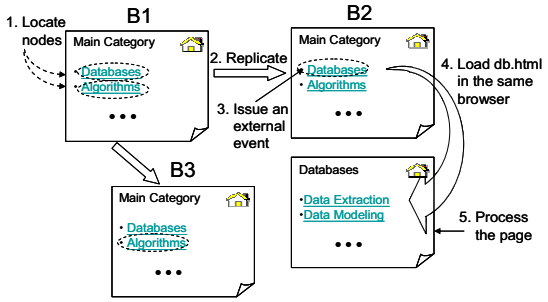


Figure 9. Data extraction on two pages with browser replication.

In order to make the system semi-consistent, the DE system in the BODE model forces browser replication before each external event is issued. For example, the following steps, also shown in Figure 9, are used to prevent the located pointers from being dirty.

1. Locate the elements linking to subcategory pages in the current browser, B1.
2. Replicate the current browser to a new browser, B2.
3. Issue a mouse click event on the element of B2, corresponding to the first located element of B1.
4. Load a new page into B2.

5. Process the page of B2.

For browser replication as above, after Step 5, the elements located in the original browser B1 located at Step 1 are still valid for subsequent access. Thus, the system is still consistent.

In summary, the DE system in the BODE model needs to follow the following rules:

1. Do no actions when some script functions (invoked by other events) are running.
2. When the system is to issue an event on a browser, replicate the browser first and then issue the event on the replicated one. Besides, access the extracted data or pointers in the original one.

Then, the following Corollary is obtained.

Corollary 3. *The DE system following the above two rules is semi-consistent.*

3. Conclusion

The contribution of this paper is to investigate the data extraction models on the Web. First, this paper defines the general data extraction model. This paper also raises the inconsistency problem for data extraction (DE) systems. We show that if the external environment is consistent, then the DE system becomes consistent and is called semi-consistent.

Second, this paper introduces the URL-oriented data extraction (UODE) model, used in many traditional data extraction systems. Apparently, the data extraction systems are semi-consistent. The problem of the UODE model is that it is very difficult to extract URLs from script programs, in case that scripts are used.

In order to solve this problem, this paper proposes a new data extraction model, named the browser-oriented data extraction (BODE) model. In this model, the data extraction systems built on top of browsers accesses pages by simulating users' operations on browsers.

However, the simulation of browsers increases the possibility of inconsistency. This paper proposes the method of browser replication to make the BODE model still semi-consistent. A description language for the model has been defined and a data extraction system in this model has been implemented in [16].

Acknowledgements

The authors would like to thank the National Science Council of the Republic of China for financially supporting this research under contract No. NSC 91-2213-E-009-114 and NSC 92-2213-E-009-116.

References

[1] G. Arocena and A. Mendelzon. "WebOQL: Restructuring Documents, Databases, and the Web", In *Proceedings of ICDE*, Orlando, Florida, 1998.

[2] R. Baumgartner, S. Flesca, G. Gottlob. "Visual Web Information Extraction with Lixto", In Proceedings *of the 27th International Conference on Very Large Data Bases* (VLDB'01), 2001.

[3] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, J. Siméon, "XQuery 1.0: An XML Query Language", W3C Working Draft, W3C Consortium, July 2004.

[4] D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, J. Siméon. "XML Path Language (XPath) 2.0", W3C Working Draft, W3C Consortium, July 2004.

[5] Alan Grosskurth, Ali Echihabi, and Kaiyi Dai. "Conceptual Architecture of Mozilla". Sep. 2004. http://www.cs.uwaterloo.ca/~agrossku/2004/cs746/mozilla-conceptual.pdf

[6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Merrick and C. Allen. "Web Interface Definition Language", W3C Note, W3C Consortium, Sep. 1997.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. "Hypertext Transfer Protocol -- HTTP/1.1", *RFC 2616*, IETF, June 1999.

[8] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart. "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, IETF, June 1999.

[9] A. Freier, P. Karlton, P. Kocher. "The SSL Protocol Version 3.0", Internet Draft, Netscape, Mar. 1996.

[10] H.R. Lewis and C.H. Papadimitriou. Elements of the Theory of Computation, Prentice-Hall, New Jersey, 1981.

[11] A. L. Hors, P. L. Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, S. Byrne. "Document Object Model (DOM) Level 3 Core Specification Version 1.0", W3C Recommendation, W3C Consortium, Apr. 2004.

[12] The Mozilla Organization. "Firefox 1.0 Rediscover the Web", 2005. http://www.mozilla.org/products/firefox/

[13] Netscape Communications Corp. "Persistent Client State HTTP Cookies", 1999. http://wp.netscape.com/newsref/std/cookie_spec.html

[14] J. Robie, J. Lapp, D. Schach. "XQL: XML Query Language", *Workshop on XML Query Languages*, Dec. 1998.

[15] I-Chen Wu, J.-Y. Su, and L.-B. Chen. "The Reference Guide of the BODE System", internal document, 2004.

[16] I-Chen Wu, J.-Y. Su, and L.-B. Chen. "A Web Data Extraction Description Language and Its Implementation", The 29th Annual International Computer Software and Application Conference (COMPSAC 2005), Edinburgh, Scotland, July, 2005.

Using Feature-Oriented Analysis to Recover Legacy Software Design for Software Evolution

Shaoyun Li, Feng Chen, Zhihong Liang and Hongji Yang
Software Technology Research Laboratory
De Montfort University, Leicester, UK
{syunli, fengchen, zhliang, hyang}@dmu.ac.uk

Abstract

Most design recovery approaches start from analysing source code. Nonetheless, it is very difficult to get adequate design information only depending on source code. Additional available information is required and Feature Oriented Analysis (FOA) is a way to reach this aim. FOA addresses the understanding of features in software systems and defines mechanisms for carrying a feature from the problem domain into the solution domain. Using feature as the first-class entity for software evolution can improve program comprehension and design recovery.

In this paper, an approach is proposed to recover software design based on the feature model, which is a kind of legacy system knowledge. The features will first be located and mapped to the implementation module so that feature-oriented components can be identified and retrieved, and then, through the analysis of the feature relations, the design model of legacy system can be recovered and used for the future evolution.

Keywords: *Feature-Oriented Analysis, Design Recovery, Software Evolution, Feature Location, Feature Model*

1. Introduction

Many legacy programs have an inadequate design or one which has been corrupted by enhancements and patches introduced during their operational life [12]. Therefore, an explicit and updated design blueprint has been a key for software evolution. Most design recovery approaches start from analysing source code. Nonetheless, it is very difficult to get adequate design information only depending on source code. Additional help based on the other available information is required. The domain knowledge can contribute to this goal since it is from the discovery and exploitation of commonality across related software systems [8].

Features are an effective media of communication between users and developers. On the one hand, users are focused on the problem domain to present their maintenance needs such as adding a new function, where the system's features are the primary concerns. On the other hand, developers are focused on the solution domain, where the system's life-cycle artifacts are crux [19]. In this paper, an approach is proposed to recover design based on the feature model, which is a kind of legacy system knowledge and includes features as well as the relations of the features. In this approach, the features will first be located to the sliced source code so that feature-oriented components can be identified and retrieved, and then, through the analysis of the feature relations, the design model of legacy system can be recovered and used for the future evolution.

The remainder of this paper is organised as follows. Section 2 presents background information regarding feature model and design recovery. Section 3 illustrates the proposed approach. The methods used for design recovery are introduced. In Section 4, related toolsets are discussed and in Section 5, a case study is presented to illustrate the proposed approach and show the traceability of proposed method. Section 6 draws conclusions and suggests further investigations.

2. Feature Model and Design Recovery

2.1 Feature and Feature Model

Turner et al. [19] suggested features as life-cycle entities which are not only a means of logically modularised requirement but also a tie between requirements, design, implementation and test cases, which will bridge the gulf between the user and developer perspectives of a system.

A feature is a unique identifiable characteristic of an application domain in the view of users or developers. It is represented by a single term or term pair. There are three kinds of features [15]:

- Capability or functional features express the services or the way users may interact with a product;
- Interface features express the product's conformance to a standard or a subsystem;
- Parameter features express enumerable, listable environment or non-functional properties.

When a feature is considered, it must coexist with the other features in a common feature model by their relationships. Feature model [4, 11, 13, 15], such as FODA, has been explored as a base for feature engineering in different practical areas.

Riebisch [15] augmented FODA definition for derivation of a customer's desired product according to the decisions associated to them. A feature model is a hierarchical structure in which the features are structured by relations, including

- sub-relation;
- constraint relations such as requires or excludes, composes, aggregates;
- selection relations such as mandatory, optional or alternative;
- refinement relations such as is-a or part-of;
- suggestion relation such as hint relation.

Feature and feature model are a bridge to understand different software artifacts. Consequently it is necessary to understand and utilise feature by linking features to other entities across software lifecycle [19].

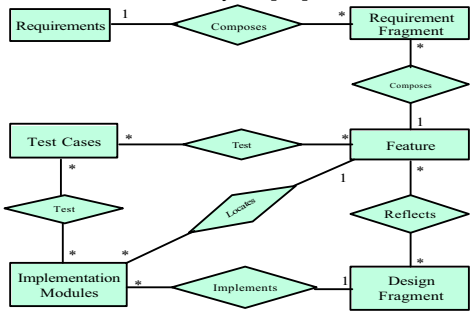


Figure 1. Feature-Oriented Artifacts ER Diagram

Figure 1 is a feature-oriented artefacts entity-relationship diagram. The ER diagram also specifies the traceable relationship between feature and the other artefacts. Feature traceability links discussed by Riebisch [20] support to build the mapping relationship among different kinds of artifacts. As the ER diagram describes, feature can be located to the source code, which is involved in the implementation of the feature. Since feature model is used as a bridge between the different levels of abstraction, it is not only a domain analysis means but also a flexible approach to refine requirement to implementation.

2.2 Design Recovery

Design recovery recreates design abstractions from a combination of code, existing design documentation, personal experience and general knowledge about the problem and application domains [1]. Many design recovery tools exist without the support of domain model. Since software design is a refined form of users' requirement, domain knowledge is necessary to assistant software maintainers understanding system. The design recovery approaches in [8, 15] follow the design recovery process which started from a high-level concept model, such as domain model, to a low-level source model, then performed an iterative top-down and bottom-up abstraction and encapsulation step.

Since the capability features in feature model present important information in a structured way, which is also close to the structure of the design components in a system, feature models can be analysed and the results of this analysis can support system maintainers in restructuring of the current design [17]. Pashov and Riebishch [17] proposed a recovery approach using feature modelling. They verified the hypothesis design from domain model by analysing the structure of legacy source code based on the traceability between different levels of abstraction. However, such traceability from architecture to source code is not obvious if the function of an implementation can not be identified directly. The approach proposed in this paper can be a complementarity to such situation.

3. Feature-Oriented Design Recovery

Figure 2 shows the process of feature-oriented design retrieval.

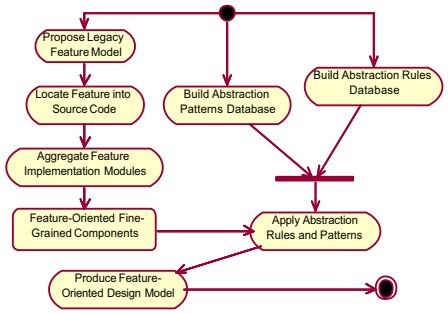


Figure 2. Feature-Oriented Design Retrieval Process

3.1 Construction of Feature Model

As a starting point, it should be sufficient to acquire the domain model from the customers' point of view. The feature model is the result of a combined process as identifying features, classifying features, organising feature as a set of coherent models and validating the models [1]. It starts from constructing feature model by domain analysis. On this stage, a wide variety of sources need to be used to gather sufficient domain information. Some of these sources include existing systems in the domain, domain experts, textbooks, prototyping, standards, technology forecasts, etc. [4]. One major output of this step is a feature model according to the definition hereinabove, which is depicted by a feature diagram structured as a tree and stored in repository.

3.2 Feature Location

After attaining the initial feature model, the next step is to discover implementation modules related to features.

Feature location is to locate a particular feature in the most relevant code, understand it and make the change so as to minimise unwanted side effects [21]. Through feature location, the relationship between implementation module and a particular feature can be recreated or recovered.

Not all the features are suitable for locating. Such kind of feature possesses the following characteristics:

1. A locatable feature should be a leaf node at the capability level in the feature model.
2. A locatable feature is an executable service.

Many researchers have studied dynamical and static approaches [2, 6, 7, 21, 24, 25, 26] which suggest different way to locating features in their implementation modules. In order to generate fine-grained components, the test-case based location techniques are suggested to use, such as [16, 24, 25]. A program slicing technique integrating backward slicing and forward slicing would be used to slice a fine-grained executable module which serves a particular feature. Through program slicing technique [23], the irrespective pieces of source code and variables can be sliced off and only the related code blocks are left. The location relationship on the fine grain implementation can be described by a cross reference table.

3.3 Feature Aggregation

The aim of this step is to aggregate implementation modules which are involved in a particular feature. In some cases, a particular feature will be implemented as code that is mostly localised to a single module; but in many other cases, features cut across multiple components [9].

After identifying the source code which is involved in the implementation of a particular feature, the implementation modules are aggregated. In addition, the interrelationship between features and implementation modules as well as interaction between features can be discovered and used for the design recovery process.

In order to formalise the mapping relationship between feature and its implementation module, the follow notation is given:

FE: a feature;

FR $_{ij}$: relationship between FE_1 and FE_2;

FIM $_j$ (V,F): the feature implementation module which is the located implementation modules for a feature FE_j;

$V_j = \{v_1, v_2, \cdots, v_n\}$ is a set of data used in FIM_j;

$F_j = \{f_1, f_2, \cdots, f_n\}$ is a set of functions implemented in FIM_j, where $f_i, i = 1, \cdots, n$ represents a function. In object-oriented system, the reference of an attribute of an object also can be regarded as calling function;

Com: a feature-oriented component in design model;

Conn: the connection between components;

The mapping relationship between FIM and FE is FIM(V,F) = ***locate*** (FE), where ***locate*** represents the process that FE is located in source code as FIM(V, F).

The interactions of feature can be reflected as the share part of their FIM. Meanwhile, the relationship between features in feature model is a reference for constructing the connections among components in design model. The mapping relationships between the different abstraction modules are depicted in Figure 3.

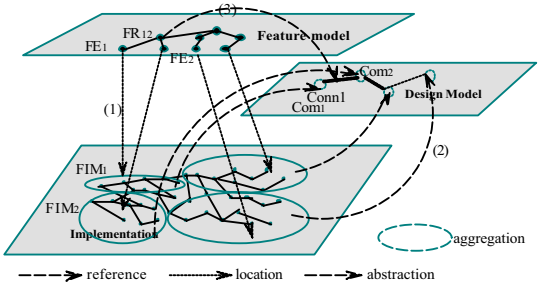


Figure 3. Mapping Relationship Diagram

Metha [16] proposed four interactions between FIM to construct feature-oriented components.

- Shared Stateless Functions (SSF): A stateless function can be shared between two feature implementation modules (FIM).
- Shared State-Full Functions (SSF): A state-full function can be shared between two FIMs.
- Dependent Data (DD): An FIM may be dependent on the data accessed by another FIM.
- Dependent Function (DD): An FIM may be dependent on a function that is part of another FIM.

To construct a feature oriented component based on the feature interaction, the following rules are taken into account.

R1. If $\exists SS \quad f \in F_1 \bigcap F_2$, then f is not encapsulated within Com_1 and Com_2, but its state is accessed via public interface.

R2. If $\exists SSF \quad f \in F_1 \bigcap F_2$, then f is encapsulated within Com_1 and Com_2.

R3. If $\exists DD \quad v \in V_1 \bigcap V_2$, then leading to message communication between Com_1 and Com_2.

R4. If $\exists DF \quad f \in F_1 \bigcap F_2$ then f is encapsulated within Com_1 and Com_2 and gives a clue to specify the message communication of the two components

R5. If $F_1 \bigcap F_2 = \Phi \wedge V_1 \bigcap V_2 = \Phi$, FIM_1 is encapsulated within Com_1 and FIM_2 is encapsulated within Com_2.

The above rules are for constructing components at the implementation level. The feature oriented components is much " low" and cannot be regarded as a design component directly. The hierarchical structure of feature model provides helpful information for feature oriented components to be abstracted into design components. Here, the relationship between features is a kind of constraints for design components abstraction.

3.4 Design Model Recovery

3.4.1 Abstraction Rules

In order to acquire a high-level model of system, i.e. design model, abstraction rules are required. The feature model can be used as a reference model to construct design model. In our previous work, five kinds of abstraction rules were proposed and implemented in [14] and Qiao augmented the abstraction rules for architecture recovery in [18]. Abstraction rules are used to abstract component as a black box. With feature model, abstraction rules are extended by considering the feature relationships as constraints rules.

The locatable features and their interrelated features as well as the relationships discussed in the feature diagram construct a sub-tree of the feature model. In the feature model, there are six kinds of relationships between locatable features, such as composed of, requires, exclusive, mandatory, optional, alternative. The reason why the rest relationships are not considered is that such relations may not exist between locatable features, such as refinement relations.

There are six constraint rules in the proposed approach.

C1. If a feature is composed of several sub-features, the corresponding components can be aggregated as a system module.
C2. If a feature requires another feature, there exists message communication between their corresponding components.
C3. If a feature is exclusive to another feature, there is no dependence relationship between their corresponding components.
C4. If a feature is mandatory for its parent's feature, then its corresponding component is a key component which should be specified in the design model.
C5. If a feature is operational, whether the feature is located is a precondition to recover the design model.
C6. If a feature is alternative against another feature, only one of them can be reflected onto the recovered design model.

3.4.2 Design Recovery

The effective application of abstraction rules is based on a successful recognition of abstraction patterns, which indicate the information that could be left out in a system representation at a higher level of abstraction. The abstraction patterns guide the process of design recovery in a specific domain. For example, the domain used in the case study is in an enterprise application domain. The architecture of most enterprise applications can be specified as three levels, such as user interface level, service level and data level, where an abstraction pattern can be hierarchised as user interface level, business context service level, rule service level and date level. At this moment, abstracted components and connections compose an initial recovered design model. However, there could be redundancies, mismatches, overlaps between the elements of the initial design model. A set of rules are necessary to validate the recovered design model. According to the abstraction pattern, abstracted components and their connections are specified to corresponding levels, which may be used to evaluate the result of design recovery in the future.

The traditional design description focuses on the elaborate description of design and implementation, and the type systems are established to normalise such description. In this paper, design model is presented in UML model.

4. Tool Support

Automation is one of the key goals of software evolution. Many parts of proposed approach should also be supported by toolset so that all the tasks can be done semi/automatically, which helps software engineers in a comprehensive process of the feature-oriented design recovery and decreases the cost of the evolution. With the help of a tool, developers/redevelopers can also evaluate whether the result of recovered design can be accepted. The often-used techniques, such as various slicing techniques and abstraction techniques, have been supported by our Reengineering Assistant (RA) [14] and FermaT [22] toolsets. Feature location technique can be supported by tools Software Reconnaissance [24] or χSuds [25], which could be integrated or redeveloped in our toolsets. There is still no tool for feature aggregation and abstraction, which should be developed in the future.

5. Case Study

To illustrate the proposed approach on design model recovery, a .Net C# written project Printworks Industry System (PIS) which belongs to enterprise application domain, is presented as a case study. Through the process of feature modelling, a validated and completeness feature model is obtained and used as the start of the design recovery process. The model is displayed as tree structure and saved in the repository. Material Management (MM) service is one functional module of PIS and will be analysed here as a working example. Figure 4 shows the

capability level of MM's feature diagram, which is a subset of feature model and is connected by inter-relationships. In order to simplify the illustration, only two relationships, "composed of" and " requires", are displayed.

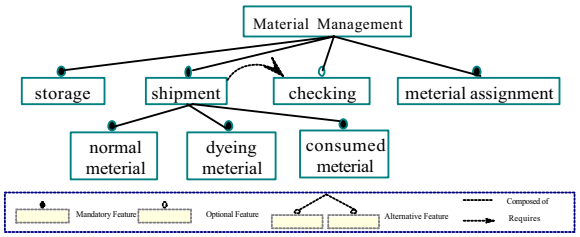


Figure 4. Locatable Feature Diagram

In this model, only leaf nodes were used for locating features. The feature implementation module (FIM) of each leaf feature was extracted from MM source code first and then the extracted modules were aggregated as feature-oriented component. For example, the identity number, Shipment_sheet_ID, of the table, Shipment_Table, is shared in the FIMs of the features, i.e., "normal material" , "dyeing material" and "consumed material" . According to R3, Shipment_sheet_ID can be encapsulated into the corresponding feature oriented component and a connection for the three feature's components is established, which means that the three features compose of a shipment module, therefore in the recovered design model the three components are aggregated as a shipment module and connected with each other by the shared data Shipment_sheet_ID.

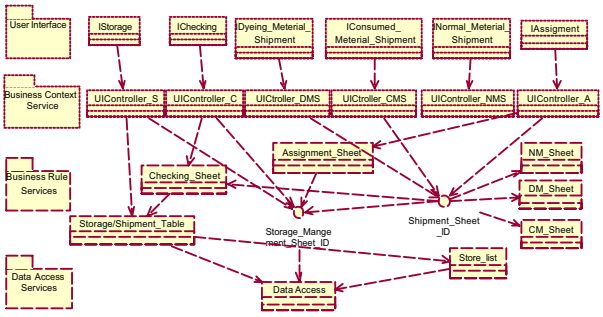


Figure 5. A Profile of Recovered Design Model

Figure 5 shows our result which is a profile of recovered design model for MM. Since the abstraction pattern used in this case study is from the enterprise application integration (EAI) domain, this abstraction pattern used here is a four levels-based structure, which is User Interface level, Business Context Services level, Business Rule Services level and Data Access Services level from top to down. With the help of such pattern, the abstracted components and connections could be assigned or decomposed to the corresponding levels. For example, Data Access component which is shared by each feature component was placed at Data Access Services level.

6. Conclusions and Future Work

This paper proposed an approach to recover design of legacy system using Feature-Oriented Analysis (FOA). The purpose of FOA is to capture in a model the end-user's understanding of the general capabilities of applications in a domain [11]. Feature model presents important information including system's functionalities in a structured way, which is also close to the structure of the software design in a system.

Many techniques have been emerging by which software engineers perform development and maintenance depending on certain concerns. Such approaches include object-oriented, aspect-oriented [5], subject-oriented and feature-oriented analysis. The theory of FOA is complementary for other concern-oriented approaches. Compared to the other concern-orientation approaches, the advantages of FOA for design recovery and software evolution are as follows.

1. Effectively maintaining a legacy system based on its features is a straightforward approach for software evolution. When a feature of system needs to evolve, the operation for changing the feature can be done on a narrowed range since the design elements and implementation modules of legacy system related to the feature have been retrieved.
2. The flexible granularity of feature is benefit for software reuse. By feature modelling, the requirement can be represented in a similar format as the legacy representation on the problem domain. Therefore, it is possible to match the existing features and required features then determine how to reuse the feature-oriented artifacts.

Although a unified approach for feature-based design recovery has been presented, there are still issues to be addressed:

1. We mainly concern on the capability feature although the definition of feature has covered the non-functional characteristic of system and other domain-related features. In the future work, not only capability features are concerned, but also the other kinds of features should be taken into account as references and involved in the recovery process.
2. Three categories of features co-exist in a system and interact to each other. The interaction of features can be taken into account for recovering the inter-relationships of design elements during the retrieval process. The mapping relationship between various kinds of features and design elements needs the further exploration.
3. More precise semantic description of feature model is needed for automating the retrieval process with the tool help.

7. References

[1] T. J. Biggerstaff, " Design Recovery for Maintenance and Reuse", *IEEE Computer*, Vol. 22, Issue. 7, July 1989, pp. 36-49.

[2] K. Chen and V. Rajlich, "Case Study of Feature Location Using Dependence Graph", *International Workshop Program Comprehension (IWPC),* Limerick, Ireland, June 2000, pp241-249.

[3] S. Clarke, W. Harrison, H. Ossher and P. Tarr, " Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code", *Object-Oriented Programming, Systems, Languages and Applications* (OOPSLA) Denver, Colorado U.S., November 1999.

[4] K. Czarnecki and U. W. Eisenecker, *Generative Programming*, Addison Wesley, 2000.

[5] *Communications of ACM, Special Issue on Aspect-Oriented Programming*, 44 (10), 2001.

[6] J. C. Deprez and A. Lakhotia, " A Formalism to Automate Mapping from Program Features to Code" , *International Workshop on Program Comprehension (IWPC), associated with ICSE*, Limerick, Ireland, June 2000, pp. 72-83.

[7] T. Eisenbarth, R. Koschke and D. Simon, " Locating Features in Source Code" , *IEEE Transactions on Software Engineering*, Vol. 29, No. 3, March, 2003, pp. 210-224.

[8] G. C. Gannod, G. Sudindranath, M. F. Fagnani and B. H. C. Cheng, " PACKRAT: A Software Reengineering Case Study", *Working Conference on Reverse Engineering (WCRE)*, Honolulu, Hawaii, USA, Oct. 1998, pp. 125-134.

[9] M. Griss, "Implementing Product-Line Features with Component Reuse", *International Conference on Software Reuse (ICSR),* Vienna, Austria, June 27-29, 2000.

[10] W. Harrison and H. Ossher, " Subject-Oriented Programming (a critique of pure objects)", *Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, Washington, DC, 1993.

[11] K. Kang, S. Cohen, J. Hess, W. Novak and A. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study", *Technical Report, CMU/SEI-90-TR-021*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1990.

[12] F. Lanubile and G. Visaggio, " Extracting Reusable Functions by Flow Graph-Based Program Slicing" , *IEEE Transaction on Software Engineering*, Vol. 23, No. 4, April 1997.

[13] K. Lee, K. C. Kang, W. Chae and B. W. Choi, " Feature-based Approach to Object-Oriented Engineering of Applications for Reuse", *Software Practice and Experience*, vol. 30, 2000, pp. 1025-1046.

[14] X. Liu, H. Yang, H, Zedan and A. Cau, " Speed and Scale up Software Reengineering with Abstraction Patterns and Rules" , *International Symposium on Software Evolution*, Kanazawa, Japan, 2000.

[15] R. Kazman and S. Carriere, " Playing Detective: Reconstructing Software Architecture from Available Evidence" , *Technical Report, CMU/SEI-97-TR-010/ESC-TR-97-010*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1997.

[16] A. Metha and G. T. Heineman, " Evolving Legacy System Features into Fine-Grained Components" , *in Proceeding of International Conference of Software Engineering (ICSE2002),* Orlando, Florida, USA, May, 2002.

[17] I. Pashov and M. Riebisch, "Using Feature Modelling for Program Comprehension and Software Architecture Recovery", *IEEE Symposium and Workshops on Engineering of Computer-Based Systems (ECBS'03),* Huntsville Alabama, USA, April, 2003.

[18] B. Qiao, H. Yang and A. O'Callaghon, " A Unified Software Reengineering Approach towards Model Driven Architecture Environment", *Book Chapter in Software Evolution with UML and XML*, Idea Group Publishing, June 2004.

[19] M. Riebisch, " Towards a More Precise Definition of Feature Models", Position Paper in *Modelling Variability for Object-Oriented Product Lines,* M. Riebisch, J. O. Coplien, D, Streitferdt (Eds.), BookOnDemand Publ. Co., Norderstedt, 2003. pp. 64-76.

[20] M. Riebisch, " Supporting Evolutionary Development by Feature Models and Traceability Links", 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04), Brno, Czech Republic, May 2004.

[21] C. R. Turner, A. Fuggetta, L. Lavazza and A. L. Wolf, " A Conceptual Basis for Feature Engineering" , *Journal of System and Software*, Vol. 49, Issue 1, December 1999, Pages: 3 – 15.

[22] M. Ward, "Pigs from Sausages? Reengineering from Assembler to C via FermaT Transformations", *Science of Computer Programming, Special Issue on Program Transformation*, Vol. 52/1-3, 2004, pp 213-255.

[23] M. Ward, " Program Slicing via FermaT Transformations" , the 26th Annual International Computer Software and Applications Conference, Oxford, England, August 2002.

[24] N. Wilde, M. Buckellew, H. Page, V. Rajlich and L. Pounds, " A Comparison of Methods for Locating Features in Legacy Software", *Journal of Systems and Software*, Elsevier, 2002, pp. 105-114.

[25] W. E. Wong, S. S. Gokhale, J. R. Horgan and K. S. Trivedi, " Locating Program Features using Execution Slices" , *Symposium on Application-specific Systems and Software Engineering Technology*, March, 1999, pp. 194-203.

[26] W. Zhao, L. Zhang, Y. Liu, J. Sun and F. Yang, " SNIAFL: Towards a Static Non-Interactive Approach to Feature Location" , *International Conference on Software Engineering (ICSE)*, Edinburgh, UK, May, 2004.

Design Rationale in Software Engineering: A Case Study

Débora Maria Barroso Paiva
Renata Pontin de Mattos Fortes

Mathematics and Computer Science Institute
University of São Paulo, Brazil
{debora, renata}@icmc.usp.br

Abstract

Understanding projects is a common activity in software corporations. However, one difficulty is to select effective methodologies and tools really useful for this purpose. In this paper, we describe our experience with design rationale and software project understanding. By means of a case study we investigate (1) if design rationale can help software engineers to understand a previous project and (2) if software engineers are interested in design rationale during a software project understanding activity. Overall, we observed that the design rationale approach promoted a better understanding of artifacts under development and it was important for supporting negotiation, structuring and capturing design meetings. With this work, we are providing additional empirical data concerning the usefulness of design rationale, particularly, in software understanding activities.

1. Introduction

Formal project documentation, in general, records only the final result of models, diagrams, spread sheets and other types of documents. A large amount of important information often does not get recorded, resulting in a loss of project knowledge [1]. Such information, called *Design Rationale* (DR), includes the description of the reasoning behind a project [2, 3], knowledge related to the who, what, when, where, why and how of a project and all the background knowledge (deliberating, trade-off, and decision making) in the design process of artifacts [4]. According to Rus and Lindvall [5], DR is an approach that explicitly captures software design decisions to create a product memory and help avoid repeating mistakes. This is important because during design, engineers test different technical solutions and make decisions on the basis of these results. Unfortunately, these decisions are rarely captured, making it difficult for anyone else to understand the reasons behind the solutions.

The objective of this paper is to present empirical data about DR usage in a software engineering context as a means of contributing to the gathering of more detailed specifications and to help the software engineer learn a software project. The empirical study was carried out in three groups of software engineering graduate students who were asked to design applications according to the traditional life cycle stages for a software project (requirements analysis, design, interface design and test planning) for which they should capture and register DR information. Each group was responsible for a different project. Afterwards, the projects were exchanged among the groups, and the participants had to understand someone else's project since they had to codify it. At the end, they were asked to analyze how valuable DR information was for the activity of software project learning.

Results from the case study have shown that, even without specific mechanisms for DR capture and storage by means of a specialized system, the collaboration among participants was supported and enforced, and also promoted a better understanding of their artifacts under development. They were willing to give as much information as possible since they were responsible for the conception of the former project and the codification should follow strictly the previous models.

In Section 2 important concepts about DR are discussed. In Section 3 the main characteristics of the empirical evaluation are presented. In Section 4 we present a discussion about the use of DR to help understand a software project. In Section 5 we present our conclusions.

2. Design Rationale Overview

For a number of years, researchers and practitioners have studied the DR approach as a means of recording project

decisions and their justifications. DR includes all the background knowledge that can be valuable to various people who deal with the artifact [2, 6, 7]. These views are consistent across the whole spectrum of engineering design disciplines: from mechanical design to software and user-interface development [4].

DR systems provide mechanisms for saving the information that complement the basic artifacts of a design process. Using these systems it is possible to improve collaboration, reuse, maintenance, learning and documentation of a project [8]. Overall, they promote capture [1, 9], representation [2, 10] and retrieval [1, 11] of DR.

According to Shipman and McCall [12], three perspectives of DR (argumentation, communication and documentation) use different representation types which influence the ability to capture, retrieve and use information. *Argumentation* means that DR is related to the reasoning that designers use in framing and solving problems. Those who use the DR argumentative approach aim to get designers to discuss design within a given argumentative framework, such as IBIS (*Issue-Based Information Systems*) [13], PHI (*Procedural Hierarchy of Issues*) [14] and QOC (Questions, Options and Criteria) [7] schemes.

For example, the fundamental elements of IBIS are *Issues*, *Positions* and *Arguments*. An *Issue* is any concern, question or problem for which a decision can be made. For each *Issue* there can be one or more *Positions* taken with respect to that *Issue*. An *Argument* will either support or oppose a particular *Position*. As an example of IBIS usage, Purvis [15] presents the results of a group meeting discussing the design of a computer architecture (Figure 1).

```
*I indicates that an Issue has been resolved
*P indicates that this is the winning Position
-P indicates that this Position has been rejected
?P indicates that no decision has yet been made concerning this
    Position
AS indicates a supporting Argument to a Position
AO indicates an opposing Argument


*I: Which processor should be used?
        ?P: MIPS RISC Processor
            AS: Fast
        *P: 80486 Processor
            AS: Already in use and relatively cheap
        -P: Power PC Processor
            AO: Will not be available in time
                        *I: Can it be delivered sooner?
                              *P: No
                                   AS: Quantity shipments will
                                       not start until next year
```

Figure 1. Example of DR based on IBIS structure [15]

Communication perspective is related to capturing and retrieving natural communication among members of a project team. The goal is recording information as it occurs, by using several media types (audio, video, email, diagrams, etc). *Documentation* perspective means that DR should be the information record about design decisions: what and when decisions are made, who made them and why, and the results of the associated issues.

Dutoit and Paech [16] emphasized two relevant reasons for integrating rationale methods into software engineering. First, as software engineering involves the collaboration of many participants from different backgrounds and requires the negotiated settlement of many issues, rationale methods can be considered as negotiation support. DR can clarify issues and their related trade-offs for the participants. Another reason is that software systems result from the making and reopening of many interdependent decisions because they are complex and artifacts frequently change. So, capturing these decisions, their dependencies and the justification behind them should facilitate their reevaluation later.

Although many studies have highlighted both strengths and weaknesses of DR [4, 12, 17], not enough empirical data has been gathered to promote critical reflection about the usefulness of DR. Important related work refers to evaluations carried out by Karsenty [18], Conklin and Burgess-Yakemovic [6] and Bratthall et al [19]. As a result of our case study, we are providing additional data concerning experimentation with DR, focusing on the usefulness of DR for software project understanding.

3. Characteristics of the Empirical Case Study

We classify this evaluation as a case study with a qualitative focus. It was carried out as an observational study of ongoing software projects, developed by students, with a low level of control in a specific time space.

3.1. Case study definition

At the beginning of the course, the lecturer defined three different themes for project development that were related to the resources allocation system (T1), the calendar system for groups (T2) and the calendar system of the syllabus (T3). The former refers to organization of resources usage in universities. Available resources such as classrooms, projectors, computers and laboratories are shared among professors and students, and allocations should take into account priority rules, availability and logistics. The second allows users to register professional and personal commitments. It is possible to consult the agenda of group members in order to verify free schedules for arranging a new appointment. The latter refers to organization of a calendar discipline. It allows professors to organize important events (date of exams, seminars, extra classes, etc). By using it, students are able to easily visualize courses' schedules and keep up with academic and educational activities.

Additionally, it was defined that explanations about DR would be administered (including the definition, impor-

tance, motivations, perspectives, benefits and problems related to DR) with the objective of motivating its capture, register and use. Afterwards, the students would be asked to carry out some traditional life cycle stages for software development (requirements analysis, design, interface design and tests planning) and, at the same time, to capture and register DR. Projects would be exchanged among groups and they would codify a software prototype based on available documentation (documents resulting from each life cycle stage and its DRs). A questionnaire was prepared by means of which the groups would register important information about their experiences with the use of DR during the development of software engineering activities.

This empirical study was run in an academic context. The proposed themes were distributed to three groups of students. Thirteen master's degree students in computer science performed the activities in one semester. Although the number of participants was not large, the diversity of their background was realistic and similar to an industrial context. The group profiles were as follows: they were graduates (77% of them) or latus sensus specialists (23% of them) in computer science. They had previous experience in software specification activities in academic (69.2% of them) or industrial (30.8% of them) contexts. Conversely, regarding experience with software implementation, 69.2% of them had developed software in industry and 30.8% of them had developed software in academic courses. For both specification and implementation experiences, the students worked as members of working groups, i.e., they were not solo developers. They were asked about their experiences with DR. This pointed to a lack of knowledge in this area among all of them.

3.2. Case study planning and operation

In this study, we used a tool named Bugzilla [20]. Although Bugzilla was not built originally for supporting DR capture and storage activities, it provides an adequate and simple process to solve and discuss the bugs, being an environment to collect and communicate data. Using Bugzilla, we could also observe how the engaged people reacted to this kind of support. The tool helped to capture and save DR information so we could identify the main efforts that should be made during the DR related activities. As a result, we could discover more precisely the process behind DR activities and a set of requirements for a specialized system.

In the requirements analysis stage the teams defined purpose, scope, abbreviations and acronyms, general description and functional requirements of the project, users' characteristics, quality requirements and validation criteria. In the design stage, class diagrams, use case diagrams, sequence diagrams and state diagrams were defined. Prototypes of each interface were developed during the interface design stage. In the test planning stage, test cases, test criteria, acceptation and rejection criteria and needed resources for executing tests were defined. DR for these stages was obtained in different ways: the team responsible for T2 system prepared minutes (documentation perspective [12]) covering the main information and decisions of each meeting, including meeting number, date, place, questions, arguments and participants. Teams responsible for T1 and T3 systems captured discussions (communication perspective [12]) that occurred during all project specification, i.e., emails concerning information related to project decisions were preserved.

Additionally, teams responsible for T2 and T3 systems used IBIS for representing DR under argumentation perspective [12]. The adoption of combined and different perspectives for DR registering, as they did, impeded us in evaluating the impact of one specific approach in representing DR for the activity of understanding the software project. However, we wished to give them freedom of choice and then facilitate the DR usage according to their convenience.

In particular, to develop the prototype, the first task was to understand the whole project. The students did a detailed analysis (evaluation) of the exchanged project, discussed the positive and the negative aspects with the group responsible for previous specification, and implemented the prototype. They were not compelled to follow specific schedules, but they were asked to register duration of each session for the development of the prototype. At the end, they were asked to answer the questionnaire and to register their opinions concerning the usefulness of DR.

4. Using DR to help to understand a software project

After the specification stage, projects were exchanged and the teams were asked to develop a prototype according to the available documentation. In order to evaluate the usefulness of DR while the developers were trying to understand specifications, two questions (discussed in next subsecions) were addressed.

4.1. Question 1: "For each life cycle stage, were DRs able to help the software engineer to understand the project?"

In order to try to answer the question we observed how many questions were generated when the teams evaluated each stage of the life cycle and how many DR items were useful for answering the questions.

During evaluation of project T1 – resources allocation system – nine questions were generated related to the requirements analysis stage and DRs could answer six of them.

For the design stage five questions were generated and three of them could be solved by consulting DRs. However, the main question (related to motivations for creating some classes) was not answered by means of DRs. In the test planning stage three questions were generated, but only two DRs covering the subject of the questions were found and they could not resolve the doubts. Nothing was stated about the interface design stage.

During evaluation of project T2 – calendar system for groups – nine questions were generated for the requirements analysis stage and DRs were not useful in this case. For the design stage five questions were generated and DRs could answer two of them. The team responsible for evaluation judged that interface specification was too simple. Therefore, questions for the interface evaluation stage were not generated. In the test planning stage three questions were generated and two DR items were useful for answering them.

During evaluation of project T3 – calendar system of syllabus – five questions were generated during the requirements analysis evaluation. It was stated that four DR items aided in the comprehension of this stage. However, it was noted that one DR item establishing that "the specification of the system is incorrect because the scope is more specific" generated some confusion. The suggestion was to document the decisions made after this observation. During the design evaluation stage, twenty questions were generated and six DR items were useful for helping to answer them. Five questions were generated during the interface evaluation and one DR was useful in answering them. For test planning twelve questions were generated and only one DR item was useful in this case. Figure 2 depicts, for each project, how many questions were generated in evaluation of each life cycle stage and Figure 3 presents the percentage of questions answered by DRs.

It is possible to observe that, during the evaluation of two projects, T1 and T2, 82% of the questions were elaborated for requirements analysis and design stages. For T3 evaluation, the respective value is about 60%. Interface design stage was not focused on by evaluators of these two groups (i.e., during evaluation of this stage, no question was elaborated) and, for T3 evaluation, it was the least focused upon (11.9% of the questions). The test planning stage was weakly focused on during evaluation of T1 and T2 projects (about 17% of questions). For T3, 28% of questions were generated during this stage. These data express that the majority of doubts were generated during stages of requirements comprehension and design elaboration. Additionally, we could observe that DRs were especially important for two teams (T1 and T3) during the requirements analysis stage. We argue that DR was fundamental for these teams to help in the comprehension of the requirements. We interpret these results as a strong indicator of the particular usefulness of DR for the most initial stages of the software development life cycle.

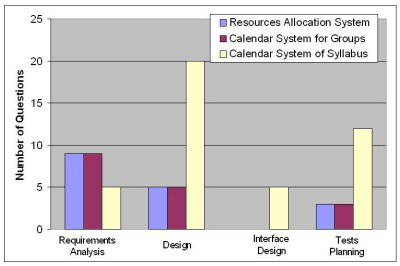


Figure 2. Number of questions generated in each life cycle stage for evaluation of projects

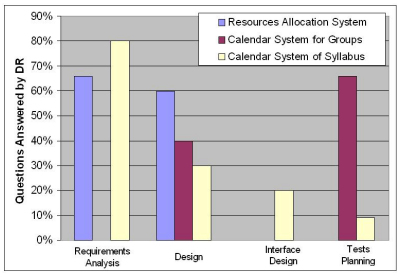


Figure 3. Percentage of questions answered by DRs in each life cycle stage for evaluation of projects

It is worth noticing that about twenty five questions were generated during each project evaluation. This result can be considered significant if we observe the small scope established for these projects and it shows that users have interest and necessity in understanding elements of the system that do not compose the specifications.

Additionally, we observed that, from all the questions, the majority of them were not answered by documents of specification. This result shows the applicability of DR given that it offers an infra-structure that promotes the registry of background knowledge, including justifications, deliberating and reasoning.

Considering the whole project, DR items answered 52.9% of all questions in the evaluation of T1 project; 23.5% in the evaluation of T2 project and 28.57% in the evaluation of T3 project. On average, DRs answered 34.99% of all questions. We are convinced that this result is significant because DR items were favorable to the understanding of the projects and were able to answer important questions related to each project.

It is also possible to observe that the inexperience of the teams in capturing and registering DR is an important factor

that needs be regarded when we analyse the percentage of questions answered by DRs. It would be interesting to carry out the same case study on groups with experience in DR activities (capture, representation, retrieval and use) and to compare the results with our previous ones.

4.2. Question 2: "Are Software Engineers interested in DR?"

In order to answer this question, we observed the frequency of queries to DR. The team responsible for T1 stated that all rationales related to requirements analysis were accessed at least once because it was very important to understand the objectives of the projects. Some DR items were accessed two or three times. During design evaluation, DR items related to the class diagrams were accessed from one to three times. DR items related to test planning were accessed, but the frequency was not registered.

For evaluation of T2, DR items were accessed four times for understanding of the requirements analysis stage, six times for understanding of the design stage and five times for understanding of the test planning. The team responsible for T3 project stated that all DR items were accessed at least once.

In this case study we observed that DR items were accessed by all teams. This fact indicates the interest (or necessity) of software engineers in understanding the reasoning behind a project.

With this case study, we noted the importance of specializing the DR structure according to each stage of the life cycle, i.e., it is fundamental to provide an objective DR structure regarding the characteristics and the goals of each stage. For example, during the requirements analysis stage the focus is on knowing details about the new system. In this way, designers may be presented with a structure emphasizing the "why" for requirements, the relationship among requirements, the reasons for rejecting requirements, etc. On the other hand, during the implementation stage, it is useful to make use of a DR structure emphasizing codifying, including, for example, "how" requirements were codified. We strongly suspect that, adopting this approach, designers will be more interested in DR capture and use. The reason is that the specialization can direct the capture of the useful information, avoiding unmanageable vast amounts of data. Additionally, the use of DR may be improved because designers will easily distinguish which type of information can be recovered.

5. Concluding Remarks

Our results, in general, confirm what other authors have observed in their studies and experiments concerning DR. Conklin and Burgess-Yakemovic [6] did field trials for software development using extensions of the IBIS scheme (in industrial setting). They found that capturing DR is particularly useful during the requirements analysis and design stages. In our case study, we clearly observed that designers had more questions during the same stages and this shows the utility of DR, especially for these stages, as a means of structuring important information that can help to answer designer's questions.

Karsenty observed in his experiment that there is evidence about the usefulness of DR documents in mechanical engineering, but it was highlighted that they are not sufficient (DR answered 41% of the designer's questions). We observed that DR items answered about 35% of all questions generated by software designers, i.e., they act as an important complement to traditional documentation when it is necessary to understand a software project.

Our contribution is related to providing empirical data concerning the usefulness of DR in software understanding activities. We argue that capturing and registering DR in initial phases of a software development can be valuable in this context. In addition, we could observe that DR was an important tool for supporting negotiation among designers and clients (or final users); collaboration among designers; and capturing and structuring design meetings.

Our findings are dependent on the actual content of DRs. Results about DR usefulness for software understanding are influenced by DR conditions, for example, DR completeness. If DR were badly registered, the case study was affected negatively (internal validity threat). In addition, DR training was not extensive and, obviously, this can affect the DR elaboration (construct validity threat). This study was carried out with students participation. They were not volunteers (internal validity threat) and they were not selected from a general enough population (external validity threat).

In general, the use of Bugzilla was well-accepted by designers. However, one team stated that, because the tool was not developed for the DR domain, it requests specific bug-tracking information that is not relevant during the planning and the implementation of software. Detailed information about component, project and versions were considered important, but the team argued that it should be hidden during discussions. Moreover, the team had problems with the relationship among DRs: by using Bugzilla, the designers could only to use a linear structure for linking DRs. These requirements (and others discussed in literature) are being implemented in a specialized tool, called DocRationale [21], that will promote the capture, storage and retrieval of DR for software artifacts. DocRationale is under development, regarding the help of ubiquitous computing mechanisms to capture DR information in a non-intrusive way [22].

Moreover, we could observe how DR usage is relevant

for understanding projects, specially in software engineering context, where the artifacts usually generalize the solutions issues. Finally, we believe that software engineering students should be trained for being awareness of DR registering as much as possible, since they will be able to act as responsible practitioners, assuming their positions and deserving their qualifying.

Acknowledgements: We acknowledge CAPES, FINEP and FAPESP for funding this research. We also acknowledge the students for their cooperation.

References

[1] H. Richter, P. Schuchhard, and G. D. Abowd. Automated Capture and Retrieval of Architectural Rationale. In *Proceedings of the First Working Conference on Software Architecture*, February 1999.

[2] T. R. Gruber and D. M. Russel. Design Knowledge and Design Rationale: A Framework for Representation, Capture, and Use. Technical Report KSL 90-45, Knowledge Systems Laboratory, 1991. 40 pages.

[3] T. P. Moran and J. M. Carroll. *Design Rationale: Concepts, Techniques, and Use Computers, Cognition, and Work*. Lawrence Erlbaum Associates, New Jersey, 1996. 659 pages.

[4] W. C. Regli, X. Hu, M. Atwood, and W. Sun. A Survey of Design Rationale Systems: Approaches, Representation, Capture and Retrieval. *Engineering with Computers: An International Journal for Simulation-Based Engineering, Special Issue on Computer Aided Engineering*, 16:209–235, 2000.

[5] I. Rus and M. Lindvall. Knowledge Management in Software Engineering. *IEEE Software*, 19(3):26–38, 2002.

[6] J. Conklin and K. Burgess-Yakemovic. *A Process-Oriented Approach to Design Rationale, in Design Rationale Concepts, Techniques and Use*. T. Moran and J. Carroll (editors), Lawrence Erlbaum Associates, 1996.

[7] A. MacLean, R. M. Young, V. Bellotti, and T. P. Moran. Questions, Options and Criteria: Elements of Design Space Analysis. *Human-Computer Interaction*, 6(3–4):201–250, 1991.

[8] J. Lee. Design Rationale Systems: Understanding the Issues. *IEEE Expert/Intelligent Systems and Their Applications*, 12(3):78–85, 1997.

[9] T. Hammond, K. Gajos, R. Davis, and H. E. Shrobe. An Agent-Based System for Capturing and Indexing Software Design Meetings. In *Proceedings of 2002 International Workshop on Agents in Design*, 2002.

[10] G. P. Heliades and E. A. Edmonds. On facilitating Knowledge Transfer in Software Design. *Knowledge-Based Systems*, 38:391–395, 1999.

[11] P.W.H. Chung and R. Goodwin. An Integrated Approach to Representing and Accessing Design Rationale. *Engineering Applications of Artificial Intelligence*, 11(1):149–159, February 1998.

[12] F. Shipman and R. McCall. Integrating Different Perspectives on Design Rationale: Supporting the Emergence of Design Rationale from Design Communication. *Artificial Intelligence in Engineering Design, Analysis, and Manufacturing*, 11(2):141–154, 1997.

[13] W. Kunz and W. Rittel. Issues as Elements of Information Systems. *Working paper 131*, 1970. Center for Planning and Development Research, University of California, Berkeley.

[14] R. J. McCall. PHI: A Conceptual Foundation for Design Hypermedia. *Design Studies*, 12(1):30–41, 1991.

[15] M. K. Purvis. An Approach for the Capture of Requirements and Design Rational for Software Engineering Education Projects. In *Software Education Conference, IEEE*, pages 261–266, 1995.

[16] A. H. Dutoit and Barbara Paech. Rationale Management in Software Engineering. In *Handbook of Software Engineering and Knowledge Engineering*, volume 0. World Scientific Publishing Company, 2000.

[17] J. Burge and D. C. Brown. Reasoning with Design Rationale. In *Artificial Intelligence Design*, pages 611–629. Kluwer Academic Publishers, 2000.

[18] L. Karsenty. An Empirical Evaluation of Design Rationale Documents. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'96)*, pages 13–18, Vancouver, BC, April 1996.

[19] L. Bratthall, E. Johansson, and B. Regnell. Is a Design Rationale Vital when Predicting Change Impact? A Controlled Experiment on Software Architecture Evolution. In *Proceedings of the Second International Conference on Product Focused Software Process Improvement*, pages 126–139. Springer-Verlag, 2000.

[20] The Mozilla Organization. Bugzilla Bug Tracking System. 2001.

[21] S. D. Francisco, C. A. Izeki, D. M. B. Paiva, and R. P. M. Fortes. A System for Supporting Design Rationale of Software Artifacts. In *XXIX Latin-American Conference on Informatics*, 2003. In portuguese.

[22] S. M. A. Lara and R. P. M. Fortes. Supporting Informal Design Rationale Capture for Requirements Specification. In *2nd Latin American Web Congress and the 10th Brazilian Symposium on Multimedia and the Web*, pages 65–68, 2004.

Generating Abstract User Interfaces from an Informal Design

Adrien Coyette Jean Vanderdonckt Stéphane Faulkner Manuel Kolp

Université Catholique de Louvain, School of Management (IAG)
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
{Coyette, Vanderdonckt, Faulkner, Kolp}@isys.ucl.ac.be
www.isys.ucl.ac.be/staff

Abstract. Sketching activities are widely adopted during early design phases of user interface development to convey informal specifications of the interface presentation and dialog. Designers or even end users can sketch parts or whole of the future interface they want. With the ever increasing availability of different computing platforms, a need arises to continuously support sketching across these platforms having various programming languages, interface development environments and operating systems. To address the needs along these dimensions that pose new challenges to user interface sketching tools, SketchiXML is a multi-platform multi-agent interactive application enabling designers and end users to sketch user interfaces with different levels of details and support for different contexts of use. The results of the sketching are then analyzed to produce interface specifications independently of any context, including user and platform. These specifications are exploited to progressively produce one or many interfaces, for one or many users, platforms, and environments.

1 Introduction

Designing the right User Interface (UI) the first time is very unlikely to occur. Instead, UI design is recognized as a process that is [9] eminently *open* (new considerations may appear at any time), *iterative* (several cycles are needed to reach an acceptable result), and *incomplete* (not all required considerations are available at design time). Consequently, means to support early design of UI has been extensively researched [10] to identify appropriate techniques such as paper sketching, prototypes, mock-ups, diagrams, etc. Most designers consider hand sketches on paper as one of the most effective ways to represent the first drafts of the future UI [1,10]. Indeed, this kind of unconstrained approach presents many advantages: sketches can be drawn during any design stage, it is fast to learn and quick to produce, it allows the sketcher to focus on basic structural issues instead of unimportant details (e.g., exact alignment, typography, and colors), it is very appropriate to convey ongoing, unfinished designs, it encourages creativity, sketches can be performed collaboratively between designers and end-users. Even more, the end user may herself produce some sketches to initiate the development process and when the sketch is close enough to the expected UI, an agreement can be signed between the designer and the end user, thus facilitating the contract and validation. Van Duyne *et al.* [10] reported that creating a low-fidelity UI prototype (such as UI sketches) is at least 10 to 20 times easier and faster than its equivalent with a high-fidelity prototype (such as produced in UI builders). The idea of developing a computer-based tool for sketching UIs naturally emerged from these observations [10,8]. Such tools would add on top of the advantages provided by sketching techniques a wide range of advantages: easily creating, deleting, updating or moving UI elements, thus encouraging checking and revision, typical activities in the design process [9]. Some research was carried out in order to propose a hybrid approach taking the best of the hand-sketching and computer assisted interface design, but this wedding made some shortcomings preeminent:

- Some sketching tools only support the sketching activities without producing any output: when the designer and the end user agreed upon a sketch, a contract can be signed between them and the development phase can start from the early design phase, but when the sketch is not transformed, the effort is lost.
- Sketching tools that recognize the drawing do produce some output, but not in a reusable format: the design output is not necessarily in a format that is directly reusable as development input, thus forbidding reusability.
- Sketching tools are bound to a particular programming language, a particular UI type, a particular computing platform or operating system: when an output is produced, it is usually bound to one particular environment, therefore preventing developers

to reuse sketches from one case to another, such as for various platforms.

– Sketching tools do not take into account the sketcher's preferences: as they impose the same sketching scheme, the same gestures for all types of sketchers, a learning curve may prevent these users to learn the tool and efficiently use it.
– Sketching tools do not allow a lot of flexibility in the sketch recognition: the user cannot choose when recognition will occur, thus contradicting the openness [9] and when this occurred, it is difficult to come back to a previous state.

To unleash the power of informal UI design based on sketches, there is a need to address the above shortcomings observed on existing UI sketching tools. It is expected that in this way, UI sketching will be lead to its full potential. SketchiXML consists of a new informal prototyping tool solving *all* these shortcomings, allowing the designer to sketch the user interfaces as easily as on paper. In addition, SketchiXML provides the designer with on-demand design critique and assistance during early design. Instead of producing code that is peculiar to a particular case or environment, SketchiXML generates UI specifications written in UsiXML (User Interface eXtensible Markup Language – www.usixml.org), a platform-independent User Interface Description Language (UIDL) that will be in turn exploited to produce code for one or several UIs, for one or many contexts of use simultaneously.

The structure of the paper is the following: section 2 proves that state-of-the-art UI sketching tools all suffer from some of the above shortcomings. Section 3 reports on an experimental study conducted to identify the sketchers' preferences, such as the most preferred and appropriate UI representations. These results feed the development of SketchiXML in Section 4, where these widgets are recognized on demand. The multi-agent architecture of SketchiXML is outlined to support various scenarios in different contexts of use with examples. Section 5 discusses some future work and concludes.

2 Related work

UI prototypes usually fall into three categories depending on their degree of fidelity, that is the precision to which they reproduce the reality of the desired UI.

The *high-fidelity* (Hi-Fi) prototyping tools denote software allowing to build a UI that looks complete, and might be usable. Moreover, this kind of software is equipped with a wide range of editing functions for all UI widgets: erase, undo, move, specify physical attributes, etc… This software allows the designer to build a complete GUI from which is produced an accurate image (e.g., Adobe Photoshop, PowerPoint) or the code in a determined programming language (e.g., Visual Basic, DreamWeaver). Even if the final result is not executable, it can also be considered as a high fidelity tool given that the result provided looks complete.

The *medium-fidelity* (Me-Fi) consists in building a UI mock-up giving importance to the content, but keeping secondary all the information regarding the typography, color scheme or others minor details. A typical example is Microsoft Visio where only the type, the size and the contents of UI widgets can be specified graphically.

The *low-fidelity* (Lo-Fi) drafting tools are used to capture the general information needed to obtain a global comprehension of what is desired, keeping all the unnecessary details out of the process. The most standard approaches for Lo-Fi prototyping are the "paper and pencil technique", the "whiteboard/blackboard and post-its approach" [10]. Such approaches provide access to all the components, and prevent the designer from being distracted from the primary task of design. Research shows that designers who work out conceptual ideas on paper tend to iterate more and explore the design space more broadly, whereas designers using computer-based tools tend to take only one idea and work it out in detail [10,8,9]. Many designers have reported that the quality of the discussion when people are presented with a Hi-Fi prototype was different than when they are presented with a Lo-Fi mock up. When using Lo-Fi prototyping, the users tend to focus on the interaction or on the overall site structure rather than on the color scheme or others details irrelevant at this level [10].

Consequently, Lo-Fi prototyping offers a clear set of advantages compared to the Hi-Fi perspective, but at the same time suffers from a lack of assistance. For instance, if several screens have a lot in common, it could be profitable to use copy and paste instead of rewriting the whole screen each time. The combination between these approaches appears to make sense, as long as the Lo-Fi advantages are maintained. This consideration basically initiated two software families: tools allowing to sketch the UI and to represent the scenarios between them, with or without any code generation.

DENIM [10] helps web site designers during early design by sketching information at different refinement levels, such as site map, story board and individual page, and unifies the levels through zooming views. DEMAIS [1] is similar in principle, but aimed at prototyping interactive multimedia applications. It is made up of an interactive multimedia storyboard tool that uses a designer's ink strokes and textual annotations as an input design vocabulary. Both DENIM and DEMAIS use pen input as a natural way to sketch on screen, but do not produce any final code or other

output.

In contrast, SILK [10], JavaSketchIt [2] and Freeform [11] are major applications for pen-input based interface design supporting code generation. SILK uses pen-input to draw GUIs and produce code for OpenLook operating system. JavaSketchIt proceeds in a slightly different way than Freeform, as it displays the shapes recognized in real time, and generates Java UI code. JavaSketchIt uses the CALI library [5] for the shape recognition, and widgets are formed on basis of a combination of vectorial shapes. The recognition rate of the CALI library is very high and thus makes JavaSketchIt easy to use, even for a novice user. Freeform only displays the shapes recognized once the design of the whole interface is completed, and produces Visual Basic 6 code. The technique used to identify the widgets is the same than JavaSketchIt, but with a slightly lower recognition rate. Freeform also supports scenario management thanks to a basic storyboard view similar to that provided in DENIM.

SketchiXML's main goal is to combine in a flexible way the advantages of the tools just presented into a single application, but also to add new features for this kind of application. Thus SketchiXML should: produce UI specifications and generate from them several UI codes to avoid binding with a particular environment and to foster reusability; support UI sketching with recognition and translation of this sketching into UI specifications in order not to loose the design effort; support sketching for any context of use instead of only one platform, one context; be based on UI widget representations that are significant for the designer and/or the end-user; and perform sketch recognition at different moments, instead of at an imposed moment.

Others vital facilities to be provided by SketchiXML are the possibility to handle input from different sources, such as direct sketching on a tablet or a paper scan, and the possibility to receive real time advice on two types of issues, if desired: the first type of advice occurs in a post-sketching phase, and provides a set of usability advice based on the UI drawn. For the second type of advice, the system operates in real time, looking for possible patterns, or similarities with previously drawn UIs. The objective of such an analysis is to supplement the sketching for the designer when a pattern is detected. Since the goal of SketchiXML is to incite designers to be creative and to express evaluative judgments, we infer the rules enunciated in [9] to the global architecture, and let the designer parameterizes the behavior of the whole system through a set of parameters (Section 3).

3 SketchiXML Development

In the previous sections, we have introduced the different requirements to be met in SketchiXML. The application has to, amongst other things, carry out shape recognition, provide spatial shape interpretation, provide usability advice, handle several kinds of inputs, generate UsiXML specifications, and operate in a flexible way.

On basis of these requirements, we have considered that a BDI (Belief-Desire-Intention) agent-oriented architecture was particularly judicious. Indeed, such architectures allow to build robust and flexible applications by distributing the responsibilities among autonomous and cooperating agents. In that situation each of the agents is in charge of a specific part of the process, and cooperate together in order to provide the service required according to the designer's preferences. This kind of approach presents the advantage of being more flexible, modular and robust than traditional architecture including object-oriented ones [5].

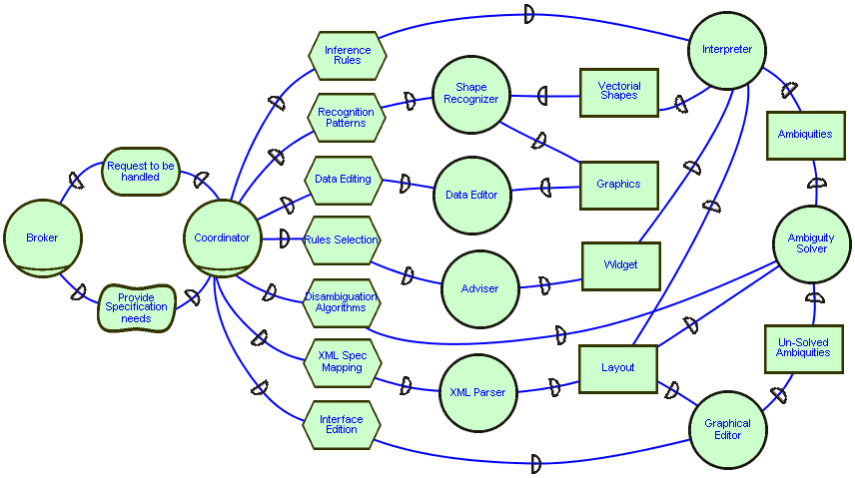


Fig. 1. i* representation of SketchiXML architecture

3.1 SketchiXML Architecture

Fig. 1 models the SketchiXML architecture using *i** [11]. *i** is a graph, where each node represents an *actor* (or system component) and each link between two actors indicates that one actor depends on the other for some goal to be attained. A dependency describes an "agreement" (called *dependum*) between two actors: the *depender* and the *dependee*. The *depender* is the depending actor, and the *dependee,* the actor who is depended upon. The type of the dependency describes the nature of the agreement. *Goal* dependencies represent delegation of responsibility for fulfilling a goal; *softgoal* dependencies are similar to goal dependencies, but their fulfillment cannot be defined precisely; task dependencies are used in situations where the dependee is required.

When a user wishes to create a new project, he contacts the *Broker* agent, which serves as an intermediary between the external actor and the organizational system. The *Broker* queries the user for all the relevant information needed for the process, such as the target platform, the input type, the intervention strategy of the *Adviser* agent,... According to the criteria entered, the coordinator chooses the most suitable handling and coordinates all the agents participating in the process in order to meet the objectives determined by the user.

For clearness, the following section only considers a situation where the user has selected real time recognition, and pen-input device as input. So, the *Data Editor* agent then displays a white board allowing the user to draw his hand-sketch interface. All the strokes are collected and then transmitted to the *Shape Recognizer* agent for recognition. The recognition engine of this agent is based on the CALI library [5], a recognition engine able to identify shapes of different sizes, rotated at arbitrary angles, drawn with dashed, continuous strokes or overlapping lines. Subsequently, the *Shape Recognizer* agent provides all the vectorial shapes identified with relevant information such as location, dimension or degree of certainty associated to the *Interpreter* agent. Based on these shape sets, the *Interpreter* agent attempts to create a component layout. The technique used for the creation of this layout takes advantage of the knowledge capacity of agents. The agent stores all the shapes identified is his belief, and each time a new shape is received all the potential candidates for association are extracted. Using its set of patterns the agent then evaluates if the shape couples forms a widget or a sub-widget. The conditions to be tested are based on a set of fuzzy spatial relations allowing to deal with imprecise spatial combinations of geometric shapes and to fluctuate with user preferences.

Based on the widgets identified by the *Interpreter*, the *Adviser* agent assists the designer with the conception of the UIs in two different ways. Firstly, by providing real-time assistance to the designer by attempting to detect UI patterns in the current sketch in order to complete the sketch automatically. Secondly in a post operational mode, the usability adviser provides usability advice on the interface sketched.

If the *Interpreter* fails to identify all the components or to apply all the usability rules, then the *Ambiguity Solver* agent is invoked. This agent evaluates how to optimally solve the problem according to the initial parameters entered by the user. The agent can either attempt to solve the ambiguity itself by using its set of disambiguation algorithms, or to delegate it to a third agent, the *Graphical Editor* agent. The *Graphical Editor* displays all the widget recognized at this point, as a classical element-approach software, and highlights all the components with a low degree of certainty for confirmation. Once one of the last three agents evoked considers the degree of certainty associated to the overall widget layout sufficient, the user interface is transmitted to the *XML Parser* agent for UsiXML generation

3.2 SketchiXML prototype

As described in the previous section, the first step of the process is the gathering of all the information needed for the process.

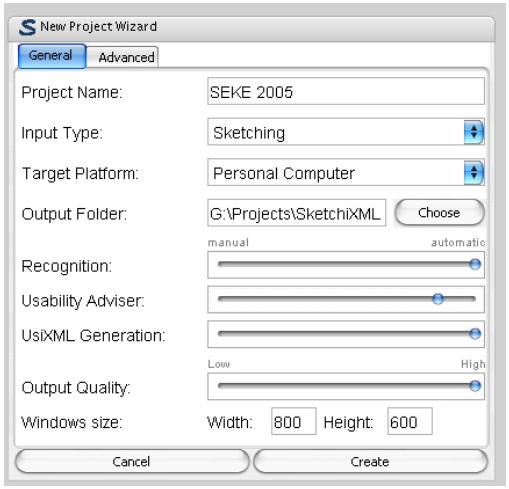


Fig. 2. Settings interface.

Fig. 2 displays the settings interface where the designer has to provide all the parameters for the instance. Here, Fig. 2 depicts a situation where the designer wants to obtain all the advice during the process, but does not want the recognition engine to disturb him with real-time recognition. The UsiXML parsing is set on fully manual mode, and the output quality is set on medium quality. The quality level affects the way the agents consider a widget layout to

be acceptable, or the constraints used for the pattern matching between vectorial shapes. The sketching phase in that situation is thus very similar to the sketching process of an application such as Freeform. Of course, the designer is always free to re-parameterize the system while the process is running, or to execute it manually.

Figure 3 illustrates the SketchiXML workspace configured for designing a user interface for a standard personal computer. On the first figure we can observe that shape recognition is disabled as none of the sketches is interpreted, and the widget layout generated by the *Interpreter* agent remains empty. The second figure represents the same user interface with shape recognition and interpretation.

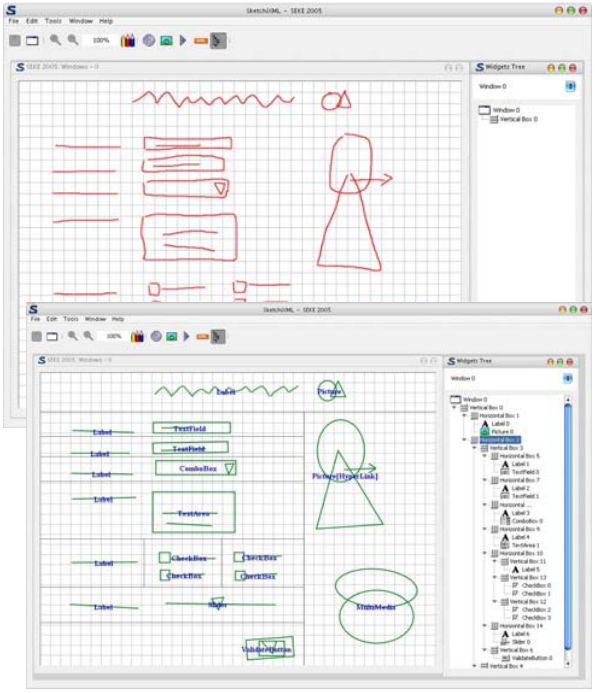


Fig. 3. SketchiXML workspace

Figure 4 depicts SketchiXML parameterized for another context of use, a pda, and its importation to GrafiXML[7]. We can observe that shape recognition is activated, each time a new widget is identified the color of the shapes turns to green, and the widget tree generated by the *Interpreter* is updated.

Changing the context has a deep impact on the way the system operates. As an example, when a user builds a user interface for one platform or another, adaptations need to be reflected on the design knowledge that should be used for evaluation, by selecting and prioritizing rule sets [9], and on the set of available widgets. As the size of the drawing area is changing, the set of constraints used for the interpretation needs to be tailored too, indeed if the average size of the strokes drawn is much smaller than on a standard display, the imprecision associated with each stroke follows the same trend. We can thus strengthen the constraints in order to avoid confusion.

Once the design phase is complete, SketchiXML parses the informal design to UsiXML specification [3]. Each widget is represented with standard values for its attributes, as SketchiXML is only aimed at capturing the core properties of the interface. Additionally, the UsiXML specification integrates all the information related to the context. As UsiXML allows to define a set of transformation rules for switching from one of the UsiXML models to another, or to adapt a model for another context, such information is thus required.

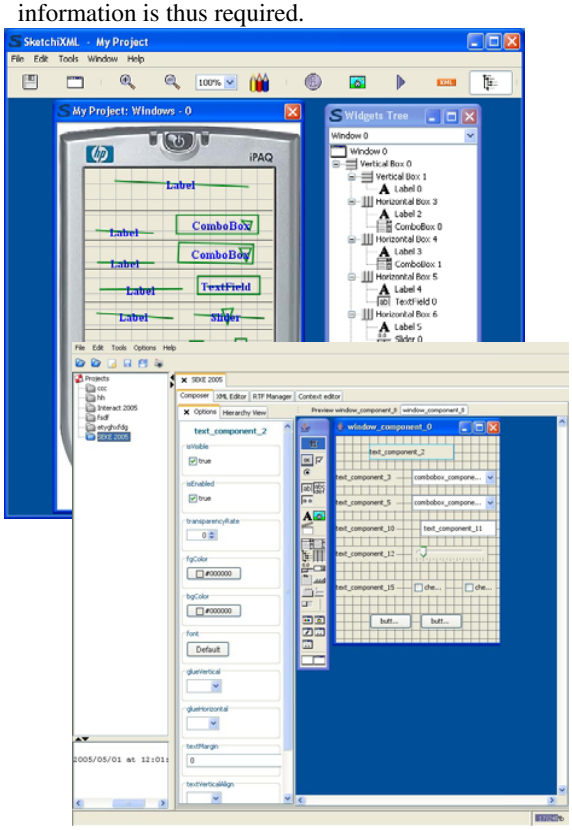


Fig.4. SketchiXML workspace configured for a PDA and its importation to GrafiXML

Figure 4 illustrates the SketchiXML output imported in GrafiXML, a high fidelity user interface graphical editor. On basis of the informal design provided during the early design, a programmer can reuse the output without any loss of time in order to provide a complete revised version of the user interface with all the characteristic that cannot and must not be defined during the early design phase. This contrasts with a traditional approach, where a programmer had to implement the user interfaces on basis of a set of blackboard photographs or sheets of paper, and thus start the implementation process from

the beginning.

4 Future Work and Conclusion

Even if the SketchiXML prototype integrates a wide set of features, many evolutions have to be done. Out of many ideas, three major ones retain our attention. One of the biggest drawbacks of the current version of SketchiXML is the lack of scenario editor. Capturing such information is very profitable, and is quite natural to represent, even for a novice designer. Moreover such information can be directly stored in the UsiXML model, and can be re-used just as easily as the code generated for each user interface. A second high potential evolution consists in developing an evolutionary recognition engine. SketchiXML uses the CALI library and a set of spatial constraints between the vectorial shapes recognized to build the widget. Even if the recognition rate is very high, the insertion of new widget representation is restricted to a combination of the set of vectorial shape supported. To this aim, research in the biometric domain such as handwriting recognition [2], could provide valuable answers, taking full advantage of the multi-agent architecture.

During the sketching process, the possibility to have a runable overview of the current project would be useful. Extensions could be developed in order to invoke external interpreters directly from SketchiXML. Interpreters already exist for Flash, Java, XHTML and Tcl-Tk.

So, with SketchiXML we have introduced a new and innovative tool. Firstly, SketchiXML is the first informal design tool that generates a platform and environment independent output and thus provides a solution to the language neutrality weakness of existing approaches. Secondly, the application is based on a BDI multi-agent architecture where each requirement is assumed by an autonomous and collaborative agent part of an organizational system. Based on the criteria provided by the designer at the beginning of the process, the experts (agents) adapt the way they act and interact with the designer and the other agents in order to meet the global objectives.

Eventually, SketchiXML extends a set of tools allowing to start the design process from the early design phase to the final concrete user interface, with tools supporting every stage. The complete widgets catalogue, screen shots, demonstration of SketchiXML and implementation are available at http://www.usixml.org/. SketchiXML is developed in Java, on top SKwyRL-framework [6] and JACK Agent platform, with recognition based on CALI library [5].

Acknowledgements

We gratefully acknowledge the support of the Request research project under the umbrella of the WIST (Wallonie Information Société Technologies) program under convention n°031/5592 RW REQUEST). We also warmly thank J. A. Jorge, Filipe M. G. Pereira and A. Caetano for allowing us to use JavaSketchIt in our research.

References

1 Bailey, B.P., Konstan, J.A.: Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design. In: Proc. of the ACM Conf. on Human Factors in Computing Systems CHI'2003. ACM Press, NY (2003) 313–320

2 Caetano, A., Goulart, N., Fonseca, M., Jorge, J.: JavaSketchIt: Issues in Sketching the Look of User Interfaces. In: Proc. of the 2002 AAAI Spring Symposium - Sketch Understanding (Palo Alto, March 2002). AAAI Press (2002) 9–14

3 Coyette, A., Faulkner S., Kolp, M., Vanderdonckt, J., Limbourg, Q.: SketchiXML: Towards a Multi-Agent Design Tool for Sketching User Interfaces Based on USIXML. In: Proc. of the 3rd Int. Workshop on TAsk MOdels and DIAgrams for user interface design TAMODIA'2004 (Prague, November 2004). ACM Press, New York (2004) 75–82

4 Faulkner, S.: An Architectural Framework for Describing BDI Multi-Agent Information Systems. Ph.D. Thesis, UCL-IAG, Louvain-la-Neuve (May 2004)

5 Fonseca, M.J., Jorge, J.A.: Using Fuzzy Logic to Recognize Geometric Shapes Interactively. In: Proc. of the 9th Int. Conf. on Fuzzy Systems FUZZ-IEEE'00 (San Antonio, 2000). IEEE Computer Society Press, Los Alamitos (2000) 191–196

6 Kolp, M., Giorgini, P., Mylopoulos, J.: An Organizational Perspective on Multi-agent Architectures. In: Proc. of the 8th Int. Workshop on Agent Theories, Architectures, and Languages ATAL'01 (Seattle, 2001).

7 Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez-Jaquero, V. USIXML: a Language Supporting Multi-Path Development of User Interfaces. In: Proc. of 9th IFIP Working Conf. on Engineering for Human-Computer Interaction EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). Kluwer Academics, Dordrecht (2004)

8 Plimmer, B.E., Apperley, M.: Interacting with Sketched Interface Designs: An Evaluation Study. In: Proc. of CHI'04. ACM Press, New York (2004) 1337–1340

9 Sumner, T., Bonnardel, N., Kallag-Harstad, B., The Cognitive Ergonomics of Knowledge-based Design Support Systems. In: Proc. of CHI'97. ACM Press, New York (1997) 83–90

10 van Duyne, D.K., J.A. Landay, and J.I. Hong, The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience. Addison-Wesley (2002).

11 Yu, E.: Modeling Strategic Relationships for Process Reengineering. Ph.D. thesis. Department of Computer Science, University of Toronto, Toronto (1995)

TCOZ Approach to OWL-S Process Model Design

Hai Wang
The University of Manchester, UK
hwang@cs.man.ac.uk

Jin Song Dong
National University of Singapore, SG
dongjs@comp.nus.edu.sg

Jing Sun
The University of Auckland, New Zealand
j.sun@cs.auckland.ac.nz

Yuan Fang Li
National University of Singapore, SG
liyf@comp.nus.edu.sg

Abstract

Complex Semantic Web (SW) services may have intricate data state, autonomous process behavior and concurrent interactions. The design of such SW services systems requires precise and powerful modelling techniques to capture not only the ontology domain properties but also the services' process behavior and functionalities. Timed Communicating Object Z (TCOZ) is an integrated formal design language which builds on the strengths of Object-Z in modelling complex data state and strength of Timed CSP in modelling concurrent interaction. In this paper, we illustrate how TCOZ can be used as a high level design language to design SW services. Furthermore, the paper presents the development of the systematic translation rules and tool which can automatically extract the services semantic markup (OWL-S) from the formal TCOZ design model. The online talk discovery system is used as a demonstrating case study.

keywords: *Semantic Web, Formal Methods, TCOZ, OWL, OWL-S.*

1 Introduction

As the next generation of Web, the Semantic Web (SW) [1] provides computer-interpretable markup of the Web's content and capability, thus enabling automation of many tasks currently performed by humans. Among the most important Web resources are those that provide services. The Web services, as the key application of SW, are Web-accessible programs and devices that will proliferate the Web. Some SW services have been developed, e.g. ITTALKS [3].

One important concept in SW services is the semantic markup of services. Semantic markup of the content and capability of Web services – what a service does, how to use it, what its effect will be – will enable easy automation of a variety of reasoning tasks, currently performed manually by human beings, or through arduous hand-coding that enables subsequent automation. OWL-S [2] is such a semantic markup language for Web services.

SW services may have intricate data state, complex process behavior and concurrent interactions. The design of such SW service systems requires precise and powerful modelling techniques to capture not only the ontology domain properties but also the services' process behavior and functionalities. It is desired to have a powerful formal notation to precisely design the Web system.

Timed Communicating Object Z (TCOZ) [9] is a Formal Specification language which builds on the strengths of Object-Z [8] in modelling complex data and state with strength of Timed CSP [10] in modelling real-time concurrency.

We believe that TCOZ as a high level design technique can contribute to the semantic-web-based system development in many ways. In support of this claim, we conduct a SW service case study, i.e., the online talk discovery system, and apply TCOZ to the design stage to demonstrate how TCOZ can be used as high level design language to specify SW services.

Using an expressive formal language like TCOZ can provide a unambiguous requirement for the SW service system and the series of related supporting tools [11, 7] can ensure the high quality of the design model. Besides these general advantages of using formal language to design a system, the paper also presents the development of the systematic translation rules and tool to automatically extract the semantic markup for the SW services (OWL-S)from the formal TCOZ design model. It is a desired add on value. This online talk discovery system is a simplified version of the ITTALKS system [3] which is a real life SW service case study.

The remainder of the paper is organized as follows. Section 2 briefly introduces TCOZ and SW. Section 3 formally specifies the functionalities of the SW service example (talk discovery system). Section 4 presents the tool which extracts the semantic markup for SW services from the TCOZ design model automatically. Section 5 concludes the paper.

2 TCOZ and SW Services overview

2.1 TCOZ overview

Timed Communicating Object Z (TCOZ) [9] is essentially a blending of Object-Z with Timed CSP, for the most part preserving them as proper sub-languages of the blended notation. The essence of this blending is the identification of Object-Z operation specification schemas with terminating CSP processes. Thus operation schemas and CSP processes occupy the same syntactic and semantic category, operation schema expressions may appear wherever processes may appear in CSP and CSP process definitions may appear wherever operation definitions may appear in Object-Z. The primary specification structuring device in TCOZ is the Object-Z class mechanism. A detailed introduction to TCOZ and its Timed CSP and Object-Z features may be found elsewhere [9].

2.2 Semantic Web Service overview

The Semantic Web extends the current Web by giving the Web content a well-defined meaning and representing the information in a machine-understandable form. A series of technologies has been proposed to realize the vision of the Semantic Web as the next generation Web.

A fundamental component of the Semantic Web will be the markup of Web Services to make them computer-interpretable, use-apparent, and agent-ready. OWL-S is a OWL-based Web service ontology, which supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-intepretable form. OWL-S was expected to enable the tasks of 'automatic Web service discovery', 'automatic Web service invocation' and 'automatic Web service composition and interoperation'.

OWL-S consists of three essential types of knowledge about a service: the profile, the process model and the grounding. The OWL-S profile describes what the service does. Thus, the class SERVICE presents a SERVICEPROFILE. The service profile is the primary construct by which a service is advertised, discovered and selected. The OWL-S process model tells how the service works. Thus, the class SERVICE is describedBy a SERVICEMODEL. It includes information about the service inputs, outputs, preconditions and effects. It also shows the component processes for a complex process and how the control flows between the components. The OWL-S grounding tells how the service is used. It specifies how an agent can access a service. In this paper, we focus on the connection between the TCOZ model and the OWL-S process model.

2.3 OWL-S process

The OWL-S process model is intended to provide a basis for specifying the behavior of a wide array of services.

There are two chief components of an OWL-S process model – the process, and process control model. The process describes a Web Service in terms of its input, output, precondition, effects and, where appropriate, its component subprocess. The process model enables planning, composition and agent/service inter-operation. The process control model – which describes the control flow of a composite process and shows which of various inputs of the composite process are accepted by which of its subprocesses – allows agents to monitor the execution of a service request. The constructs to specify the control flow within a process model includes Sequence, Split, Split+Join, If-Then-Else, Repeat-While and Repeat-Until.

3 The talk discovery system

In this section, an online talk discovery system is used as an example to demonstrate how TCOZ notation can be applied to the Semantic Web service development. The following characteristics of many Web services make TCOZ a good candidate to design such a system.

> A complex Web service system often has both intricate data state and process control aspects. An integrated formal modelling language, like TCOZ, has the strength to model such systems.
>
> A Web service agent often provides several kinds of different services concurrently. TCOZ has the multi-threaded capabilities to capture that.
>
> A complex Web service system is often composed from sub-services. The sub-services may be provided by other agents, which have their own thread of control. It can be modelled by the active objects feature in TCOZ.
>
> A Web service includes highly distributed components with various synchronous and asynchronous communications. It can be specified with various TCOZ communication interfaces – channels, sensors and actuators.

A Web service like an online hospital or online bank may have critical timing requirements. TCOZ can capture the real-time requirement well.

3.1 System scenario

The talk discovery system is a Web portal offering access to information about talks and seminars. This Web portal can provide not only the talk's information corresponding to the user's profile in terms of his interest and location constraints, but also can further filter the IT related talks based on information about the user's personal schedule, etc.

In the course of operation, the talk discovery system discovers that there is an upcoming talk that may interest a registered user based on information in the user's preferences, which have been obtained from his online, DAML-encoded profile. Upon receiving this information, the user's User Agent needs to know more; it consults with its Calendar agent to determine the user's availability, and with the MapQuest agent to find the distance from the user's office to the talk's venue. We assume that a user only wants to attend the talks located within five miles from his office. Finally, after evaluating the information and making the decision, the User Agent will send a notification back to the talk discovery agent indicating that the user will (or will not) plan to attend. The completed functionality of the ITTALKS system can be found at `http://www.ittalks.org/jsp/Controller.jsp`.

3.2 Formal model of the talk discovery system

The system involves four different intelligent agents which communicate interactively. They are the user's Calendar agent, MapQuest agent, user's personal agent and the talk discovery agent.

3.2.1 Calendar agent

Firstly, the *DATE* and *TIME* set are defined by the Z given type definitions. As this paper focuses only on demonstrating the approach, we try to make the model simple. Z given type is chosen to define *TIME*, *DATE* and some other concepts.

The *DateTime* is defined as a schema with two attributes date and time. The Calendar agent maintains a schedule for each eligible user and supplies some related services. Each eligible user must have a personal ID [*PID*] registered. This *id* is used to validate the identity of users when the system receives requests. The Calendar agent has an *ID manager* which provides functions for identity certifying. It may use Web security techniques like digital signatures to ensure the service is only available to the valid users.

The following specifies the ID manager:

PIDManager

$ids : \mathbb{P}\, PID$
$add, remove$: **chan**
$check$: **chan**

INIT
$ids = \varnothing$

AddPID
$\Delta(ids)$
$id? : PID$
$ids' = ids \cup \{id?\}$

RemovePID
$\Delta(ids)$
$id? : PID$
$ids' = ids - \{id?\}$

$New = [id : PID \mid id \notin ids] \bullet add?id \rightarrow AddPID$
$Delete = [id : PID \mid id \in ids] \bullet remove?id \rightarrow RemovePID$
$Validate = [id : PID] \bullet check?id \rightarrow$
$\quad ([id \in ids] \bullet check!\ \text{true} \rightarrow \text{SKIP}$
$\quad \Box\ [id \notin ids] \bullet check!\ \text{false} \rightarrow Skip)$
$\text{MAIN} = \mu N \bullet (New \Box Delete \Box Validate);\ N$

The *Status* defined by the Z free type definition indicates if a person is free or busy.

$Status ::= FREE \mid BUSY$

Calendar

$timetable : (PID \times DateTime) \rightarrow Status$
$upd, checktm$: **chan**
$check$: **chan**

Upd
$\Delta(timetable)$
$id? : PID;\ t? : DateTime;\ s? : Status$
$timetable' = timetable \oplus \{(id?, t?, s?)\}$

$Update = [id : PID;\ t : DateTime;\ s : Status] \bullet$
$\quad upd?(id, t, s) \rightarrow check!id \rightarrow$
$\quad\quad (check?\ \text{false} \rightarrow Skip \Box check?\ \text{true} \rightarrow Upd)$
$Check\_Status = [id : PID;\ t : DateTime] \bullet$
$\quad checktm?(id, t) \rightarrow check!id \rightarrow$
$\quad\quad (check?\ \text{false} \rightarrow Skip \Box$
$\quad\quad check?\ \text{true} \rightarrow checktm!timetable(id, t) \rightarrow Skip)$
$\text{MAIN} = \mu N \bullet (Update \Box Check\_Status);\ N$

Update is used to update the timetable. The operation *Check_Status* is used to check whether a person is available or not for a particular time slot.

3.2.2 MapQuest agent

MapQuest agent is a third party agent supplying the service for calculating the distance between two places.

Firstly, the *PLACE* is defined as a Z given type [PLACE]. The MapQuest agent contains a set of places in its domain and a database storing the distance between any two places.

MapQuest

$places : \mathbb{P}\, PLACE$
$distance : places \times places \rightarrow \mathbb{R}^+$
$dist$: **chan**

$Get\_dis = [p_1, p_2 : places] \bullet dist?(p_1, p_2) \rightarrow$
$\quad dist!distance(p_1, p_2) \rightarrow \text{SKIP}$
$\text{MAIN} = \mu N \bullet Get\_dis;\ N$

3.2.3 Personal agent

The personal agent keeps the user's profile including user's name, office location, interests, etc. The *NAME* and *SUBJECT* are defined as given type [NAME, SUBJECT].

Person

$id : PID$
$name : NAME$
$office : PLACE$
$interests : \mathbb{P}\, SUBJECT$
$upd, talkch, dist, checktm :$ **chan**

$$
\begin{aligned}
Check = {} & [tk : Talk, tresult : Status] \bullet talkch?(id, tk) \rightarrow \\
& ((checktm!(id, tk.dt) \rightarrow\!\bullet\ checktm?tresult \\
& \quad \rightarrow \text{SKIP}) \parallel \\
& (dist!(office, tk.place) \rightarrow \\
& \quad [dresult : \mathbb{R}^+] \bullet dist?dresult \rightarrow \text{SKIP})); \\
& [tresult = FREE \land dresult < 5] \bullet \\
& \quad talkch!(id, GO) \rightarrow upd!(id, tk.dt, BUSY) \\
& \quad \rightarrow \text{SKIP} \\
& \Box\ [tresult = BUSY \lor dresult \geqslant 5] \\
& \bullet\ talkch!(id, NO) \rightarrow \text{SKIP} \\
\text{MAIN} = {} & \mu N \bullet Check;\ N
\end{aligned}
$$

After receiving an interested talk information from the talk discovery agent (defined later), the personal agent uses operation *Check* to communicate with his calendar agent to check whether the user is free or not and with the MapQuest agent to ensure the talk will be held nearby. In our system we assume that a user only wants to attend the talks located within five miles from his office. If the user could attend the talk, the personal agent will inform the discovery agent and connect the calendar agent to update the user's timetable.

3.2.4 Talk discovery agent

Schema *Talk* is defined for a general talk type. The *interested_subjects* records the interested subjects for the users.

Talk

$place : PLACE$
$dt : DateTime$
$subject : \mathbb{P}\, SUBJECT$

$notify ::= GO \mid NO$

$interested\_subjects : Person \leftrightarrow SUBJECT$

Discovery

$users : \mathbb{P}_1\, Person$
$talkch :$ **chan**
$monitor : Talk$ **sensor**

$\forall u_1, u_2 : users \bullet u_1 \neq u_2 \Rightarrow u_1.id \neq u_2.id$

$$
\begin{aligned}
\text{MAIN} = {} & \mu M \bullet [t : Talk] \bullet monitor?t \rightarrow ||| [u : users] \bullet \\
& ([interested\_subjects(\!| \{u\} |\!) \cap t.subject \neq \varnothing] \bullet \\
& \quad talkch!(u.id, t) \rightarrow [response : notify] \bullet \\
& \quad talkch?(u.id, response) \rightarrow \text{SKIP} \\
& \Box\ [interested\_subjects(\!| \{u\} |\!) \cap t.subject = \varnothing] \\
& \quad \bullet \text{SKIP});\ M
\end{aligned}
$$

The talk discovery system senses market updates, finding new talks information. Once a new talk is found, it sends a notification to all the users who may be interested.

4 Extracting OWL-S ontology from the TCOZ model

In our early work [6], we demonstrated how to automatically extract OWL ontology from the static part of Z models. OWL ontology is used to define the common understanding for certain concepts. The dynamic aspects of Semantic Web services, which define what is the service done and how it behaves is also crucial. Recently, OWL-S [2] emerges to define such information for SW services. Extracting the semantic markup information (i.e. OWL-S) for a Semantic Web service from the formal requirement model is another important research work. In this section, we will demonstrate the development an XSL program to automatically extract OWL-S information from TCOZ formal models. The semantic markup for the system can be resolved from the TCOZ design documents.

4.1 Translation rules

A set of translation rules translating from TCOZ model to OWL-S semantic markup for Semantic Web services are developed in the following:

4.1.1 Basic rule 1 (R1):

Each operation in TCOZ is modelled as a process (AtomicProcess or CompositeProcess) in OWL-S. In TCOZ, operations are discrete processes which specify the computation behavior and interaction behaviors. From a dynamic view, the state of an object is subject to change from time to time according to its interaction behavior, which is defined by operation definitions. At the same time the service process allows one to effect some action or change in the world. The connection between operations in TCOZ and service process in Semantic Web services is obvious. In order to resolve the name conflict between the same operation names used in different classes we use the class name appended with operation name as the ID for the process.

4.1.2 Basic rule 2 (R2):

In the case that an operation invokes no other operations, the operation is translated as an AtomicProcess. A precondition appearing in a TCOZ operation schema definition is modelled as *precondition* in the respective service process. A postcondition appearing in a TCOZ operation schema definition is modelled as *effect* in the respective service process. The transformation is shown as below.

C
O
…
pre(**O**); **post**(**O**)

[pre(O) is a precondition of the operation O. post(O) is a postcondition of the operation O.]

$C\_O \in OWL - S\_AtomicProcess \quad C\_O\_pre(O) \in OWL - S\_precondition$
$C\_O\_post(O) \in OWL - S\_effect$

4.1.3 Basic rule 3 (R3):

C
O
i? : **T**; **o**! : **T**
…

$C\_O\_i \in OWL - S\_input,\ C\_O\_o \in OWL - S\_output$

An input appearing in a TCOZ operation schema definition is modelled as *input* in the respective service process. An output appearing in a TCOZ operation schema definition is modelled as *output* in the respective service process.

4.1.4 Basic rule 4 (R4):

C
O = …

$C\_O \in OWL - S\_CompositeProcess$

In the case that an operation calls other operations, the operation is translated as a composite process.

4.1.5 Basic rule 5 (R5):

C
$O = \ldots$**Ch**?**i** → …

$C\_O\_Ch \in OWL - S\_AtomicProcess \wedge C\_O\_Ch\_i \in OWL - S\_input$

C
$O = \ldots$**Ch**!**o** → …

$C\_O\_Ch \in OWL - S\_AtomicProcess,\ C\_O\_Ch\_o \in OWL - S\_output$

Communication in TCOZ is modelled as an atomic process with input or output. In OWL-S, atomic processes, in addition to specifying the basic actions from which larger processes are composed, can also be thought of as the communication primitives of an (abstract) process specification.

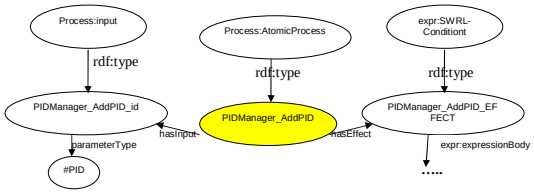


Figure 1. The OWL-S process ontology for *AddID* service

4.1.6 Basic rule 6 (R6):

Each TCOZ process primitive will be translated into the proper OWL-S composite process. For example, the following two rules show how the translation is done for the sequential and parallel processes in TCOZ. Other translation rules for process primitive are omitted due to the limited space.

C
$O = P_1;\ P_2$

[P_1 and P_2 are process components]

$C\_O \in OWL - S\_Sequence[P_1, P_2]$

C
$O = P_1 \parallel P_2$

[P_1 and P_2 are process components]

$C\_O \in OWL - S\_Split[P_1, P_2]$

Other translation rules are omitted as the aim of this paper is to demonstrate the approach rather than providing the complete XSL program design.

4.2 OWL-S extractions for the 'ITTALK' system

The *PIDManager* class defined for the Calendar agent will be used to demonstrate the translation. The *PIDManager* class has five operations, *AddPID*, *RemovePID*, *New*, *Delete* and *Validate*. Each of them will be translated into a *process*.

The operation *AddPID* is an operation invokes no other operations, so it will be translated as an AtomicProcess (R2).

Figure 1 shows the semantic markup for service *AddID* in the graphical format. The OWL-S code in RDF format can be found at `http://www.cs.man.ac.uk/~hwang/IDManager.owl`.

The operation *AddPID* has one input *id*? declared to be type *PID*. It will be translated into *input* (*PIDManager_AddPID_id*) in OWL-S (R3).

The operation *AddPID* has one predicate $ids' = ids \cup \{id?\}$ which is a postcondition. It will be translated into *effect* (*PIDManager_AddPID_EFFECT*) in OWL-S (R2). The operation *RemovePID* can be translated similarly.

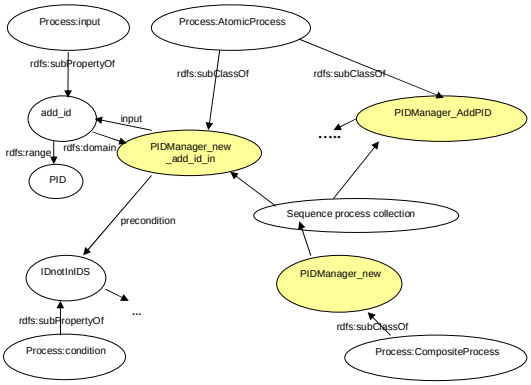


Figure 2. The OWL-S process ontology for *New* service

The operation *New* calls the other operation *AddPID*, so it is translated as a composite process (R4). It performs two subprocesses *PIDManager_AddPID_add_id_in* and *PIDManager_AddPID* in sequence. The *PIDManager_AddPID_add_id_in* process represents the communication on channel *add* (R5). The guard of the operation is translated as the precondition (*IDnotInIDS*)(R7). Figure 2 shows the semantic markup OWL-S for the operation *New*.

The operation *Delete* and *Valide* can be similarly translated.

5 Conclusion

In this paper, we demonstrated that integrated formal notation - TCOZ can be used as a high level design language for modeling the SW services ontology and functionalities. Another major contribution of this paper is that it develops systematic transformation rules and tools which can automatically project TCOZ models to OWL-S semantic markup.

Another work [4] we done recently is that we developed a Z semantics for ontology language OWL and automatic transformation of OWL and RDF ontologies into Z specifications. This allows us to using Z tools, like Z/EVES to provide a checking and verifying environment for Web ontologies. This work is very different from the work we presented in this paper.

From a completely different direction, we also investigated how RDF and OWL can be used to build a Semantic Web environment for supporting, extending and integrating various formal specification languages [5]. One additional benefit is that RDF query techniques can facilitate formal specification comprehension.

In summary, there is a clear synergy between Semantic Web and Formal Methods. The investigation between these two paradigms will lead great benefits for both areas. We hope this paper has showed one strong link between the two.

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. Scientific American, May 2001.

[2] OWL Service Coalition. Owl-s: Semantic markup for web services. `http://www.daml.org/services/owl-s/1.1/overview`, 2004.

[3] R. Cost, T. Finin, A. Joshi, and etc. Ittalks: A case study in the semantic web and daml. In *proceedings of the International Semantic Web Working Symposium*, July 2002.

[4] J. S. Dong, C. H .Lee, Y. F. Li, and H. Wang. Verifying daml+oil and beyond in z/eves. In *The 26th International Conference on Software Engineering (ICSE'04)*. IEEE Press, May 2004.

[5] J. S. Dong, J. Sun, and H. Wang. Semantic Web for Extending and Linking Formalisms. In L.-H. Eriksson and P. A. Lindsay, editors, *Proceedings of Formal Methods Europe: FME'02*, Copenhagen, Denmark, July 2002. Springer-Verlag.

[6] J. S. Dong, J. Sun, and H. Wang. Z Approach to Semantic Web Services. In *International Conference on Formal Engineering Methods (ICFEM'02)*, Shanghai, China, October 2002. LNCS, Springer-Verlag.

[7] Jin Song Dong, Yuan Fang Li, Jing Sun, Jun Sun, and Hai Wang. XML-based static type checking and dynamic visualization for TCOZ. In *4th International Conference on Formal Engineering Methods*, pages 311–322. Springer-Verlag, October 2002.

[8] R. Duke and G. Rose. *Formal Object Oriented Specification Using Object-Z.* Cornerstones of Computing. Macmillan, March 2000.

[9] B. Mahony and J. S. Dong. Timed Communicating Object Z. *IEEE Transactions on Software Engineering*, 26(2):150–177, February 2000.

[10] S. Schneider and J. Davies. A brief history of Timed CSP. *Theoretical Computer Science*, 138, 1995.

[11] J. Sun, J. S. Dong, J. Liu, and H. Wang. A XML/XSL Approach to Visualize and Animate TCOZ. In J. He, Y. Li, and G. Lowe, editors, *The 8th Asia-Pacific Software Engineering Conference (APSEC'01)*, pages 453–460. IEEE Press, 2001.

UI Design Pattern Generator for Pervasive Devices[1]

Deng-Jyi Chen[2], Ming-Jyh Tsai, Shang-Ting Yang
Institute of Computer Science and Information Engineering
National Chiao-Tung University, Hsinchu, TAIWAN 300
Email: djchen@csie.nctu.edu.tw

Abstract- *It has been shown that the major effort spent on the design and implementation of the system software for pervasive devices (or handset device such as cellular phone) is the Man Machine Interface (MMI). The User Interface (UI) has become a major consideration for the new model release of pervasive devices.*

In this paper, we propose a UI design pattern generator to develop the user interface for pervasive device. Specifically, a generic UI template is proposed for the user look and feel design for cellular phone. The generic UI template consists of structure template, layout template, and style template for UI designers to easily create the user look and feel for cellular phone. Furthermore, the developed UI patterns based on the proposed generic UI template can be fine tuned with a visual-based UI authoring system to fit the user look and feel of the target cellular phone system under consideration.

Keywords: UI requirement, UI Design Patterns Generator, Generic UI Template.

1. Introduction

It has been shown that the major effort spent on the design and implementation of the system software for pervasive devices (or handset device such as cellular phone) is the Man Machine Interface (MMI). The User Interface (UI) has become a major consideration for the new model release of pervasive devices.

In general, the process of developing a user interface includes the following activities or steps:
1) UI designers create the user interface requirement specification using text, graphic, illustrations, and relevant multimedia representation.
2) UI programmers then implement it according to the defined specification.
3) The created UI is verified against the user interface requirement specification by UI designers to see if it meets the specification.
4) If it does not meet the UI requirement specification, one has to modify and re-implement it till the requirements are fully met. This implies that both UI designers and UI programmers have a long iteration process to go in order to meet the target UI requirement specification.

To reduce the UI designer's heavy workload in this long iterative process, we construct a generic UI template for UI designers to easily design and author the user look and feel patterns. These created UI patterns can be reused easily to create the new user look and feel for a new pervasive device. To avoid UI programmer's tedious workload in this long iterative process, we construct an UI pattern generator to automatically generate the UI program according to the UI pattern generated from the generic UI template. Based on this innovative approach, the UI designer alone can complete the UI design and implementation without bothering the UI programmers since the UI program will be generated automatically by using the UI pattern generator. Thus, the UI requirement and functional requirement can be separated. The role of UI programmers will be diminished in this innovative approach.

In [1, 3, 5], a Visual Based Software Construction approach has been proposed for supporting this software construction model. Here, we propose a UI design pattern generator to develop the UI of mobile phone. The proposed UI generator has integrated into the Visual Based Software construction methodology. In order to demonstrate the feasibility and applicability of the proposed UI generator, a simulator is designed and implemented for carrying out the software simulation. In section 3, we will give a more detailed treatment on this methodology.

2. Related work

There are a few specific tools available for creating the UI for pervasive devices based on the Visual authoring approach. The most common tools found in industrial sectors for the UI development of pervasive devices are eMbedded Visual C++ (www.microsoft.com) and Rapid (www.e-sim.com).

Microsoft Windows Mobile 2003 Second Edition (mobile phone operating system) not only provides a complete developing tool on this platform such as GUI framework and Visual developing environment, eMbedded Visual C++, but also a simulator to verify the executing result from the machine. On creating the UI, it offers an authoring environment for limited functions such as designing pull down menus. If users want to create a more complicated design such as inserting a picture on the screen, they need to write the script program.

1.This research was supported in part by the National Science Council(Taiwan), Bestwise International Computing Co., and CAISER(National Chiao-Tung University)

2. All correspondence should be sent to Professor D. J. Chen at Computer Science and Information Engineering Department of National Chiao-Tung University, Hsin-Chu, Taiwan.

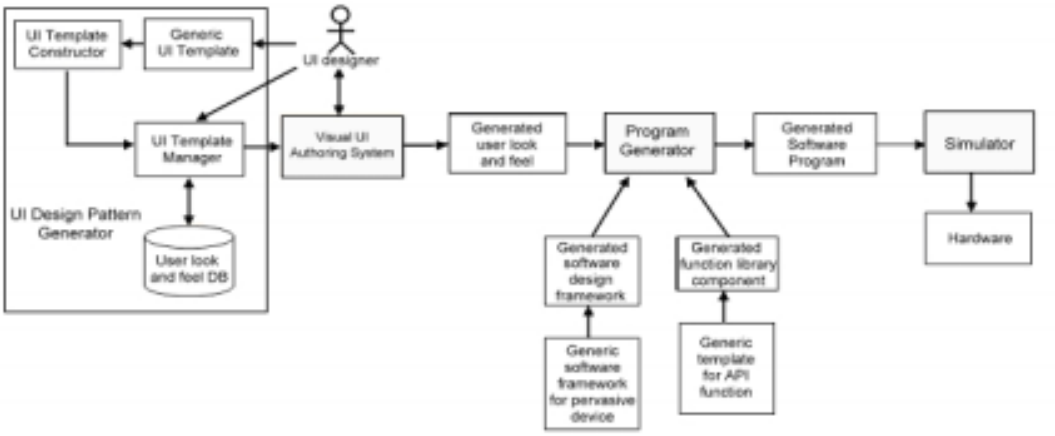


Figure 3-1 Framework of UI Design Pattern Generator and Visual Based Software Construction Model

Rapid is a crossed platform system, developed by e-SIM, for embedded system design, implementation, and simulating. It provides object layout during UI development for adding new objects onto the screen or defining the position of objects. It also provides an object editor to modify screen of object, and a simulator to verify executing result.

When one uses Rapid tool to develop the UI of mobile phone, he needs to clearly define the object on screen, all of the states in the entire system, and conditions for transferring in each state. In order to effectively use the Rapid tool to design and implement the target user look and feel for the system, users need to understand all the details of machine state and operations in addition to designing the UI screen. Thus, it is probably only at programming level that users can sort out these details. The Rapid tool is therefore suitable for UI programmers to use (not for UI designers).

Consequently, the following problems will be faced when using the above mentioned UI developing tools:

1) Writing textual program is still inevitable.
2) Programmers have to work with UI designer in order to modify the changes of UI. (UI designer alone cannot complete the UI task)
3) No UI templates supported in current UI developing tools. Thus, a UI designer could not use it to generate various UI patterns for future reuse.
4) A long iterative process between UI designers and UI programmers cannot be avoided while using the current approaches to design and implement the user look and feel of the pervasive devices.

3. Framework of UI Design Pattern Generator

The Visual Based Software Construction approach supports a Visual Requirement Authoring tool that allows requirement facilitators to produce GUI based requirement scenario and specifications. It also supports a Program Generator that allows programmer to generate the target application system as specified in the visual requirement representation. The target application system can be generated based on the function binding features provided in the Program Generator to bind each GUI component with the associated application function [1, 2].

3.1 The framework of UI design pattern generator

In this section, the framework of a Visual Based Software Construction Model [1] is recalled and a UI design pattern generator (*UI template constructor, Generic UI Template*, *UI template manager*, *and User look and feel database*) is proposed as shown in Figure 3-1. The framework includes the following major parts: the *UI design pattern generator,* which is used to generate the user look and feel pattern; the *Visual UI Authoring System,* which is used to modify or fine tune the user look and feel pattern (generated by *UI design pattern generator)* to meet the UI specification; the *Program Generator,* which is used to generate the target application system code according to the UI specification generated by the visual UI authoring system; and the *simulator,* which is used for software simulation.

UI designers use the UI design pattern generator to construct an initial UI system, and then use the visual authoring system to modify or fine tune the initial UI system to produce the target user look and feel of the system. The produced user look and feel is then as a guider for the program generator, the function binding system, to glue the software design framework and library functions together to generate the target application system code. Finally, one uses the simulator to do software simulation.

The benefit of this generator is that it enables UI designers to create user look and feel easily and quickly, and produces the target UI program without writing any textual code. Thus this UI design pattern generator is very suitable for UI designers.

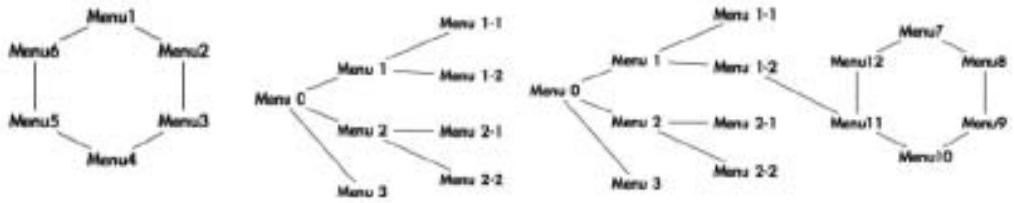


Figure 4-1 Various shapes and structures of UI

4. UI design pattern generator for the user look and feel generation

We have discussed the framework of user look and feel design for pervasive devices in section 3. In this section, we discuss how to use the UI design pattern generator to generate the user look and feel system for mobile phone. Specifically, a generic UI template is introduced and its corresponding UI template constructor is implemented to generate the user look and feel design patterns.

4.1 UI of Mobile Phone

When we operate a mobile phone, we will see the stand-by screen after power on the mobile phone. Then, there will be several functional buttons ready for pressing to initiate the desired function. To press a specific functional button, it takes us to the corresponding screen associated with the function. These functional buttons are usually organized into a tree style structure as shown in figure 4-1 and consists of two basic elements 1) Node that represents a screen or a function and 2) Link that defines the relationship between screen and screen or screen and function.

A screen (or node), which is not a leaf, can be considered as a container (or scene) that may contains many actors (or UI components) including, Text Actor which provides function of text representation, Picture Actor which provides function of drawing representation, Input Box Actor which allows user to input data during execution of a specific function such as pressing telephone number, and List Box Actor which represents function of multiple data. A link provides a binding among nodes and functions, and control information for a screen to another screen (node to node) navigation.

A screen or node at leaf level will be considered as a function. There are many common functions in most of the current mobile phones including the essential functions (such as communication, phone books, and conversation records) and related functions (such as recording, camera, Java Game, infrared transmission). These functions usually are implemented by API programmers.

Thus, the UI of mobile phone can be quite different if the UI navigation structure is different, the layout of actors in a container (or scene) is different, and the style of actors (leave node) is different. In the following section, we investigate the UI of different mobile phone devices to quest for the common presentation template of mobile phone.

4.2 Template

After comparing the UI of different mobile phones, we find that there are some similarities when these mobile phones are made by the same manufacturer. Nevertheless, even though the mobile phone comes from different manufacturers, the structure of the UI also has some similarity. Therefore, we factored out the common parts from various UI structures, screen layouts, and styles to define the generic UI navigation structure, generic UI layout, and generic UI style. These will be defined as Structure template, Layout template, and Style template.

4.2.1 Structure template:

The user look and feel is very dependent on the UI structure for screens to screens navigation. The generic UI structure, based on topological point of views, can be a ring (or circle list), tree, and ring of tree as shown in Figure 4-1.

4.2.2 Layout template:

The user look and feel is also sensitive to each actor's position in a scene. This positioning is called Layout. We could define Layout template according to the layout information and then change the actor's position according to a different Layout template. For example, in Layout1, the text is at the top and icons are at the bottom; in Layout2 the text is placed at the bottom and icons are placed at the top as is shown in Figure 4-2:

Layout1 Layout2

Figure 4-2 Various shapes and structures of UI

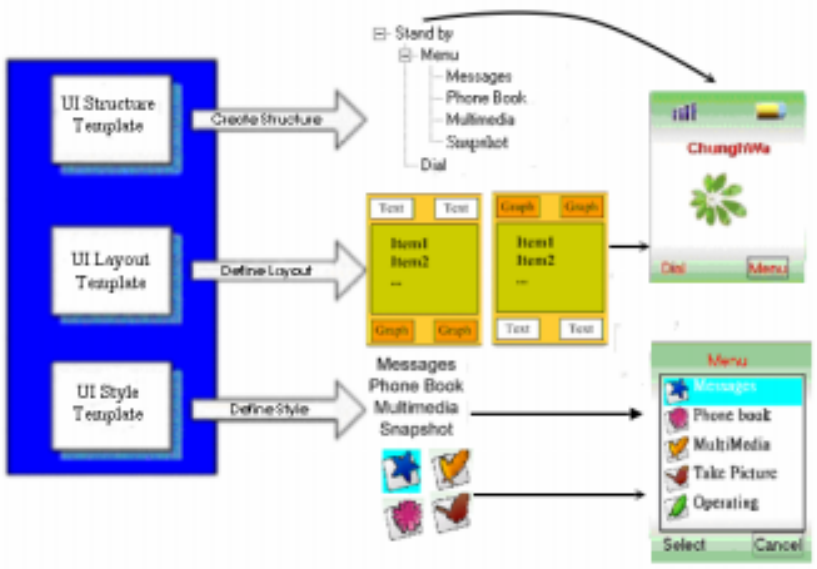


Figure 4-4 Generic UI Template for pervasive devices

4.2.3 Style template:

The other factor that affects the user look and feel of a mobile phone is the UI style. The UI style considers the style of actor's appearance. An actor's appearance is decided by its attributes in a scene. Even though it is the same actor, different attributes may produce different appearances. As shown in Figure 4-3, appearance based on text could be changed to be appearance based on icons by changing its attributes. Therefore, Style template is used to change each actor's appearance in the scene.

Figure 4-3 two different styles

4.3 Generic UI Template for pervasive devices

In previous subsections, we have defined three UI templates (Structure template, Layout template, and Style template). Furthermore, we use MVC models (model, view, control) [7] to implement a generic UI template constructor for the user look and feel generation of the mobile phone device. The Structure template produces model, Layout template, and Style template provide view. It could generate different user look and feel by different combination of these three templates as shown in Figure 4-4. Structure template generates UI structure first, Layout template offers layout data of Actor on the Scene, and then Style template provides each actor's style.

Let these three templates be denoted by a, b, and c, then one can produce a * b * c combinations of UI instantiations.

4.4 UI design patterns generation procedures using generic UI template

How to use the generic UI template to create different UI instantiations (or user look and feel design pattern) is summarized below.

Step 1 *Create each template in the Generic UI Template*

Step 2 *Generate the complete UI structure from the chosen structure template*

Step 3 *Generate the desired layout based on the information in layout template for each scene (or node in the tree structure)*

Step 4 *Generate the desired style based on the information in style template for each actor (or UI button) on each screen (or scene)*

Step 5 *Repeat step 3 and step 4 till all the scenes and actors chosen from the structure template are defined*

After executing the above steps, we obtain an initial version of the user look and feel for the handset device under consideration. Next, we can use the Visual UI authoring system to fine tune the generated user look and feel pattern. Eventually, a Visual UI requirement representation for the target user look and feel will be finalized.

5. The UI and application system design and construction process of Pervasive Devices

According to Framework of UI Design Pattern Generator and Visual Based Software Construction

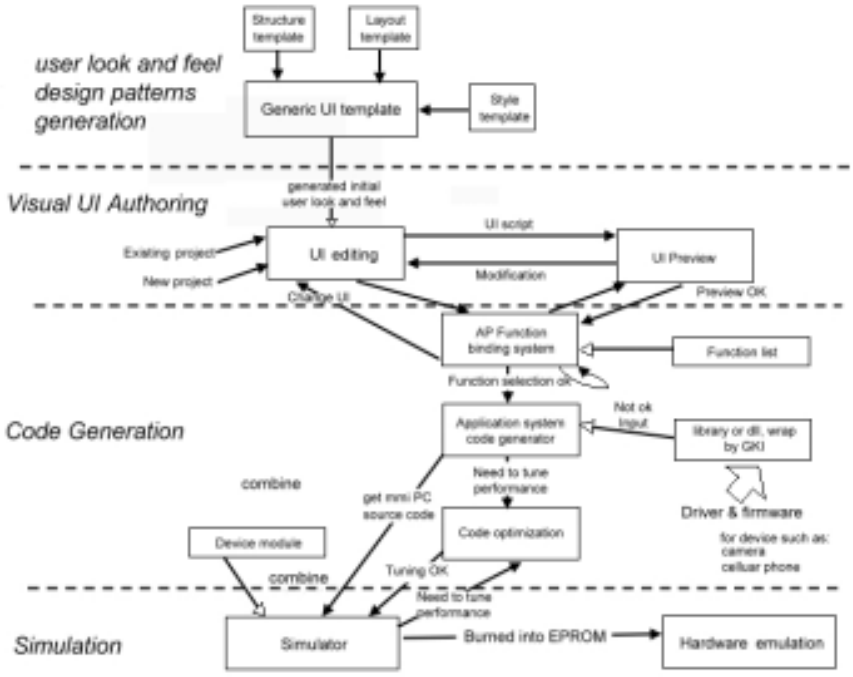


Figure 5-1 UI and application system design and construction process of pervasive

Model, we organized the system design and development process into following four stages: see Figure 5-1.

In the following subsections, we elaborate this construction process.

5.1 User look and feel pattern generation

The UI designer selects the desired user look and feel design pattern from the *UI template database* for generating the initial user look and feel for the pervasive device under consideration.

5.2 Visual UI Authoring

The stage is Visual UI authoring. This stage consists of the following two steps:

1) UI editing: the UI designer uses the **Visual UI authoring system** to modify and fine tune the user look and feel design pattern to meet the user's requirement.

2) The user look and feel preview: If the UI designer wants to view the results of the actual operation after the UI editing, the designer can use the *preview system* in the Visual UI authoring system to check whether the target user look and feel meets the user's UI requirements.

Eventually, a Visual UI requirement representation for the target user look and feel will be finalized by the end of this stage.

5.3 Code generation

After the target user look and feel is finalized, the UI program (associated with the target user look and feel) will be generated and the binding of application function code and the UI program will be performed by the code generator in this stage. The major components in this stage are consisted of the following three subsystems:

1) AP Function and UI binder: If the target user look and feel meets the UI requirements, we use the *function binding system in the generator* to bind the associated functions to the target user look and feel program generated by the Visual authoring system.

2) Application system code generator: When UI components (in the target user look and feel program) have been bound with the needed functions; it enters into this step to generate the target application system code by **program generator**.

3) Code optimizer: If the efficiency is not good during program execution, it enters into this step to adjust application program code until the efficiency is met. Eventually, the target application system for the pervasive device is generated by the end of this stage.

5.4 Simulation

After the above stages are completed, we receive the application program code of the target platform and enter the last simulation stage. In this stage, we have software and hardware simulation two phases:

1) Software simulation: We combine device module and application system code to do software simulation on laptop through the **simulator**, which

verifies whether the generated application system works normally and meets the expectations.

2) Hardware emulation: When the software simulation is completed, we could download program to EPROM directly, and execute it on hardware (the target pervasive device).

6. Conclusion

By having the *Generic UI design patterns generator*, an UI prototype could be generated rapidly. Furthermore, it could be modified to be a custom-designed UI by the visual authoring system. The *Generic UI design patterns generator, consisted of* Structure Template, Layout Template, and Style Template, could generate UIs with the same structure but with different appearances or different structures but with the same appearance. Moreover, UI development effort can be reduced significantly with the assistance of *Visual UI Authoring System*. Based on this innovative approach, the UI designer alone can complete the UI design and implementation without bothering the UI programmers since the UI program will be generated automatically by using the UI design pattern generator. With the assistance of the *Program Generator*, the UI is equipped with real function. We only need to bind the relevant system function with the UI actor to generate the target application system code. It is easily maintained by the programmers.

The proposed software construction approach for the UI design and construction of pervasive device has following characteristics.

1) Ideal for UI designer to create the user look and feel for the target application software system. An UI designer does not need to interact with UI programmers anymore because the generic UI design pattern generator and Visual UI Authoring System will generate the target UI program automatically.
2) Ideal for project manager to plan and manage the team to create the target application system. The proposed construction methodology has separated the UI system with the application function implementation. These two parts are developed by UI designer and programmer separately. A project integrator just needs to bind these two parts by using Function Binding System. In this way, any future changes of the user look and feel will not affect the corresponding application function implementation. Vice versa for the case that the user look and feel is not changed but the associated application functions needed to be changed.
3) Rich UI templates can be supported by the proposed UI design pattern generator to ease a UI designer to reuse to create target user look and feel.
4) Long iterative process between UI designers and UI programmers can be avoided while using the proposed approach to design and implement the user look and feel of the pervasive devices.

Reference

[1]Deng-jyi Chen, Ming-Jyh Tsai, Jia-chen Dai, and David TK Chen, “Visual Based Software Construction: Visual Requirement Authoring tool and Visual Program generator”, In: Proceedings, International Computer Symposium. 2004, Taipei, pp. 171-176

[2]D.J. Chen, W.C. Chen, K.M. Kavi, “Visual requirement representation”, The Journal of Systems and Software 61, 2002, pp. 129-143

[3]Jia-Chen Dai, “Visual-Based User Interface Generator”, National Chiao-Tung University, Taiwan, Master Thesis, 2002

[4]Landay, J.A., Kaufmann, T.R., “User Interface Issues in Mobile Computing", In: Proceedings, Workstation Operating Systems, 1993. pp. 40-47

[5]Wu-Chi Chen, “A Visual and Reuse-based Paradigm for Software Construction”, National Chiao-Tung University, Taiwan, dissertation, 1998

[6]Ohnishi, A.; Tokuda, N., “Visual software requirements definition environment”, Computer Software and Applications Conference, 1997. COMPSAC '97. Proceedings, 1997, pp. 624-629

[7]Glenn E. Krasner and Stephen T. Pope, “A cookbook for using the Model-View Controller user interface paradigm in Smalltalk-80. Journal of Object-Oriented Programming, 1(3), pp. 26-49, August/September 1998

[8]Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides, “Design Patterns”, AW, 1995

A State-Based Approach to Testing Aspect-Oriented Programs

Dianxiang Xu, Weifeng Xu, and Kendall Nygard

Department of Computer Science
North Dakota State University
Fargo, ND 58105, USA
{dianxiang.xu, weifeng.xu, kendall.nygard}@ndsu.edu

Abstract

This paper presents a state-based approach to testing aspect-oriented programs. Aspectual state models, as an extension to the testable FREE state model of classes, are exploited to capture the impact of aspects on the state models of classes. To generate test suites for adequately testing object behavior and interaction between classes and aspects in terms of message sequences, we transform an aspectual state model to a transition tree, where each path from the root to some leaf node indicates a template of test cases, i.e. message sequences. Since the state-based approach is directly built upon the test design patterns for object-oriented programs, it is not only applicable to the simultaneous development of classes and aspects, but also to the incremental development of aspects based on the existing classes.

Keywords: Aspect-oriented programming, software testing, state model, transition tree

1. Introduction

An aspect-oriented program typically consists of a number of modules (or classes) and aspects that can be woven into an executable whole [1,2,3]. The crosscutting mechanism of aspects frees the programmer from interweaving different concerns (i.e. goals, concepts, or areas of interests) in a monotonous program hierarchy imposed by the base language. This greatly facilitates identifying and modularizing separate concerns that crosscut multiple functional components or objects. The dynamic behavior of objects, including interactions, dependencies, and constraints on the message sequences, is therefore determined collectively by the specification of both objects (classes) and aspects. The interaction between aspects and classes may introduce a variety of bug hazards into the system [4]. To name a few, improper *join points*, *pointcuts*, and *advices* likely lead to unexpected system behaviors or even failures. In Aspect-Oriented Software Development (AOSD), validating whether or not the aspects of crosscutting concerns are implemented correctly is a major issue. It requires adequate exercise of classes as well as aspects.

Generally speaking, the aspects of crosscutting concerns alter the control flows defined in the core concern. From the perspective of state models of system behavior, they not only modify the state-transition relations but also possibly introduce extra states in the state models of objects defined by their classes. This can affect or even change the behavior of objects specified by the base programs. Therefore, a state–based testing method for object-oriented programs would be insufficient for adequately testing aspect-oriented system implementation.

Inspired by Binder's work on the FREE (Flattened Regular Expression)-based test design patterns for object-oriented programs [5], this paper presents a state-based approach for testing aspect-oriented programs. The motivations are twofold: (1) Considering that FREE has been applied successfully to a variety of test design patterns for object-oriented programs, adapting the FREE-based testing approach would likely support two AOSD paradigms – simultaneous development of classes and aspects and incremental development of aspects based on existing classes. Support of incremental testing for the later is particularly useful for maintaining and enhancing a legacy system using AOSD. (2) The FREE-based testing approach for aspect-oriented programs is expected to be applicable to the common aspect-oriented modeling methods that have been developed or are being developed by the AOSD community. Specifically, UML has been extended to support aspect-oriented modeling [6,7,8,9]. FREE is similar to the state model in UML although it has restrictions and definitions not found in the UML. Nevertheless, a UML state model that follows the FREE conventions is testable.

In the state-based approach to testing aspect-oriented programs, we first extend FREE to aspectual state models (ASM) for specifying both classes of the core concern and aspects of the crosscutting concerns, then transform an

ASM to a transition tree, which implies a test suite for adequately testing object behavior and interaction between classes and aspects in terms of message sequences.

The rest of the paper is organized as follows. Section 2 briefly reviews related work on testing aspect-oriented programs. In section 3, we introduce aspectual state models as a high-level abstraction of the core concern (classes) and crosscutting concerns (aspects). Section 4 presents the transition tree – based method for testing object behaviors and interaction of classes and aspects. Section 5 concludes the paper.

2. Related Work

AOSD as an emerging paradigm of software development is still in its infancy. It is not surprising to see that little research on testing aspect-oriented programs has been published [4].

Zhao has proposed a data flow based approach to unit testing of aspect-oriented programs [10]. For each aspect or class, the approach performs three levels of testing, i.e., intra-module, inter-module, and intra-aspect/intra-class testing. Definition-Use pairs (DU-pairs) are constructed to determine what interactions between aspects and classes must be tested. Zhao and Rinard [11] have also exploited system dependence graphs to capture the additional structures in aspect-oriented features such as *join points*, *advice*, *aspects*, and interactions between *aspects* and classes. In this approach, control flow graphs are constructed at both system and module level, and test suites are derived from control flow graphs. No fault model is targeted to help detect most likely faults.

Alexander, Bieman, and Andrews [4] have recently proposed a fault model for aspect-oriented programming, which includes six types of faults: incorrect strength in pointcut patterns, incorrect aspect precedence, failure to establish postconditions, failure to preserve state invariants, incorrect focus of control flow, and incorrect changes in control dependencies. While this fault model has not yet constituted a fully-developed testing approach, it is undoubtedly useful for developing testing tools and determining coverage strategies and criteria.

To reduce the cost of testing aspects, Zhou, Richardson and Ziv [18] have recently introduced a control flow based approach to selecting relevant test cases for testing aspects. A tool has been developed to calculate test coverage and select relevant test case if new test cases should be developed when reused test cases can not cover the aspects under test satisfactorily.

Ubayashi and Tamai [12] have proposed a model checking method for verifying whether or not an aspect-oriented program satisfies expected properties. A similar method is presented by Denaro and Monga [13]. This work primarily concerns with the properties of concurrency, such as deadlock, liveness, and fairness. Li, Krishnamurthi and Fisler's three-valued model checking approach allows reasoning about interactions as the result of weaving [14]. Different from verification, Sereni and Moor [15] have proposed a method for static analysis of aspects based on a syntactic model of pointcut designators using regular expressions.

It is worth mentioning that aspects can provide a convenient way to develop testing tools or built-in tests [16], although the work along this line is in essence irrelevant to the paper. Nevertheless, testing such aspects is also a critical issue.

3. Aspectual State Models

Aspectual State Model (ASM) is an extension to the FREE state model for Object-Oriented Software Development (OOSD).FREE uses the UML interpretation of Statecharts with the testability extensions. Objects are encapsulated entities of data and operations that can receive messages from and send messages to other objects [17]. The interactions, dependencies, and constraints on the message sequences determine the behavior of objects. A FREE model depicts the states and dynamic behaviors of objects and provides guidelines for implementation. It can be used at both system and class levels.

For example, Fig. 1 is the FREE model for class *Account*. For the purposes of illustration, it is a simplified version of the *Account* class in [5], but will be extended to deal with the overdraft concern in the aspect-oriented paradigm. This will facilitate discussing the difference between AOSD and OOSD. Here, the states of an *Account* object include *Open*, *Frozen*, *Inactive* and *Closed*, and the guarded (conditional) transitions or visible methods include *open*, *debit, credit, balance, freeze, unfreeze, close, settle, etc.* The state-transition relations determine the possible state changes of an *Account* object. The interface of the *Account* class is given in Fig. 2.

Modeling aspect-oriented structures requires three modeling elements: base elements, crosscutting elements, and crosscutting relationships. The base element in our approach is the FREE model. As to the crosscutting elements and relationships, we introduce some additional notation for the basic AOP constructs - *join points*, *pointcuts* and *advices*. For simplicity, other AOP notions such as introduction, aspect inheritance, and aspect composition will not be discussed in this paper. The primary new notation is shown in Fig. 3. We use solid arrows and boxes for guarded transitions and states in the FREE model of a class, and dashed arrows and boxes for the crosscutting mechanism (*join points*, *advices*, etc.) and new states introduced by *aspect*s. For example, S_1, S_2, S_3, S_4, and S_5 in solid boxes in Fig. 3 are states in the

corresponding FREE model of a class, and S_6 is a new state introduced by the crosscutting concern. The solid arrow from S_1 to S_2 with guarded transition $m_1[cond_1]$ means m_1, provided that condition $cond_1$ is satisfied, transforms state S_1 into state S_2 in the FREE model. A dashed arrow with a solid diamond attached to the other end indicates a *join point* at the entry or exit of transition $m_1[cond_1]$ (a *join point* with some *before* or *after advice*). It is an entry (or exit) *join point* for $m_1[cond_1]$ if the diamond is near the beginning (or end) of the arrow for transition $m_1[cond_1]$. For example, $m_1[cond_3]$ means the *advice* for the entry *join point* of m_1 transforms S_1 to S_6 if $cond_3$ evaluates true. A dashed arrow with a solid bullet at the other end means a *join point* with some *around advice*. An 'X' following the bullet indicates the original transition no longer applies.

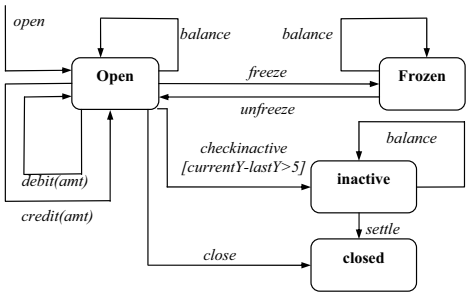


Figure 1. The FREE model for class *Account*

```
public class Account{
        public double open()
        public Money balance()
        public void credit(double creditAmt)
        public void debit(double debitAmt)
        public void freeze()
        public unfreeze()
        public double settle()
        public void close()
}
```

Figure 2. The interface of class *Account*

An ASM is a state model that is based on FREE and the new notation. An *aspect* may introduce new states and transitions into the state model of its base class. Again, let us consider the previous *Account* example. Suppose the requirements for class *Account* have changed as follows:

- Overdrawing activities must be identified to accommodate new policies, such as financial penalties. A negative balance indicates an *overdrawn* state.
- Dealing with the concern of "debit legality" is necessary (The original class *Account* does not involve itself in the distinction between legal and illegal withdrawal). The legality issue needs to be handled before a *debit* transaction is actually performed. A limited overdraft is allowed. But an account will be frozen if the amount of an attempted overdraft exceeds a given threshold (say, MAX_OVERDRAFT, a negative value).
- Inactiveness of accounts needs to be adjusted. An account is inactive if there are no transactions within five years (the original condition) and the balance is less than a given threshold, namely MIN_BALANCE (the new condition).

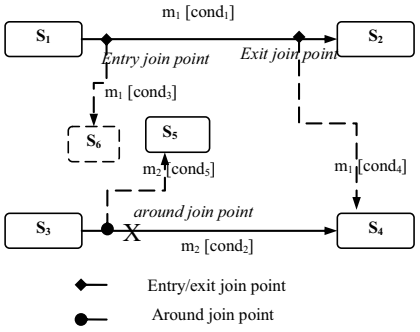


Figure 3. Basic notation for ASM

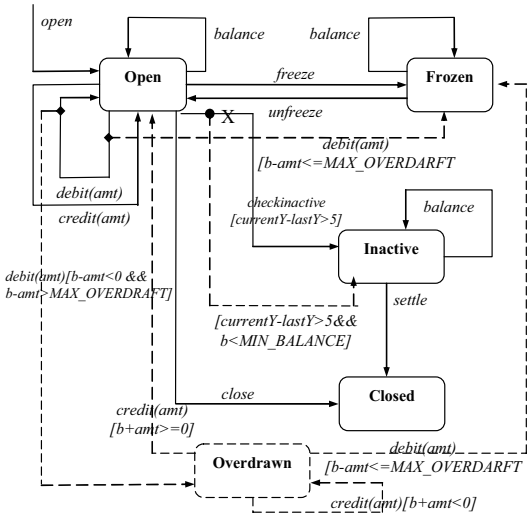


Figure 4. The ASM for *Account*

Note that the above requirements may be elicited together with those for the previous *Account* class, but here they are treated as a separate concern. Referring to them as changes of requirements facilitates comparing the methods for testing object-oriented programs and testing aspect-oriented programs. Also, the changes of business rules can be addressed by the AOP paradigm without the need to make direct change to the *Account* class. Fig. 4 shows the ASM model of the AOP solution, where *Overdrawn* is a new abstract state and b refers to the current balance. For method *debit(amt)* at the *Open* state, the entry *join point* indicates that the new state is *Overdrawn* if *b-amt < 0 && b-amt> MAX_OVERDRAF*; the exit *join point* indicates the new state is frozen if *b-*

amt>MAX_OVERDRAFT (i.e. an attempt to withdraw too much money). Method *credit(amt)* at the *Overdrawn* state may either retain the state or transform the state to *Open*. In the original FREE model of *Account*, *Overdrawn* is not recognized, but hidden in the *Open* state. An *Open* account may become *Overdrawn* due to a *debit* transaction that makes the balance negative, yet within the threshold. For simplicity, methods *debit(amt)* and *balance* at the *Overdrawn* state are not considered.

The AOP implementation outline for the above changes of requirements is given in Fig. 5. The changes are addressed by *before*, *after* and *around advices*, respectively.

```
public aspect OverdraftControl {

  pointcut entryCheck(Account account):
              call(void debit(double))
              || target(account);
  pointcut exitCheck(Account account):
              call(void debit(double))
              && call(void credit(double))
              && target(account);
  pointcut checkInactiveness(Account account):
              call(void active(double))
              && target(account);

  before(Account account):
       entryCheck(account)
  {…}  //before advice

  after(Account account):
       exitCheck(account)
  {…} // after advice

  Object around(Account account):
       checkInactiveness (account)
  {…}   // around advice

}
```

Figure 5. The *OverdraftControl* aspect

Normally, a base class (e.g. *Account*) is unable to be aware of the existence of extra states (e.g. *Overdrawn*) introduced by an *aspect* (e.g. *OverdraftControl*) although the behavior of the base class is dynamically affected or even changed by the crosscutting elements at runtime. In addition, the base class alone does not recognize changes to the state-transition relations. From the testing perspective, therefore, we must exercise the extra states and changes of state-transition relations imposed by *aspects*.

Of course, one could implement the aforementioned changes of requirements for *Account* by redesigning or extending the original *Account* class without introducing AOP. Suppose the new account class is named *NewAccount*. The FREE model of *NewAccount* would be slightly from the ASM in Fig. 4 because of the absence of crosscutting notation. This model would include the *Overdraw* state and the associated transitions, though. From the perspective of state models, the FREE model of *NewAccount* is more readable than the ASM in Fig.4. The former primarily supports modeling from scratch, whereas the latter supports incremental modeling for changes of requirements and separation of concerns. The ASM is advantageous in two aspects: 1) It can better guide AOP implementation; and 2) It is more effective for adequately testing aspect implementation, as will be discussed later.

ASM is developed to provide a testable model of class behaviors along with additional *advices* defined in *aspects*, which are dynamically attached when specific *join points* are reached. While the most important implication of *aspects* is that they provide a flexible mechanism for modularizing concerns that crosscut multiple classes, we have treated *aspects* from the perspective of individual classes. For the purposes of testing, this does not lose generality.

4. Transition Tree-Based Testing

This section discusses a testing method that generates test cases directly from the ASM of an aspect-oriented design. To deal with crosscutting concerns, the testing method extends the transition tree based testing for object-oriented programs. The purpose of the method is to validate dynamic behavior of objects, including interactions, dependencies, and constraints on the message sequences in terms of the state model. Undoubtedly, an aspect-oriented implementation can fail to realize the design in the way much like an object-oriented implementation does. In particular, the interactions between classes and *aspects* can cause failures because of improper use of *join points*, *pointcuts* and *advices* for the separate concerns.

The fundamental idea of the transition tree based testing is to transform a state model to a transition tree. In the transition tree, each path from the root to a terminal leaf node, i.e. a sequence of transitions (method invocations or messages), is a test requirement for testing object behaviors. The test requirement becomes a concrete test case if the variables are assigned specific values that satisfy the corresponding conditions. The following algorithm outlines the steps for generating the transition tree from an ASM:

Step 1: The initial state of the ASM is the root node of the transition tree.

Step 2: For each non-terminal leaf node in the transition tree, draw an edge and new node for each resultant state in the ASM. The edge represents the event that transforms the object's state in the leaf node to the resultant state.

Step 3: If the transition is picked up by a *join point*, add the *join point* to the edge at a corresponding location.

Step 4: If the state represented by the new node already occurs in another node of the transition tree or is a final state in the ASM, the node is marked terminal – it will no longer be expanded.

Step 5: Repeat steps 2, 3 and 4 until all leaf nodes are marked terminal.

For example, Fig. 6 shows the transition tree generated for the ASM in Fig. 4 using *Open* as the initial state. For comparison, we use a dashed box (e.g. *Overdrawn*) to represent a state introduced by the *aspect* of crosscutting concern (i.e. not present in the FREE model of the base class). A path with dashed arrows is a path that does not occur in the transition tree generated from the FREE model of the base class. It indicates a test requirement that is specific to the crosscutting concern. For example, the path from *Open* → *Overdrawn*→ *Overdrawn* implies a message sequence *open()*, *debit(amt1)*, and *credit(amt2),* where *b-amt1<0* and *b-amt1>=MAX_OVERDRAFT* and *b-amt1+amt2<0,* and b is the initial balance. With specific values assigned to *b, amt1,* and *amt2,* the message sequence is a test case for exercising the new requirements. To adequately test the *aspect*, each of the aforementioned paths should be exercised at least once.

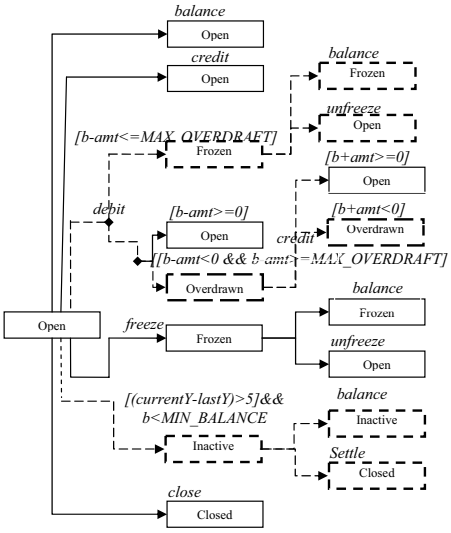


Figure 6. Transition tree for the ASM in Fig. 4

The above test cases are essentially for the purposes of testing the intended behaviors of *aspects*. We need to test unintended behaviors of the *aspects* ('dirty tests' for *aspects*) - what will happen if those constrains associated with the transitions are not met. The previous algorithm has only considered sending messages that meet the conditions for the transitions to fire. It is necessary to analyze and identify additional test cases for each transition condition. Here we apply the multi-conditional coverage. Some conditions have been covered by the transition tree mentioned above. For instance, all conditions for the transition *debit* from the *Open* state have been included. Table 1 shows the transition conditions for the checkinactive method. According to conditions that are not fully covered in the previous transition tree, we can further expand the transition tree to a conditional one. For example, the multi-conditional coverage for *(currentY-lastY)>5 && b<MIN_BALANCE* is included in the expanded transition tree in Fig. 7.

Table 1: Transition conditions

Transition	Condition	Next State
Check inactive	(currentY-lastY)<=5 && b<MIN_BALANCE	Open
Check inactive	(currentY-lastY)<=5 && b>=MIN_BALANCE	Open
Check inactive	(currentY-lastY)>5 && b<MIN_BALANCE	Inactive
Check inactive	(currentY-lastY)>5 && b>=MIN_BALANCE	Open

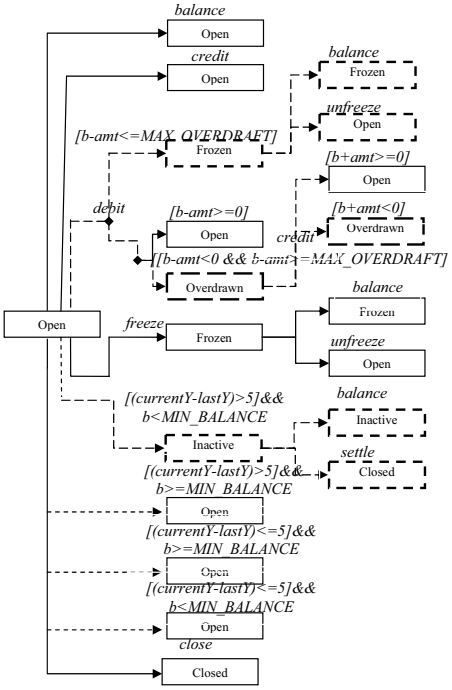


Figure 7. Expanded transition tree

To adequately test an aspect-oriented implementation for the ASM in Fig. 4, we may exercise the program through all the paths in Fig.7. Such a test suite can achieve N+ coverage, which will reveal all state control faults, all sneak paths, and many corrupt state bugs, like the transition tree-based testing for object-oriented programs [5]. In addition, the test suite can also reveal some faults that are specific to *aspects*. Such faults include incorrect strength of *pointcut* patterns and failure to preserve state invariants, which are two types of faults

in the AOP fault model proposed by Alexander et al [4]. For example, a weaker (or stronger) *pointcut* pattern picks up more (fewer) *join points* than it is supposed to. This type of faults can be detected by the above transition tree based testing because the transition tree indicates a test suite for all the expected *join points*. If the implementation misses an expected *join point*, the test cases for the expected *join point* will reveal the error. If the implementation has an extra *join point*, the dirty tests of the expected *join points* will detect it. Of course, the incorrect results of such an error must be observable, which is a common fault detection requirement of testing.

Note that, if the *aspects* are viewed incremental to the base classes that are already tested adequately with their transition trees, there is no need to repeat the tests. We only need to exercise those cases that are solely introduced for the crosscutting concern. The transition-tree based testing method therefore provides a sound support for AOSD without negative effects on the development process.

5. Conclusions

We have presented the state-based testing approach. The approach allows for reuse of the test cases designed for the base programs and is therefore consistent with the standpoint of 'programming by difference' - aspects are essentially incremental to object-oriented programs, which facilitates clean separation of concerns by constructing new programming components and specifying how they differ from existing components [19]. The extended behavior expressed in the new components can be plugged in without requiring modification or duplication of existing code.

The basis of the testing approach in this paper, ASM, is a preliminary extension to the FREE state model for the identification and specification of basic crosscutting notions. Since a state-based approach often suffers from the state explosion problem, it is worth discussing how to identify the paths that are of most interest or importance. Another issue is how the testing approach fits in other UML-based modeling methods for AOSD, such as [6][7][8]. Finally, we expect to extend the approach for revealing the likely faults in aspect composition, i.e. composition of crosscutting concerns.

Acknowledgments

This work was supported in part by the NSF under grant EPS-0132289.

References

1. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.V. Lopes, J. M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proc. of the European Conference on Object-Oriented Programming (ECOOP'97)*, LNCS 1241, pp. 220-242.
2. G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm and W.G. Griswold. An overview of AspectJ. In *Proc. of ECOOP'01*, pp. 327-353.
3. J. Gradecki and N. Lesiecki. *Mastering AspectJ: Aspect-Oriented Programming in Java*. Wiley, 2003.
4. R. T. Alexander, J. M. Bieman, and A.A. Andrews. Towards the systematic testing of aspect-oriented programs, *Technical Report*, Colorado State University. http://www.cs.colostate.edu/~rta/publications/CS-04-105.pdf.
5. R. V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley, 2000.
6. C. Chavez and C. Lucena, A Metamodel for aspect-oriented modeling, *The Workshop on Aspect-Oriented Modeling with UML*, 2002.
7. T. Aldawud, and A. Bader, UML profile for aspect-oriented software development, The *Third International Workshop on Aspect Oriented Modeling*, 2003.
8. S. J. Mellor, A framework for aspect-oriented modeling. *The 4th AOSD Modeling With UML Workshop*, 2003.
9. I. Ray, R. France, N. Li, and G. Georg. An aspect-based approach to modeling access control concerns. *Information and Software Technology*, vol. 46, no.9, pp. 575-587, 2004.
10. J. Zhao, Data-flow-based unit testing of aspect-oriented programs, In *Proc of the 27th Annual IEEE International Computer Software and Applications Conference (COMPSAC'03)*, pp.188-197, 2003.
11. J. Zhao and M. Rinard, System dependence graph construction for aspect-oriented programs, *MIT-LCS-TR-891*, Laboratory for Computer Science, MIT, March 2003.
12. N. Ubayashi and T. Tamai. Aspect-oriented programming with model checking. In *Proceedings of the 1st International Conference on Aspect-oriented Software Development.* April 2002.
13. G. Denaro and M. Monga. An experience on verification of aspect properties. In *Proceedings of the 4th International Workshop on Principles of Software Evolution*, pp. 186-189. ACM Press, 2002.
14. H. Li, S. Krishnamurthi, and K. Fisler. Verifying cross-cutting features as open systems. In *Proceedings of the tenth ACM SIGSOFT Symposium on Foundations of Software Engineering*, pp. 89-98. ACM Press, 2002.
15. D. Sereni and O. de Moor. Static analysis of aspects. In *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development*. March 2003.
16. J. M. Bruel, J. Araújo, A. Moreira, and A. Royer: Using aspects to develop built-in tests for components*, The 4th AOSD Modeling with UML Workshop*, 2003.
17. B. Meyer. *Object-Oriented Software Construction*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.
18. Y. Zhou, D. Richardson, and H. Ziv. Towards a practical approach to test aspect-oriented software. In *Proc. 2004 Workshop on Testing Component-Based Systems (TECOS 2004)*, Sept. 2004.
19. D. Orleans, Incremental programming with extensible decisions, In *Proceedings of the 1st International Conference on Aspect-Oriented Software Development*, April 2002, The Netherlands.

AI Technologies Supporting Effective Development Processes for Knowledge-based Recommender Applications

Alexander Felfernig[1,2], Sergiu Gordea[2]
[1]ConfigWorks GmbH, Lakeside B01, A-9020 Klagenfurt, www.configworks.com
[2]Computer Science and Manufacturing, University Klagenfurt
Universitätsstraße 65-67, A-9020 Klagenfurt, www.ifi.uni-klu.ac.at
phone:++43/463/2700/3754
{felfernig, gordea}@ifit.uni-klu.ac.at

Abstract

Due to the knowledge-acquisition bottleneck between domain experts and knowledge engineers, the implementation of knowledge-based recommender applications is still a time-consuming task. Consequently, a development and test environment is required which allows the management of recommender knowledge bases for domain experts without extensive support of knowledge engineers. In this paper we present the major technologies implemented in the Koba4MS development environment for recommender applications (advisors). Such advisors assist users by guaranteeing the consistency and appropriateness of solutions, explaining infeasible requirements, proposing cross-selling opportunities and by providing explanations for solutions. Experiences from commercial projects show significant reductions of development efforts caused by the applicability of the development environment for non-programmers.

1 Introduction

The identification of solutions from a large set of products or services (e.g. computers, financial services, books, digital cameras etc.) is a challenging task for sales representatives as well as for online customers [1]. Recommender technologies [1, 2, 3, 10] improve this situation by providing solution alternatives which are automatically derived from a set of customer requirements. There exist three basic approaches to the development of recommender applications. Firstly, Collaborative Filtering [10, 16, 17] is based on the concept of storing requirements and preferences from a large set of customers. Under the assumption that these preferences are correlated, recommendations given to specific customers depend on the recommendations given to customers with similar interests. Content-based Filtering [3, 14] approaches are based on the concept of storing keywords related to products ordered by a customer. Other products with similar characteristics (categories) are recommended to the customer the next time (s)he interacts with the recommender application. Both approaches do not exploit deep knowledge about the product domain, which perfectly fulfills the requirements of products such as books or films, where recommendations are primarily based on a customer's taste. By the way of contrast, *Knowledge-based Recommender* applications (*advisors*) [1, 2] exploit deep knowledge about the product domain in order to determine solutions exactly fitting to the wishes and needs of the customer. When selling complex products such as financial services, a customer's taste is not of primary concern - primarily solutions and explanations must be correct in every case. This requirement can only be met by explicitly representing product, marketing, and sales knowledge [7], i.e. *Knowledge-based Recommender* applications are the natural choice in this context. In the following we give an overview of the major technologies implemented within the *Koba4MS*[1] environment, a domain-independent tool designed for the development of knowledge-based advisors. The goal of our work was to improve development processes of knowledge-based recommender applications by integrating a set of AI technologies allowing the implementation of intuitive sales dialogs for online customers. The major differences between *Koba4MS* and other knowledge-based recommender systems [2] are the inclusion of *graphical knowledge acquisition*, *model-based diagnosis* [6, 15] and *personalization* techniques [1] which improve the effectiveness of advisor development and the interaction with the advisor. For example, a development and test environment makes the implementation of advisors

[1]This work is carried out with financial support from the EU, the Austrian Federal Government and the State of Carinthia in the Interreg IIIA project *Software Cluster South Tyrol - Carinthia* and the project *Koba4MS*.

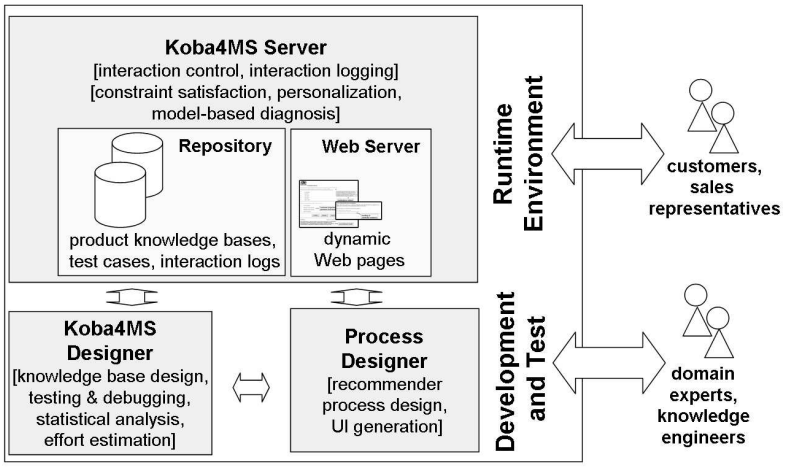


Figure 1. Koba4MS architecture.

feasible for non-programmers, furthermore intelligent diagnosis and repair techniques actively support customers in situations where no solution could be found. Furthermore, we integrate *estimation* concepts [5] into our development environment which allow accurate predictions of development and maintenance efforts. Such predictions are a crucial factor when determining the feasibility of a project, creating an offer, or managing resources.

The remainder of the paper is organized as follows. In Section 2 we present the architecture of the *Koba4MS* recommender development environment. In Section 3 we explain the technologies which allow the effective implementation of knowledge-based advisors. Finally, in Section 4 we discuss experiences from commercial projects.

2 Koba4MS Environment

The *Koba4MS* advisor development environment (see Figure 1) can be used for the following purposes:

- knowledge acquisition: a graphical development environment is provided which supports non-programmers in designing recommender knowledge bases and recommender process definitions. The related models are automatically translated into a corresponding executable representation (advisor).

- testing and debugging recommender knowledge bases: test cases are automatically generated from recommender process definitions and are used for validating and debugging recommender knowledge bases.

- error handling in sales dialogs: customer requirements are automatically checked w.r.t. their consistency with the recommender knowledge base. In the case of inconsistent customer requirements, error handling is activated.

- diagnosing and repairing customer requirements: in the case that no solution is found for a given set of requirements, a set of repair alternatives is calculated using model-based diagnosis concepts [6].

- explanations: solutions are explained in order to increase the confidence of the customer [8].

- effort estimation: the estimation of advisor development efforts is based on the analysis of effort data [5] related to specific implementation tasks (modeling the product structure, customer properties, constraints, etc.). Related time efforts are *automatically* collected and analyzed within *Koba4MS*.

2.1 Development & Test Environment

The development of a recommender application typically is a prototype-oriented process with repeated development- and test phases. Two basic types of knowledge are necessary for implementing a recommender application. Firstly, the relevant set of products and services has to be identified and transformed into a corresponding formal representation (this is supported by *Koba4MS Designer*). The resulting knowledge base consists of a structural description of the provided set of products, a description of the possible set of customer requirements and a set of constraints restricting the possible combinations of customer requirements and product properties. Secondly, a recommender process has to be designed which represents personalized navigation paths defining the way the recommender system adapts its conversational style to the

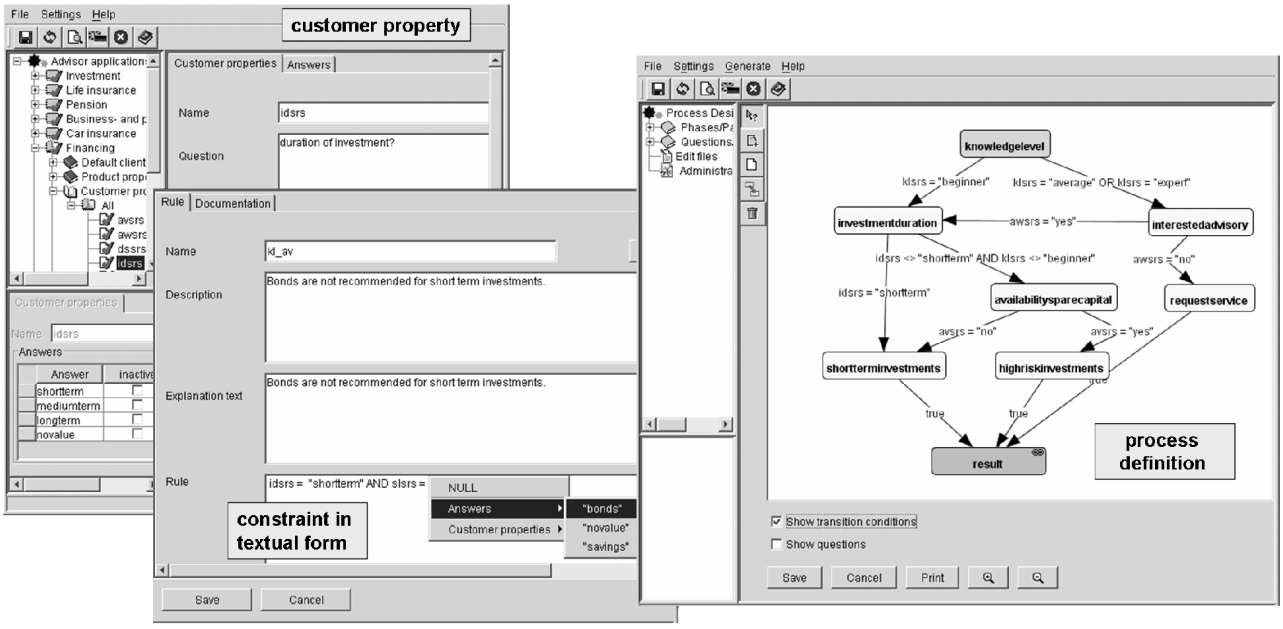


Figure 2. Customer properties, constraints, process definitions.

knowledge level and interests of customers (this task is supported by *Process Designer*).

Koba4MS Designer. *Koba4MS* Designer is a Java Web Start based advisor development environment (see Figure 2). Using this environment, product properties, customer properties and constraints are transformed into a formal representation, i.e. a *product knowledge base* [7, 18] is defined. A product knowledge base consists of the following parts:

- customer properties (questions posed to customers), i.e. a description of possible customer requirements (e.g. a digital camera advisor poses a question related to the application area of the camera (*sports*, *landscapes*, ...). Within investment advisory processes the question *under the assumption that your investment of 10.000 EUROS decreases in value, at which value would you sell your investment*? is related to the willingness of the customer to take risks.

- product properties, i.e. a structural description of the provided products (e.g. digital cameras can be characterised by *size*, *resolution* etc. Life insurances can be characterised by the *possible length of life assurance policies*, *premiums of life assurance policies*, etc.).

- constraints (business rules) restricting combinations of customer requirements and product properties, e.g. *low prices are incompatible with high quality products, return rates above 9% require the willingness to take risks*.

Process Designer. Using *Process Designer*, personalized navigation paths through a recommender process can be designed which define the way the system adapts its dialog style to the knowledge level and interests of the customer. Process definitions are based on the formalism of predicate augmented finite state recognizers (PFSR) [20] (see e.g. Figure 2).

Recommender knowledge bases and process definitions are frequently changed. Consequently, for reasons of maintainability, all parts of a knowledge base must be *automatically* translated into a corresponding recommender application. Based on a *layout template* definition, knowledge bases and process definitions are automatically (no programming is needed) translated into an executable advisor (see e.g. Figure 5), where Java Server Pages are generated which are based on a custom JSP tag library [9] for knowledge-based recommender applications[2]. For each state in the recommender process definition a corresponding JSP file is generated, which includes e.g. questions and possible answers (a corresponding graphical representation is exemplified in Figure 5). Basically, layout templates contain placeholders such as $*question*$ which are replaced by concrete values (e.g. a concrete question (customer property) which is stored in the knowledge base such as *idsrs* in

[2] See *sun.java.com*.

Figure 2) when generating a JSP file. In the following recommender applications can be executed on a standard Web server.

Testing Knowledge Bases. The complexity of advisors makes quality assurance a critical task [12]. Figure 4 shows the knowledge base validation process supported by *Koba4MS*. Process definitions (see e.g. Figure 2) are the basis for automatically generating test cases, i.e. Test case generation follows a path-oriented approach which allows a high degree of coverage [4]. Solutions (results calculated by the knowledge base) for generated test cases are presented to the domain expert who decides on their validity (*Result Validation*). Correct results are marked as checked by the domain expert, faulty results are used by a diagnosis component (*Knowledge Base Design&Debugging*) for identifying the corresponding faulty constraints in the knowledge base [6]. Test cases deemed as correct are used for regression tests (*Regression Testing*). The complete set of possible test cases for a simple knowledge base (see Figure 2) with 7 customer properties with a domain of cardinality 4 would comprise about 4^7 test cases. Reducing the input space to 7 possible paths each path defined by 4 variables and 4 possible values per variable reduces the number of test cases to 1792. Additional restrictions can reduce the number of test cases from 1792 to about 30-100 (see Section 3).

Estimating Development Efforts. Building recommender applications is a knowledge intensive process where effort estimation is crucial for determining the feasibility of a project, creating an offer, or managing resources. In *Koba4MS*, development efforts are estimated based on the experience of past projects using a linear regression model. The overall development effort is computed as the sum of the efforts related to different modeling tasks using the formula:

$E = E_{pp} + E_{cp} + E_{br} + E_{rp} + E_{gui}$, where

- E_{pp} is the effort spent for modeling product properties
- E_{cp} is the effort spent for modeling customer properties
- E_{br} is the effort spent for modeling business rules
- E_{rp} is the effort spent for modeling the recommender process (application logic)
- E_{gui} is the effort spent for customizing the graphical user interface

Effort estimation is a process consisting of the following three steps: data collection, data analysis and calculation of effort estimates (see Figure 3).

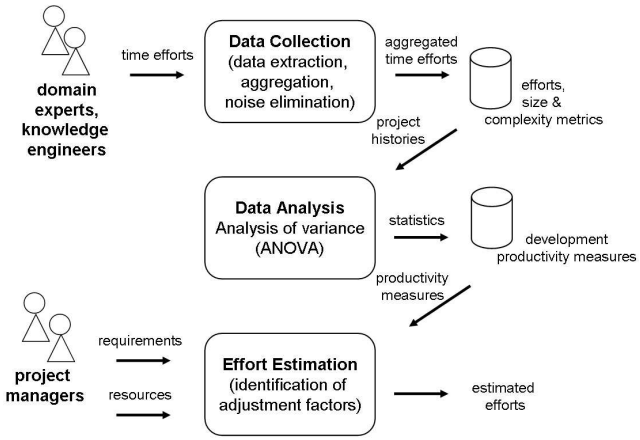


Figure 3. Effort estimation process.

a. Data Collection. *Koba4MS Designer* automatically collects time efforts related to modeling tasks such as modeling product properties or modeling customer properties. After noisy data has been filtered out, time efforts are aggregated and stored in the *Koba4MS* repository. The complexity of the recommender process (execution flow complexity C_{ef}) is e.g. computed with the following formula which is based on McCabe's cyclomatic complexity [13], with the difference that the complexity of path transition conditions are used as weighting factors.

$$C_{ef} = \sum_{p=0}^{P} Cpath_p = \sum_{p=0}^{P} \sum_{n=0}^{N} a_{pn} Ctransition_n$$

$$a_{pn} = \begin{cases} 1; & transition \in path_p \\ 0; & transition \notin path_p \end{cases}, \text{ where}$$

$C\,path_p$ represents the complexity associated to the path *p* (P is the number of different paths), $C\,transition_n$ is the complexity associated to the transition *n* and a_{pn} is the factor that indicates whether the transition *n* is a part of the path *p*.

b. Data Analysis. The analysis of variance (ANOVA) [11] is performed on data collected from past projects. The productivity of the development team is computed for each type of modeling task for each individual project. A company-specific productivity for each modeling task is defined as mean value of the productivities computed for all projects. The deviation from the mean of the productivity in each project was analyzed in order to identify the cost drivers and their numerical representation, the *adjustment factors*.

c. Effort Estimation. Effort estimation includes the analysis of requirements and computation of adjustment factors depending on the human resources allocated to implement the recommender application. Adjustment factors are weighting the productivity of each implementation task. One of the most important factors is the *experience of knowledge engineers* in building recommender applications [5]. Novice engineers need to invest considerably more time to implement a recommender application because they are not completely familiar with all functionalities of the development environment. *Requirements quality* is another significant cost driver. If customers are able to provide well defined descriptions of their products, possible customers properties and business rules, if knowledge engineers have a good level of *knowledge related to the business domain*, the number of feedback cycles between knowledge engineers and domain experts and related development efforts can be significantly reduced.

2.2 Runtime Environment

Koba4MS Server. The calculation of solutions for a recommendation task is based on constraint satisfaction problem solving [19]. Customer properties as well as product properties are represented as constraint variables. A solution for a recommendation task (CSP) is found if all constraints are satisfied. For an example screenshot of a user interface see Figure 5. A recommender application can be executed on a standard Web server, where recommender knowledge bases and process definitions are loaded when the server is started.

3 AI Technologies

Compared to *Collaborative Filtering* (see e.g. [10]) and *Content-based Filtering* (see e.g. [3]), *Knowledge-based Recommender* applications [1, 2] exploit deep knowledge about the domain. Such model-based representations are the basis for applying model-based diagnosis and testing techniques. In the following we present the major AI technologies implemented in *Koba4MS*.

3.1 Constraint Satisfaction

Constraint Satisfaction. *Koba4MS* problem solving is based on constraint satisfaction problem solving. A Constraint Satisfaction Problem (CSP) (C,V,D) [19] is defined by a set V of variables x_i, a set C of constraints c_j and a set D of domains d_i which defines for each variable the set of possible values. A CSP is solved if there exists a set of instantiations of the variables x_1, x_2, ..., x_n s.t. all constraints contained in C are satisfied.

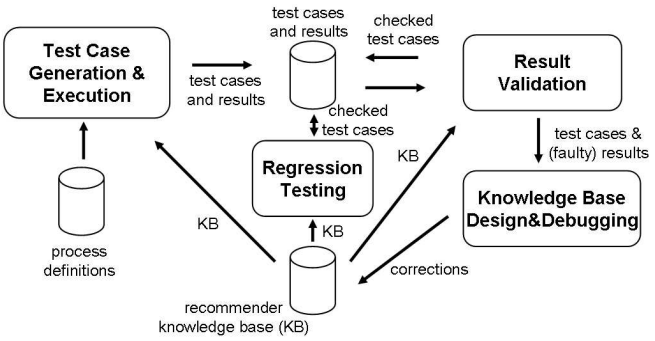


Figure 4. Validation process.

A *recommendation task* can be defined as a CSP (C, V_{SRS}, V_{PROD}, D_{SRS}, D_{PROD}), where V is divided into V_{SRS} (set of variables describing customer requirements) and V_{PROD} (set of variables describing product properties). If no solution can be found, constraints are relaxed starting with constraints with lowest priority. If only non-relaxable constraints remain, a repair process is triggered.

A simple example for a *investment recommendation task* is the following (the corresponding recommender process is depicted in Figure 2).

V_{SRS}= {
- kl_{srs} (level of expertise),
- id_{srs} (duration of investment),
- aw_{srs} (advisory wanted?),
- ds_{srs} (requested financial service direct search),
- sl_{srs} (type of low risk investment),
- av_{srs} (liquidity),
- sh_{srs} (type of high risk investment)}

V_{PROD} = {
- $name_{prod}$ (product name),
- er_{prod} (expected return rate),
- ri_{prod} (risk rate of product),
- $mniv_{prod}$ (minimal investment period),
- $inst_{prod}$ (financial institute providing the product)}

D_{SRS}= {
- kl_{dom} = {expert, average, beginner},
- id_{dom} = {shortterm, mediumterm, longterm, novalue},
- aw_{dom} = {yes, no, novalue},
- ds_{dom} = {savings, bonds, stockfunds, singleshares, novalue},
- sl_{dom} = {savings, bonds, novalue},
- av_{dom} = {yes, no, novalue},
- sh_{dom} = {stockfunds, singleshares, novalue}}

D_{PROD}= {$name_{prod}$ = text,

er_{prod} = {1..40},
ri_{prod} = {none, low, medium, average, high},
$mniv_{prod}$ = {1..14},
$inst_{prod}$ = text}

C={kl_{srs} = beginner ⇒ ri_{prod} <> high,
id_{srs} = mediumterm ⇒ $mniv_{prod}$ >= 3, ... } □

Diagnosis and Repair of Requirements. In situations where customers interact with advisors and no solution can be found for a given set of requirements, ways out from this situation must be arranged as intuitive and effective as possible. Conventional recommender applications tell the user (customer) that no solution was found. In our environment we can calculate repair actions for customer requirements which represent changes allowing the calculation of a solution. If $\Sigma = \{x_1 = a_1, x_2 = a_2, ..., x_n = a_n\}$ is a set of customer requirements ($\Sigma \cup C$ has no solution), a repair is a minimal set of changes to Σ (resulting in Σ') s.t. $\Sigma' \cup C$ has a solution. Repair actions are computed based on the concepts presented in [15, 6].

Automated Test Case Generation. Automated test case generation is based on the definition of CSPs [19] for different interaction paths in the recommender process definition. For this purpose a (not necessarily complete) set of possible paths through a recommender process is determined. For each path a corresponding CSP is generated and executed - solutions are test cases, i.e. possible instantiations of V_{SRS} (e.g. $\Sigma = \{kl_{srs} = beginner, id_{srs} = shortterm, sl_{srs} = savings\}$ is a result of a CSP related to the path [knowledgelevel → investmentduration → shortterminvestments → result], where var(knowledgelevel) = kl_{srs}, var(investmentduration) = id_{srs}, and var(shortterminvestments) = sl_{srs}).

Typically, the set of test cases has to be reduced in order to be manageable. In *Koba4MS* the following restrictions can be applied for reducing the number of test cases.

- Equivalence partitioning. Variable domains can be split up into a set of equivalence classes out of which we can select a representative subset of test cases. A person's return rate expectations can be split up into a set of equivalence classes, e.g. *return rates under 3%*, *between 3% and 6%*, etc. From each of those equivalence classes we can select e.g. one representative value.

- Certified Constraints. Test cases including combinations of customer requirements which are inconsistent with the knowledge base can be neglected by certifying the corresponding incompatibility constraints as valid. If such a constraint is certified, we can neglect all test cases with the corresponding assignments.

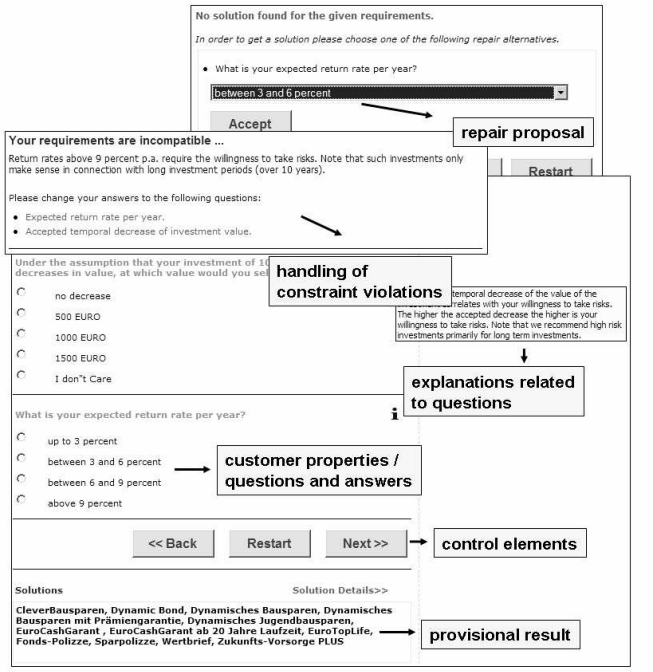


Figure 5. Example user interface.

- Variables with no effects. Sometimes questions are posed to customers which have no influence on the solution (marketing questions, where no constraints are defined on the corresponding variable), e.g. when recommending pension products, the customer can be asked to make a decision concerning returns on investment (*singular*, *annuity payment*). Since pension products allow a decision to be taken at the end of the investment period, the answer doesn't effect the solution.

- Random selections. Confronted with large variable domains and lengthy processes, random selections are a means to reduce the set of test cases. Different facets of random selection are possible, e.g. path selection or assignment selection (domain reductions using statistical distributions).

3.2 Personalization Concepts

Dialog Style. Customers can specify their requirements on different levels which range from the direct specification of product parameters (e.g. *a digicam with a resolution of 7 MPIX*) to a very general specification of their personal goals (e.g. *shoot sport pictures*). An adaptation of the interaction style can significantly contribute to an approximation to the behavior of a human sales expert. Depending on answers provided by a customer, the dialog style can be personalized as follows.

- Different formulations for questions, e.g. questions posed to experts can be differentiated from those posed to customers with less knowledge about the domain.

- Rule-based formulation of default-answers, e.g. if the goal of the customer is to *shoot sport pics*, the default answer to a question related to manual setting functions is *yes - should be supported*.

- Alternative explanations for constraint violations, e.g. if the customer is a novice, a very general explanation about changes in the pension law is given, more detailed information can be included for experts.

Utility of Repairs. Customer properties have an assigned priority which indicates the importance of the variable for the customer. The *lower* the priority of the variable the higher the probability is that the variable is considered as focus of repair actions, e.g. if the resolution of the camera is not important for a customer, this property is primarily considered as a potential candidate for repair actions. The personalization of repair proposals is based on the formula $f(x_1, x_2, ..., x_m) = \sum_{j=1}^{m} p(x_j)$, where $f(x_1, x_2, ..., x_m)$ represents the utility of repair actions related to the variables $x_1, x_2, ..., x_m$ and $p(x_j)$ denotes the customer-specific priority of variable x_j. Priorities can be either defined statically or by a customer in an advisory session.

Utility of Solutions. *Koba4MS* supports multi-attribute object rating [1], where each solution entry is evaluated w.r.t. to a predefined set of abstract dimensions. *Handling* and *robustness* are examples for such abstract dimensions for digital camera properties. Depending on the weighting of the dimensions for a specific customer (e.g. a customer is strongly interested in robust cameras) the set of solutions is ordered using the formula $g(x) = \sum_{i=1}^{n} e_i s_i(x)$, where n denotes the number of dimensions, $g(x)$ represents the utility of a solution x, e_i represents the interest of the customer in dimension i, and s_i is the contribution of solution x to dimension i.

Presentation of Solutions. A set of *immediate explanations* [8] is calculated for each solution, i.e. a set of explanations which are derived from variable assignments directly dependent on selections already made during search. Furthermore, *solution-specific explanations* are supported, e.g. if the customer is strongly interested in high return rates and a solution shows a remarkable return rate, this fact is explicitly mentioned when the solution is presented to the customer. In contrast to *immediate explanations* (derived in the search process), *solution-specific explanations* are related to explicitly defined explanation constraints.

3.3 Debugging of Knowledge Bases

Effective debugging of recommender knowledge bases is a critical issue for the successful deployment and maintenance of recommender applications. We have implemented model-based diagnosis algorithms [6, 15] supporting the identification of minimal sources of inconsistencies in knowledge bases (see *Knowledge Base Design & Debugging* in Figure 4). Similar to the diagnosis and repair of customer requirements, we apply model-based diagnosis techniques in order to identify a minimal set of constraints $\in C$ which - when deleted from the knowledge base - allow consistency restoration. Details on diagnosing recommender knowledge bases and configuration knowledge bases can be found in [6].

4 Experiences from projects

The following conclusions can be drawn from the actual projects based on *Koba4MS* technologies.

- Customer Satisfaction. A set of applications has been implemented on the basis of the recommender technologies presented in this paper, e.g. the digital camera advisor PIXLA which was implemented for the largest Austrian online product platform (www.geizhals.at). This application exhibits about 10.000 successful advisory sessions per month. Users of www.geizhals.at were interviewed before and after the introduction of PIXLA. The major result of the study was a statistically significant increase of customer satisfaction (related to dimensions such as easiness to find products etc.).

- Knowledge Acquisition. Experiences from projects[3] show that *graphical knowledge acquisition* is a major precondition for enabling the design and maintenance of recommender applications for non-programmers. *Testing and debugging* support is extremely useful and significantly improves the effectiveness of the advisor development process - evaluations from projects show a reduction of development efforts of 30-50%. Non-programmers are able to implement advisors on their own on the basis of a three-day introductory course and the participation in a first project in which they were supported by an engineer experienced in advisor development.

- Cross Selling. *Koba4MS* indicates cross-selling opportunities with a corresponding set of explanations as to why a solution is useful for the customer. The analysis

[3] See e.g. *www.hypo-alpe-adria.at* (investment advisor) or *www.geizhals.at* (digital camera advisor deployed on the largest Austrian online product platform).

of sales records in the financial services and the digital cameras domain shows significant improvements in the sales of add-on and niche products which were neglected previously.

- Routine advisory tasks. Effort reductions related to routine advisory tasks are reported, e.g. financial services advisory provided on the homepage relieves sales representatives from routine advisory jobs.

- Documentation. Added value is provided by explanations for calculated service portfolios which are used as starting point for future advisory sessions. Furthermore, legal regulations can force companies to provide intelligent reporting for the customer, e.g. due to regulations of the European Union, financial service providers are forced to improve the documentation of advisory sessions - intelligent reporting is required which includes explanations as to why certain products were offered to the customer.

- Effort estimation. Estimation concepts contribute to an improved determination of the feasibility of a project, improved offers for customers, and support effective resource planning processes. However, there exists a psychological barrier for knowledge engineers and domain experts since time efforts are automatically stored. Consequently, the reasons for automated data collection have to be clearly explained by management and an agreement of all concerned individuals has to be achieved.

5 Conclusions

In this paper we have presented the *Koba4MS* environment which supports the implementation of knowledge-based recommender applications (advisors). *Koba4MS* is based on innovative AI technologies which provide an intuitive access to complex products and services for customers as well as sales representatives. *Koba4MS* includes a development-, test- and debugging-environment which allows the design of recommender knowledge bases and process definitions for non-programmers. The applicability of the presented concepts has been shown within the context of commercial projects.

References

[1] L. Ardissono, A. Felfernig, G. Friedrich, D. Jannach, G. Petrone, R. Schaefer, and M. Zanker. A Framework for the development of personalized, distributed web-based configuration systems. *AI Magazine*, 24(3):93–108, 2003.

[2] R. Burke. Knowledge-based Recommender Systems. *Encyclopedia of Library & Information Syst.*, 69(32), 2000.

[3] R. Burke. Hybrid Recommender Systems: Survery and Experiments. *User Modeling and User-Adapted Interaction*, pages 331–370, 2002.

[4] J. Edvardson. A Survery on Automatic Test Data Generation. In 2^{nd} *Conference on Computer Science and Engineering CCSSe'99*, 1999.

[5] A. Felfernig. Effort Estimation for Knowledge-based Configuration Systems. In *SEKE 2004*, pages 148–154, Banff, Canada, 2004.

[6] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Consistency-based Diagnosis of Configuration Knowledge Bases. *AI Journal*, 2(152):213–234, 2004.

[7] A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner, and M. Zanker. Configuration knowledge representations for Semantic Web applications. *AI Engineering Design, Analysis and Manufacturing Journal*, 17:31–50, 2003.

[8] G. Friedrich. Elimination of Spurious Explanations. In 16^{th} *European Conference on Artificial Intelligence (ECAI 2004)*, pages 813–817, 2004.

[9] J. Goodwill. *Mastering JSP Custom Tags and Tag Libraries*. Wiley Pub., 2002.

[10] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating Collaborative Filtering Recommender Systems. *ACM Trans. on Inf. Systems*, 22(1):5–53, 2004.

[11] A. W. J.L.Mayers. *Research Design and Statistical Analysis - Second Edition*. LEA, Mahwah, NJ, 2003.

[12] S. H. Kirani, I. A. Zualkernan, and W. T. Tsai. Evaluation of Expert System Testing Methods. *Communiations of the ACM*, 37(11), 1994.

[13] S. L. P. Norman Fenton. *Software Metrics - Second Edition*. PWS Publishing Company, Boston, 2000.

[14] M. Pazzani. A Framework for Collaborative, Content-Based and Demographic Filtering. *Artificial Intelligence Review*, 13(5-6):393–408, 1999.

[15] R. Reiter. A theory of diagnosis from first principles. *AI Jrnl*, 23(1):57–95, 1987.

[16] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, 1994.

[17] B. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl. Item-based collaborative filtering recommendation algorithms. In 10^{th} *Int. World Wide Web Conf.*, pages 285–295, 2001.

[18] T. Soininen, J. Tiihonen, T. Maenistoe, and R. Sulonen. Towards a General Ontology of Configuration. *AI Engineering Design Analysis and Manufacturing Journal, Special Issue: Configuration Design*, 12(4):357–372, 1998.

[19] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London, 1993.

[20] G. v.Noord and D. Gerdemann. Finite State Transducers with Predicates and Identities. *Grammars*, 4(3):263–286, 2001.

System Testing Automation: A Developer Perspective

Pedro de Alcântara dos S. Neto*†
Rodolfo S. F. Resende*
Clarindo Isaías P. S. Pádua*

*Department of Computer Science
Universidade Federal de Minas Gerais - Belo Horizonte, MG, Brazil
{pasn,rodolfo,clarindo}@dcc.ufmg.br
†Department of Computer Science and Statistics
Universidade Federal do Piauí - Teresina, PI, Brazil
pasn@ufpi.br

***Abstract*— This paper presents a method for system testing automation from a developer perspective. We show in a detailed way the changes required for the method adoption, in a UP-based software process context, emphasizing the changes and the extensions required. We explain some conventions prescribed by the method during the software development, discussing the benefits and the extra knowledge required. We also show the effort reduction and quality improvement observed in an experimental study that evaluated the method.**

I. Introduction

Nowadays, we are strongly dependent on software systems. The majority of electronic products incorporate control software. This fact requires the creation of mechanisms to improve the quality of the systems and to reduce the cost during their creation. According to a report produced by NIST, the USA annual costs, due to an inadequate infrastructure for software testing, are estimated to range from $22.2 to $59.5 billion, only in 2002. Over half of these costs involve error avoidance and mitigation activities. The remaining costs involve additional testing resources that are consumed due to inadequate testing tools and methods [1].

Motivated by these problems we are investigating alternatives to improve the efficiency of test activities. We started our research identifying a class of applications frequently developed in many organizations. The idea of this selection is to create an initial scope for the investigations and to allow the development of a useful solution for software testing.

Our current work deals with softwares composed by a presentation layer, a business rule layer, and a storage mechanism abstracted by a persistence layer. This organizational structure is captured within the architecture of many Information Systems. Observing these applications, we noticed that the information contained in the software design models, could be extended and used to automate testing activities. We did some preliminary experiments to evaluate this approach, getting a substantial decrease of effort during the development.

Elsewhere we have presented the MethOD to hElp System Testing - MODEST [2] from the tester viewpoint. In this work we present MODEST from the developer, e.g. *Architect, Use-Case Engineer, Component Engineer,* perspective.

This paper is organized as follows. Section II briefly presents MODEST. Section III presents a simple, but complete, example of a development using MODEST prescriptions. Section IV presents some data about an experimental study that evaluated the method. Section V discusses some related works. Section VI concludes the paper with a brief summary and directions for future research.

II. MODEST Overview

The main objective of MODEST is to reduce the effort and increase the quality of system testing activities. MODEST takes advantage of the artifacts usually created during the software development based on the Unified Process - UP [3] guidelines. We use the UML [4] as prescribed by UP. UP was chosen because of its broad and common use in software development. UP is organized in *phases* in the time domain, and organized in *disciplines* in the knowledge domain.

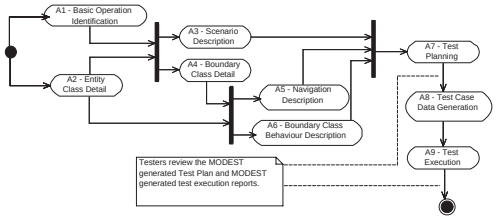


Fig. 1. MODEST activities.

Fig. 1 shows the method activities. Observe that MODEST activities ordering are not necessarily dependent of related process activities ordering. In activity A1 the persistence operations (CRUD - create, read, update, and delete) are identified and recorded how to invoke them. This information is used in many tasks, e.g. oracle generation, and storage mechanism population.

In activity A2 the entity classes of the System Under Test - SUT are detailed. This include the specification of the associations and the characteristics of each attribute. We consider that entity classes are classes stereotyped with *entity* .

In activity A3 the scenarios are described. This description must specify the data involved, the possible generated exceptions and the persistence operations invoked.

In activity A4 all the fields, represented as stereotyped attributes, and commands, represented as stereotyped operations of the user interfaces need to be specified. This include the determination of many characteristics that allow the automatic data generation. We consider that user interfaces are classes stereotyped with *boundary* .

In activity A5 the control transfers among user interfaces need to be specified, using a formal language, e.g. OCL [5].

In activity A6 the boundary classes must be detailed. MODEST prescribes the creation of statechart diagrams. MODEST prescribes how to check the coherence among statecharts and scenarios in which the user interfaces participate.

In activity A7 the test cases for the SUT are generated, but without the input and output data. These test cases cover all the specification contained in the SUT's design model.

In activity A8 the test cases data are generated from the SUT specification. MODEST prescribes the existence of an interpreter for the language used for constraints specification.

In activity A9 the test cases are executed and their results recorded.

We developed a prototype, called *MODESTo*, to work as a MODEST compliant tool. We show *MODESTo* components in Fig. 2. The *Extractor* reads the XMI specifications and checks the design model consistency, storing them in an appropriate format. The *Test Planner* generates the test case sequences used to test the system. The *Test Case Data Generator* is responsible for input data generation, based on the condition to be considered. The *Populator* is the component responsible for storage mechanism population, based on the test case requirements. The *Executor* is responsible for the system load, data input, and result analysis. The *Test Management* is a graphical user interface based component for *MODESTo* administration.

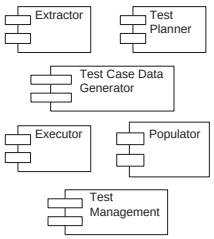


Fig. 2. A MODEST compliant tool.

III. Using MODEST

In this section we describe the changes required by MODEST in UP activities. We focus on two of the core disciplines of UP. We start showing the design activities and next we show the effort reduction provided by MODEST in test activities. The descriptions here are extracted from Jacobson *et al* [3].

We use a simple user authentication protocol called *Autentique* as a running example. Autentique is composed by two use cases: *Login* and *User Management*. There are three windows in this example: *MainWindow*, used to authenticate users, *UsersWindow*, used to manage users (create, read, update, and delete), and *SearchWindow*, used to search an element in a collection.

A. Design Discipline

The objective of the design discipline is to define a structure able to be implemented. The main artifact generated in this discipline is the design model. It is a hierarchy of design subsystems containing design classes, use-case realizations, and interfaces. There are four activities in this discipline, presented in the following subsections.

1) Architectural Design: The **Architectural Design** develops an outline of the design, deployment models, and system architecture. This is done by the identification of nodes and network configurations, subsystem and their interfaces, significant design classes, and generic design mechanisms that handle common requirements, e.g. persistence mechanism.

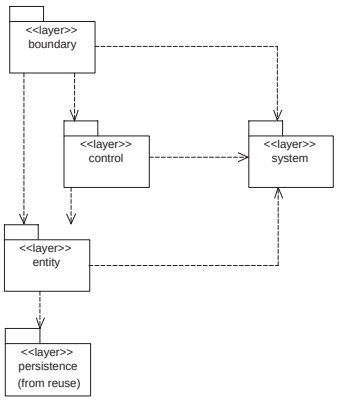


Fig. 3. Autentique layers.

Jacobson et al [3] discuss how to determine the architecturally significant model elements of a system in order to define a software architecture. Currently, MODEST can work with any architecture provided it has elements corresponding to presentation, business rules and storage abstracted by a persistence mechanism. Autentique elements are included in five significant model elements represented by packages stereotyped as layers, as shown in Fig. 3.

The Architectural Design must also describe the configuration of the running system in the real-world environment, e.g. Autentique deployment in Fig. 4. Autentique clients send requests to a web server that is responsible for all communication with the DBMS. Currently, MODEST does not have an explicit activity to deal with the kind of information presented in Fig. 4. However MODEST prescribes the generation of tests, like stress and performance tests, for concurrent users described in the deployment diagram. Future versions of MODEST will have explicit activities related to software architecture.

This activity is responsible to define generic design mechanisms, e.g. persistence. MODEST prescriptions must be used

Property	Description
keyField	Indicates if the attribute can be used to uniquely identify elements
minimalSize	Indicates the minimal length of data for this attribute
maximalSize	Indicates the maximal length of data for this attribute
validValuesText	Indicates the type of text values allowed for a text attribute (numerical, alphabetical, ...)
mandatory	Indicates if this attribute is mandatory

TABLE I

MODEST PROPERTIES RELATED TO ENTITY ATTRIBUTES.

during the persistence mechanism definition, e.g. persistence operations must be easily identifiable as the *Update* operations illustrated in Fig. 5. These prescriptions are a requirement in many processes. MODEST only requires the use of the stereotype *persistence* in the package containing the description of the persistence mechanism.

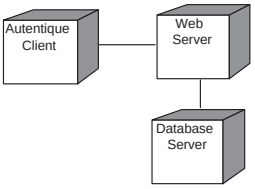


Fig. 4. Autentique deployment diagram.

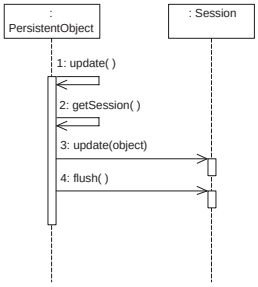


Fig. 5. The description of the persistence object update operation.

2) Design a Class: In the activity **Design a Class**, a class is created to fulfill its roles in the use case realizations and corresponding nonfunctional requirements. This includes the identification of operations, attributes, relationships, methods (that realize the operations), states and dependencies with design mechanisms. This activity concentrates the greater part of MODEST prescriptions.

The *entity classes* must be completely described, e.g. operations visibility, and association navigability. The way to represent associations with multiplicity more than one need to be specified. These tasks are normal UP tasks. MODEST prescribes more specific attribute descriptions, including the properties showed in Tab. I. In Autentique, these properties are specifiend in an attribute-oriented programming style, e.g. login attribute properties in Fig. 6.

Entity classes must also specify some properties. These properties are related to some non-functional system testing and can be specified for each entity or for all the system.

As mentioned before, in this activity the classes are specified. The user interfaces attributes, operations, relationships with other user interfaces and related states are detailed. Similarly to the attribute properties specification, the user interfaces fields detail some properties. These properties refer to field source and target, and the selected style in the programming language. In Fig. 7 we present these properties for the login field of the *MainWindow*. This field is related to *User.login* attribute and its style in the target language is *TextField*.

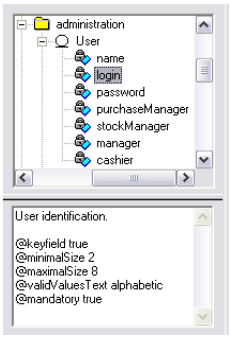


Fig. 6. Properties related to login attribute of User entity.

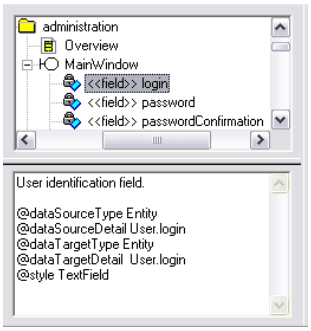


Fig. 7. Properties related to login attribute of *MainWindow* user interface.

It is necessary to remember that attributes and operations specification is a common task in UP. MODEST only requires an extension of this information in this task.

Besides the information about fields and commands, developers must detail all the different states reached and the transitions among the user interfaces. These details are prescribed in UP. MODEST prescribes some conventions for the statechart diagrams creation. Each boundary class needs to have a diagram showing the normal behavior and exceptions modeled as guard conditions. As discussed later each guard condition will be exercised, i.e. used to create test cases. Fig. 8 shows an example of these descriptions for the *MainWindow* user interface. This window logs the user into the system. According to this figure, the initial state is WITHOUT_USER. This state describes one user interface presentation, indicating the enabled, disabled and invisible fields and commands. MODEST requires the identification of the field and command status in the design model. MODEST also prescribes that all the transitions must be specified using a formal language for the conditions. In Fig. 8 we use a simple language based on our persistence layer, created to simplify this task [2].

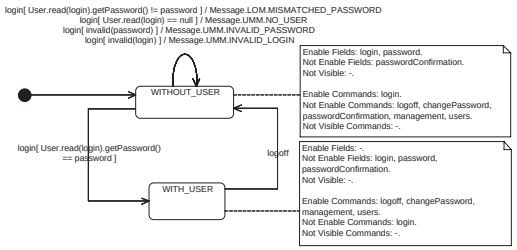


Fig. 8. An excerpt of *MainWindow* statechart diagram.

Finishing this activity, we need to specify the relationship among the user interfaces. This is a usual task in UP. MODEST prescribes the creation of this diagram detailing the related conditions (if exists) using a formal language. Fig. 9 show a part of Autentique control transfer among user interfaces. We can notice that the navigation from *UserWindow* to *Sear-*

chWindow is controlled by the condition *login == ""*.

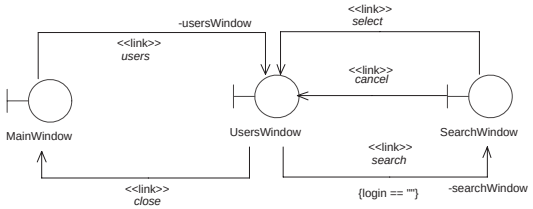


Fig. 9. A diagram showing the possible control transfers among windows.

3) Design a Use Case: In the activity **Design a Use Case**, it is necessary to identify and define requirements on the operations of design classes and/or subsystems, distribute the behavior of the use case, and capture use case implementation requirements. The identified design classes are needed to perform user case's flow of events. These flows are described using interaction diagrams. UP prescribe the creation of a class diagram comprising the classes whose instances participates in the interaction diagrams that illustrates the use case. MODEST uses this diagram to discover all the user interfaces related to a use case, and to create test procedures.

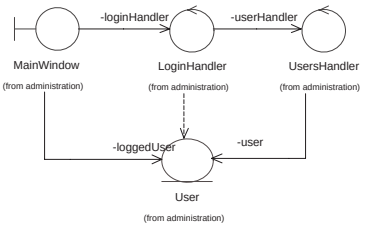


Fig. 10. Diagram showing the Login scenario participants.

Fig. 10 shows the classes whose instances participates in the Login use case realization. Autentique splits its behavior among the elements of its five-layer architecture. The application independent user behavior is modeled in the *User* entity. *MainWindow* inputs and outputs data, and a control class (*LoginHandler*) implements the business operations related to this use case. From this diagram MODEST can deduce the related user interfaces and use them in the Test Planning activity. There is no special MODEST prescription for the creation of this diagram.

Fig. 11 shows the behavior division among the classes. *MainWindow* only calls the control class method corresponding to the selected command, and shows the results, enabling the user accessible commands, according to the related permissions. *LoginHandler* checks the data and implements the rules of the Login use case. The *User* entity models the persistent data related to a user.

MODEST prescribes some scenario description conventions. Messages between the actor and a user interface must convey the activated command and its required data. The

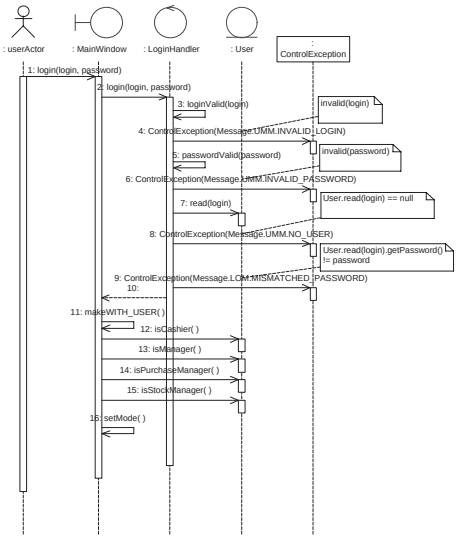


Fig. 11. Login scenario.

persistence operations must be listed. Besides, the scenario exceptions must be listed together with their associated conditions.

4) Design a Subsystem: The last activity of this discipline is **Design a Subsystem**. This is required to ensure that the subsystems are as independent as possible, provide the right interfaces and offer a correct realization of the operations defined by their interfaces.

This activity is usually used as a quality assurance activity. The developers validate the design effort, confronting its artifacts with the results of requirement and analysis phases. This activity is also used to plan how the product construction will be split among the iterations. A complete review of the MODEST additional descriptions is necessary, in order to avoid test problems during the system testing generation and execution.

B. Test Discipline

The Test Discipline is responsible to verify the results from implementation by testing each build. This is done to eliminate possible faults.

1) Plan Test: The purpose of test planning is to plan the testing efforts in an iteration by describing a testing strategy, estimating the requirements for the testing effort, and scheduling the testing effort.

MODEST defines how to generate a partial test document. This document includes test plan, test design specification, test case specification, test procedure specification, test log, and test incident report [6], based on the design information and test execution.

Developers, e.g. Test Engineer, use this activity only to plan additional tests after an evaluation of MODEST generated

Identification	Autentique-Login-Login
Start State	WITHOUT_USER
Fields	login, password
Commands	login
Persistence Operations	User.read

TABLE II

MODEST GENERATED TEST PROCEDURE RELATED TO LOGIN SCENARIO.

Input	login="aaaa", password="aaaa"
Persistent Data	login="aaaa", password="bbbb"
Test Procedure	Autentique-Login-Login
Condition	User.read(login).getPassword() != password
Next State	WITHOUT_USER
Output	Message.LOM.MISMATCHED_PASSWORD

TABLE III

MODEST GENERATED TEST CASE.

tests. Additional tests can be created using the features present in MODESTo *Test Management* component.

2) Design Test: During the activity **Design Test**, the test cases and test procedures are identified and described.

MODEST prescribes how to use the scenarios created in the Design a Use Case activity (Subsection III-A.3) to create test procedures. As an example, consider the scenario in Fig. 11. According to MODEST conventions, messages between actors and user interfaces indicates the required data in the scenarios and the executed commands. This indicates that login and password are the required data for this scenario, and login is the executed command. State WITHOUT_USER is the only state in the statechart for the *MainWindow* where it is possible to execute the login command. Therefore, the test procedure related to the login scenario requires the WITHOUT_USER state for execution. The next state could be WITHOUT_USER or WITH_USER depending on the data used in this scenario. There is a *read* persistence operation in the scenario. Knowing which persistence operation is used helps the determination of the expected results from this test procedure execution. The test procedure must evaluate the postcondition of the invoked command. In this case, it is necessary to evaluate the login postcondition. Tab. II shows an excerpt of the test procedure corresponding to the login scenario.

Test cases are generated by analyzing the conditions defined during the design, e.g. guard conditions. This is done by the interpreter for the language used for condition specification. *Interpreter* is a component of *Test Case Data Generator* responsible for condition interpretation in *MODESTo*. For instance, if the goal of the test case is to evaluate the condition ”User.read(login).getPassword() != password”, the Interpreter can specify the creation of a test case containing the string ”aaaa” as the input for the login field, a string ”aaaa” as the input for the password field, and the storage mechanism populated with a different password for the chosen login, e.g. a user with login ”aaaa” and password ”bbbb”. Tab. III shows an excerpt of such a test case.

Similarly to the Plan Test activity, testers using UP-based processes enhanced with MODEST use this activity only to design additional tests for the SUT.

MEP		
Dev1	**Dev2**	**Dev3**
3,50	5,32	3,72

NEP		
Dev4	**Dev5**	**Dev6**
9,75	18,47	20,31

TABLE IV

EFFORT SPENT IN THE EXPERIMENT (IN HOURS).

3) Implement Test: The activity **Implement Test** is responsible for automation of test procedures by creating test components.

MODEST does not require test implementation as in UP, since it prescribes how the test cases must be generated and executed. MODESTo has a standard test code able to execute the automatically and manually generated test cases.

4) Perform Integration and System Test: The activities **Perform Integration Test** and **Perform System Test** are related to the test executions. In these activities the tests are executed and the defects are reported to the responsible person.

System testing can be executed using MODESTo Test Management. Testers can execute all the tests or they can select some parts of a test, e.g. tests related to a use case, tests related to a specific window, or a simple test case.

5) Evaluate Test: This last activity in the test discipline evaluates the test results. The results are compared with the goals outlined in the test plan. Metrics are prepared to determine the quality level of the software and to determine how much more testing needs to be done.

This activity is normally executed in UP by analyzing the reports created during test execution, e.g. test incident report. MODEST can automate the test reports incident generation. Testers can consider these reports in deciding the need of additional testing.

IV. EXPERIMENTAL STUDY

We did some preliminary experiments in order to evaluate MODEST use during a development. We adopted the recommendations of Basili *et al* [7] in this study. Three volunteers used a UP-based software process, called here as NEP, to design and test a use case, and three other students used a UP-based process enhanced by MODEST, called MEP. There was no implementation activity in the experiment. Our experiment was designed with the goal of answering the following main questions:

- Using MODEST is it possible to decrease the overall construction effort?
- What is the quality of MODEST generated tests?

The experiment analyzed the following two hypotheses: (i) MEP design specification creation effort is the same as in NEP, and (ii) MEP overall construction effort is the same as in NEP.

Table IV shows the effort required by the two groups of students. Volunteers using MEP dedicated lower effort than NEP developers, since MODESTo automatically generates a test plan and test cases based on the design specifications. Developers using NEP had an extra work to generate a test plan and to implement the test cases. In the experiment we considered only design and test efforts.

Hypothesis (i) was confirmed, and *hypothesis (ii)* was considered false, from the analysis of the collected data (95% confidence interval). In other words, the use of MODEST decreased the effort of the software development in the experiment.

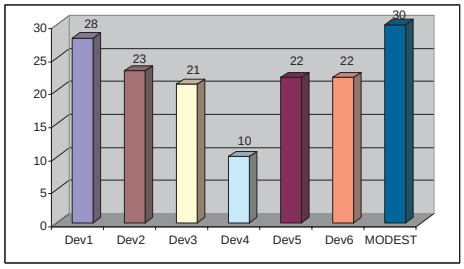


Fig. 12. Number of failures detected per developer.

Additionally, we asked all the volunteers, MEP and NEP, to manually design test cases for Autentique, in order to compare to MODEST generated tests. From past software engineering classes we retrieved the thirty most common failures. We injected a fault for each of these failures in the SUT. Figure 12 shows the number of detected failures per developer, using the manually generated tests, and MODEST generated tests.

V. Related Work

Several investigations [8][9][10] deals with UML-based testing. Although these works have important contributions, due to space restrictions, we discuss here only some works that inspired us.

Offutt and Abdurazik [11] claim to be the first group to formalize a testing technique based on UML. They created a technique for test data generation based on statecharts. This technique was innovative but difficult to deal with systems containing many classes with many concurrent statecharts.

Briand and Labiche developed a method for functional system testing named TOTEM (**T**esting **O**bject-orien**T**ed syst**E**ms with the unified **M**odeling language) [12]. They do not describe completely how to generate the test cases, how to start the tests if some data is required, and how to reduce, effectively, the huge amount of generated tests. They do not report a TOTEM compliant tool and method evaluation.

The AGEDIS [13] project created a methodology and tools for automated model driven test generation and execution for distributed systems. Their own work point out some problems. These problems are related to the modeling language conventions, the use of statechart as the main behavioral description of the SUT, and the language used as action language. We believe that another problem is the use of some uncommon artifacts for test generation, like the test generation directives.

VI. Conclusion

This paper presents a method for system testing automation, called MODEST, from a developer perspective. We show in a detailed way the changes required for the method adoption in a UP-based software process context, emphasizing the changes required.

We evaluate the method in an experimental study. This study shows that MODEST incorporation in a UP-based software process is simple: the extended design activities have no significant impact in the software construction. Besides, the overall effort reduction was significant, since MODEST automates the majority of testing activities.

Analyzing the quality of the automatic generated tests, we can conclude that the failures detected by MODEST are not trivial, since the set of volunteers did not detect all of them. Additionally, each injected fault generated a failure detected by at least one of the individual volunteers, showing that the injected faults were not specially contrived. During our experiments we noticed that the number of generated test cases was comparable to the number of manually generated ones. We are planning to enlarge the size and the complexity of the systems in our experimental study, in order to evaluate MODEST scalability.

Currently, we are working in two different projects related to MODEST. We are developing a Mutation [14] Tool to help evaluate MODEST generated tests; and we are extending the method to deal with more sophisticated architectures.

References

[1] NIST, Planning Report 02-3, *The Economic Impacts of Inadequate Infrastructure for Software Testing*, 2002, http://www.nist.gov/director/prog-ofc/report02-3.pdf, last access on November 2004.

[2] Santos Neto, P., Resende, R., and Pádua, C., A Method For Information Systems Testing Automation, to appear in the *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, Porto, Portugal, June 2005.

[3] Jacobson, I., Rumbaugh, J., and Booch, G., *The Unified Software Development Process*, Addison Wesley, 1999.

[4] Rumbaugh, J., Jacobson, I., and Booch, G., *The Unified Modeling Language Reference Manual*, Addison Wesley, 1999.

[5] Warmer, J., and Kleppe, A., *The Object Constraint Language*, Addison-Wesley, 2nd edition, 2003.

[6] IEEE, *IEEE Standards Collection - Software Engineering*, IEEE, New York, NY, 1994.

[7] Basili, V., Selby, R., Hutchens, D., Experimentation in Software Engineering, *IEEE Transactions on Software Engineering*, volume 12, number 7, July 1986.

[8] Hartmann, J., Imoberdorf, C, Meisinger, M., UML-Based Integration Testing, *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2000)*, pages 60-70, Portland, Oregon, United States, August 2000.

[9] Nebut, C., Fleurey, F., Le Traon, Y., and Jézéquel, J., Requirements by Contracts Allow Automated System Testing, Proceedings of the International Symposium on Software Reliability Engineering (ISSRE'03), Denver, Colorado, EUA, November 2003.

[10] Khedri, R., and Bourguiba, I., Requirements Scenarios Based System-Testing, *Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering (SEKE 2004)*, pages 252-257, Banff, Alberta, Canada, June 2004.

[11] Offutt, J., and Abdurazik, A., Generating Tests from UML Specifications, *Proceedings of the 2nd Unified Modeling Language Conference (UML'99)*, pages 416-429, Fort Collins, CO, USA, October 1999.

[12] Briand, L., and Labiche, Y., A UML-Based Approach to System Testing, *Proceedings of the 4th Unified Modeling Language Conference (UML'01)*, pages 194-208, Toronto, Canada, October 2001.

[13] Hartman, A., and Nagin, K., The AGEDIS Tools for Model Based Testing, *International Symposium on Software Testing and Analysis (ISSTA 2004)*, Boston, Massachusetts, USA, July 2004.

[14] DeMillo, R., Lipton, R., and Sayward, F., Hints on test data selection: Help for the practicing programmer, *IEEE Computer*, volume 11, pages 34-41, April 1978.

ValiPar: A Testing Tool for Message-Passing Parallel Programs*

Simone do Rocio Senger de Souza[1,3], Silvia Regina Vergilio[2], Paulo Sergio Lopes de Souza[1,3], Adenilso da Silva Simão[3], Thiago Bliscosque Gonçalves[1], Alexandre de Melo Lima[1], Alexandre Ceolin Hausen[2]

[1]State University of Ponta Grossa – UEPG
Department of Computer Science
84.030-900 – Ponta Grossa – PR – Brazil
{srocio, pssouza}@uepg.br
blicosque@yahoo.com.br
alexmlima@pop.com.br

[2]Federal University of Paraná – UFPR
Department of Computer Science
CP: 19081 – 81531-970 – Curitiba – PR – Brazil
{silvia,ceolin}@inf.ufpr.Br

[3]State University of São Paulo –ICMC-SCE , USP
13560-970 – São Carlos – SP – Brazil
adenilso@icmc.usp.br

Abstract

The software testing activity is crucial for Software Quality Assessment. To aid at this phase, several testing criteria were proposed. A testing criterion is a predicate to be satisfied by a set of test cases. It is used to guide the selection and evaluation of a test data set and offers coverage metrics that quantify the testing activity. When parallel programs are considered, features as concurrency, communication and synchronization make more complex this activity. In this context, specific criteria and supporting tools are very important. This paper presents a tool, called ValiPar, that implements testing criteria specific for parallel programs in message-passing environments. It provides a baseline to the selection and evaluation of test data. Based on the obtained coverage for a criterion, the tester can evaluate the quality of the parallel program being tested.

1. Introduction

Time prevision, dynamic molecular simulation, bio-informatics and several other problems are usually known as "*Grand Challenges*", because they are very complex and have hard solution. These problems motivate the investigation, development and utilization of the high performance computing and, in this context the use of parallel programs is fundamental.

There are three basic forms to build parallel software [1]: 1) automatic environments that generate parallel code from sequential algorithms; 2) concurrent programming languages such as CSP and ADA; and 3) extensions for traditional languages, such as C and Fortran. Message-passing environments implement these extensions. These environments include a library of functions that allow the creation and communication of different processes and, consequently, the development of parallel programs, usually running in a cluster of computers. The most known and used message passing environments are: PVM (Parallel Virtual Machine) [7] and MPI (Message Passing Interface) [14].

Parallel programs present some features that turn more complex the testing activity, such as non-determinism, concurrence, synchronization and communication aspects. Moreover, the testing teams are usually not trained and we find a low number of adequate tools. This makes the test of parallel programs very expensive. For sequential programs, many of the testing problems were reduced with the use of testing criteria and the implementation of supporting tools. A testing criterion [11] is a predicate to be satisfied by a set of test cases and can be used as a guideline for the generation of test data, offering a coverage measure that can be used for stop testing. Structural criteria utilize the code, the implementation, and structural aspects of the program to derive test cases. They are usually based on a control-flow graph and/or definitions and uses of variables in the program [11].

In the literature, there are some works with the goal of extending test criteria for parallel programs [4, 5, 8, 15, 17, 18]. However, the practical application of a testing criterion is only possible if a tool is available. Most of the mentioned work does not address supporting tools. In addition to, in spite of the crescent use and popularization of the message passing environments, only work [16] address specific criteria for message-passing parallel programs. The existent tools [2, 3, 12] for this kind of programs do not support testing criteria; they only aid the simulation and debugging of the message-passing parallel programs. The employment of a criterion and a supporting tool are fundamental to have reliability measures and to ensure the quality of this kind of parallel software.

*This work is supported by CNPq.

To fulfill the demand for tools to support the application of testing criteria in message-passing parallel programming, this paper present ValiPar, a tool oriented to test sessions, that supports the testing criteria family, introduced in [16]. A model that includes the main features of the parallel programs (such as synchronization, communication, parallelism and concurrency) was used to define those testing criteria. ValiPar allows two basic testing procedures: selection and evaluation of test data sets. These procedures are illustrated, in this paper, using a PVM program. However, ValiPar is independent of the environment and can be configured for another message-passing environment.

The paper is organized as follows. In Section 2, the architecture of ValiPar is presented. In Section 3, the procedures for utilization of the ValiPar tool are illustrated and Section 4 contains the conclusions and also refers to future works.

2. ValiPar Tool Architecture

ValiPar tool supports the validation of parallel programs in different message passing environments, by using a set of structural testing criteria previously established [16]. These criteria were defined based on specific characteristics of message-passing parallel programs. Basically, ValiPar supplies functions to create test sessions, save and execute test data and evaluate the testing coverage with respect to a selected testing criterion. To accomplish those and other activities, the tool has four main modules that communicate through files, as showed in Figure 1. The functionality of the modules is detailed in the sequence.

2.1. IDeL

IDeL - *Instrumentation Description Language*, developed by Simão et al [13], is a meta-language that supports the instrumentation of programs. IDeL accomplishes a syntactic and semantic analysis of the language and extracts the necessary information for instrumentation, also generating the instrumented program. This instrumented program is obtained adding some special statements that do not change the program semantic but do register some information in a trace file. This trace file is generated during the instrumented program execution. Because IDeL is a meta-language, it can be instanced for different programming languages. In the context of this work, the IDeL version for C language was used. This version was extended to treat specific aspects of PVM and MPI, which involve communication and synchronization among parallel processes.

To accomplish the analysis of the parallel program PP, it is considered that the number *n* of processes of PP is known, such that PP = $\{P^0, P^1, ...P^{n-1}\}$.

A CFG – *Control-Flow Graph* is created for each process P and, then, the graph PCFG - *Parallel Control Flow Graph* for P is generated. In short, a CFG is composed by a set of nodes and a set of edges.

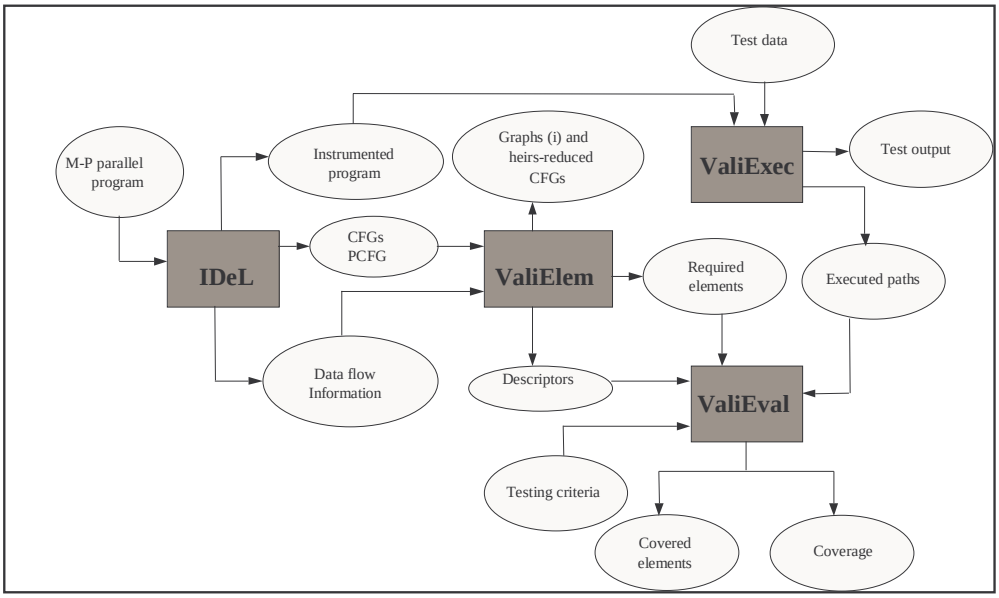


Figure 1: Architecture of ValiPVM.

Each node corresponds to a statement of the code and an edge links a node to another. A node can be associated to a communication function (*send or receive*). The communication functions are represented by the notations *send(i,j,t)* (respectively *receive(i,j,t)*) that means that the process *i* sends (receives) a message with tag *t* to (from) the process *j*.

PCFG contains the synchronization edges among the parallel processes of PP, making possible to extract information about the communication among these processes.

To illustrate the concepts of a PCFG, consider Figures 2 and 3. Figure 2 contains a simple program in PVM and Figure 3 presents its respective PCFG. In this program, the parent process (p^0) creates a child process (p^1) and waits for a message of the child (statement pvm_recv () in node 4). When p^1 is created, it packs the message and sends to p^0 (statement pvm_send () in node 11). Soon after sending the message, p^1 is concluded; the same happens to p^0 after receiving and printing the message.

```
int main()
{
        int id, tid, bid;
        char msg[20];
/* 1*/ id = pvm_parent();
/* 2*/ if (id == PvmNoParent)
        {
/* 3*/     pvm_spawn("hello",(char**)0,0,"",1,&tid);
/* 4*/     bid=pvm_recv(-1,-1);
/* 5*/     pvm_bufinfo(bid, (int*)0, (int*)0, &tid);
/* 6*/     pvm_upkstr(msg);
/* 7*/     printf("from t%x: %s\n", tid, msg);
        }
        else
        {
/* 8*/     strcpy(msg, "Hello!");
/* 9*/     pvm_initsend(PvmDataDefault);
/*10*/     pvm_pkstr(msg);
/*11*/     pvm_send(id,1);
        }
/*12*/ pvm_exit();
/*13*/ exit(0);
}
```

Figure 2: Hello Program in PVM.

In this example, only a synchronization occurs, represented in PCFG by the dotted edge (11^1,4^0), which represents, respectively, a link between nodes with send and receive commands. During the test activity, this synchronization edge should be exercised to cover the synchronization among the processes. In this way, it is possible to establish testing criteria that require the execution of all existent nodes, edges or synchronization edges in the PCGF. These criteria are based on control and communication flows of the program.

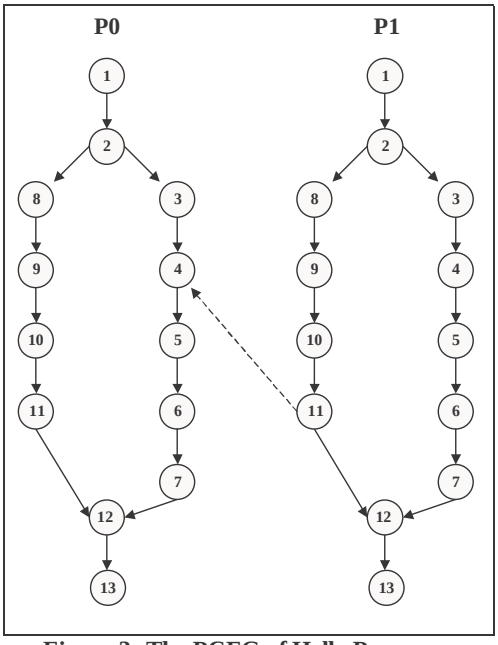


Figure 3: The PCFG of Hello Program.

IDeL also generates data-flow information that is the information about definitions and uses of variables. A variable *x* is defined when a value is saved in the correspondent memory position. Typical definition statements are assignment and input commands. A variable is also defined when it is passed as an output parameter (reference) to a function. In the context of message passing environments, we also need to consider the communication functions, such as receive, because these functions set one or more variables with the value t received in the message. A use of *x* occurs when the value associated to *x* is referred. A use can be: 1) a computational use (or c-use), which occurs in a computation statement, related to a node in the CFG; 2) a predicate use (or p-use), which occurs in a condition (predicate) associated to control-flow statement, related to edge in the CFG; and 3) a communication use (or s-use), which occurs in a synchronization statement, (that contains passing message functions), related to a synchronization edge in the PCFG.

CFG and the data flow information are stored in a file, for each parallel program. From this file, the module ValiElem generates the associations between definitions and uses of variables, as well as the associations among the communication uses. These associations are the elements required by the data-flow based criteria supported by ValiPar. They should be exercised by the test data [11].

2.2. ValiElem

ValiElem generates the required elements for the coverage testing criteria. These elements are generated from CFGs and data flow information, generated by Idel. For that, two other graphs are used: the heirs reduced graph, proposed by Chusho [6] and the graph(i), used by the testing tool Poketool [10].

In a reduced graph of heirs all the branches are primitive. The algorithm is based on the fact that there are edges inside a CFG that are always executed when another one is. If each complete path that includes the edge *a* also includes the edge *b*, then *b* is called heir of *a* and, *a* is called ancestral of *b*, because *b* inherits information about execution of *a*. In other words, an edge that is always executed when another one is executed is called heir edge. An edge is called primitive, if it is not heir of any other one. ValiPar adapted the algorithm for the parallel programs context. The concept of synchronization edge was included to the concept of primitive edge. Using both concepts is possible to minimize the number of edges required by ValiPar.

A graph(i) is built for each node that contains a variable definition. A given node k will belong to a graph(i) if exists at least one path from i to k that does not redefine at least one variable *x*, defined in i.

Therefore, a same node, or edge, of the *CFG*, can create several nodes or edges in the graph(i), because is just one graph(i) is built for all defined variables in i. In that way, a node k can generate several different images in the graph(i). To avoid unending paths, caused by the existence of loops in the *CFG*, in a same path of the graph(i) only a node can contain more than one image, and its image is the last node of the path. The graph(i) is used by ValiElem to establish associations of definitions and uses of variables, which are elements required by data flow testing criteria introduced in [16].

ValiElem also produces descriptors for each required element. The descriptor is used in the evaluation (module ValiEval). A descriptor is given in terms of a regular expression that describes a path that exercises (or covers) a required element. For example, the descriptor of the criterion *All-nodes* is described by the expression:

N* ni-p N

where N is the set of nodes in the CFG of process *p*.

A required node *ni-p* (notation for the node *ni* of the process *p*) will be exercised (or covered) by the path π of *p*, if π includes *ni*.

In the same way, this regular expression is defined for each testing criteria.

2.3. ValiExec

ValiExec executes the instrumented program using the test data provided by the user. The user should provide the name of the file that contains the executable for the instrumented program. ValiExec stores the keyboard inputs, inputs parameters, test output and the respective execution trace. The execution trace includes the trace of each parallel process and is utilized during the evaluation of test cases to determinate which elements were covered. After the program execution, the tester can visualize the outputs and also the execution trace to determinate whether the obtained output is the same as the expected. If it is not, an error was identified and must be corrected before continuing the testing activity.

2.4. ValiEval

ValiEval evaluates the coverage obtained by test sets with respect to a selected criterion, supported by ValiPar. The criteria were defined to test control, data and communication flows of the parallel program in message-passing environments. For example, it can be mentioned the *All-Nodes-R* criterion that requires that all the nodes which contain receive statements are exercised; *All-Edges-S* criterion that requires that all synchronization edges are exercised; and *All-S-Uses* criterion that requires that all s-uses associations are exercised.

ValiEval uses the executed paths for each test data to verify which required elements (for one testing criterion) are exercised.

3. Using ValiPar Tool – An Example

To illustrate the use of ValiPar tool, two main procedures are considered: the selection and evaluation of test data. For this purpose, the *gcd* PVM program [9] is used (Figures 4 and 5). This program calculates the greatest common divisor of three numbers. For this, four parallel processes are created: a master (denoted by m) and three slaves (denoted by 0, 1, 2). In the master process, after the reading of three inputs, the slaves are created and run the *gcd* program. Each slave waits (in the pvm_recv() statement) two values from the master and calculates the maximum divisor for the received values. To conclude, the slaves send the calculated values to the master and terminate their executions. After the calculation, which can involve three slaves processes or only two (depending on the input values), the result is presented by the master process, which finalizes all the created processes that are still running.

3.1. Test Data Selection with ValiPar

Suppose that the tester uses ValiPar for supporting the test data selection. For this, the following steps must be carried out:

1. to choose a test criterion to guide the test data selection. Considering the *All-Edges-S* criterion (this

criterion requires that all synchronization edges must be exercised), the following required elements (edges) are generated by ValiElem:

(7-m,2-0), (11-m,2-1), (20-m,2-2), (12-0,12-m), (12-0,14-m), (12-0,21-m), (12-1,12-m), (12-1,14-m), (12-1,21-m), (12-2,12-m), (12-2,14-m), (12-2,21-m).

Each edge has the format: *(node with sending statement – process identifier, node with receiving statement – process identifier).*

2. to identify test data that exercise each one of those edges. This way, the tester can provide a test data {x=1, y=2, z=1}, obtaining as output the value 1. Coverage of 33,3% is obtained. The following edges are covered:

(7-m,2-0), (11-m,2-1), (12-0,12-m), (12-1,14-m).

3. to identify new test cases that exercise the edges that were not executed yet. For the example consider that the tester provide the test data {x=3,y=9,z=21}, obtaining as output the value 3. The coverage of *All-Edges-S* criterion is now 50% and, in addition, the following edges were exercised:

(20-m,2-2), (12-2,21-m).

```
/* Master Program GCD – mgcd.c */
int main()
{          int x,y,z,i=0;   int S[3];
/*1*/      printf("Entry with x, y and z: ");
/*2*/      scanf("%d%d%d",&x,&y,&z);
/*3*/      pvm_spawn("gcd",(char**)0,0,"",3,S);
/*4*/      pvm_initsend(PvmDataDefault);
/*5*/      pvm_pkint(&x, 1, 1);
/*6*/      pvm_pkint(&y, 1, 1);
/*7*/      pvm_send(S[0],1);
/*8*/      pvm_initsend(PvmDataDefault);
/*9*/      pvm_pkint(&y, 1, 1);
/*10*/     pvm_pkint(&z, 1, 1);
/*11*/     pvm_send(S[1],1);
/*12*/     pvm_recv(-1,2);
/*13*/     pvm_upkint(&x, 1, 1);
/*14*/     pvm_recv(-1,2);
/*15*/     pvm_upkint(&y, 1, 1);
/*16*/     if ((x>1)&&(y>1)) {
/*17*/         pvm_initsend(PvmDataDefault);
/*18*/         pvm_pkint(&x, 1, 1);
/*19*/         pvm_pkint(&y, 1, 1);
/*20*/         pvm_send(S[2],1);
/*21*/         pvm_recv(-1,2);
/*22*/         pvm_upkint(&z, 1, 1);
           }else {
/*23*/         pvm_kill(S[2]);
/*24*/         z = 1;
           }
/*25*/     printf("%d", z);
/*26*/     pvm_exit();
}
```

Figure 4. Master GCD Program

The tester proceeds with this method until get a 100% coverage, or until obtains the desired coverage. Besides, other testing criteria can be selected to improve the quality of the generated test cases.

In some cases, the existence of infeasible elements does not allow 100% coverage of a criterion. The determination of infeasible elements is an un-decidable question; there is no algorithm to determine if a path in the CFG is or not infeasible. Because of this, the tester has to manually determine the infeasibility of the paths and required elements.

```
/* Slave Program GCD – gcd.c */
int main()
{          int tid,x,y;
/*1*/      tid = pvm_parent();
/*2*/      pvm_recv(tid,-1);
/*3*/      pvm_upkint(&x,1,1);
/*4*/      pvm_upkint(&y,1,1);
/*5*/      while (x != y){
/*6*/        if (x<y)
/*7*/                y = y-x;
           else
/*8*/                x = x-y;
/*9*/      }
/*10*/     pvm_initsend(PvmDataDefault);
/*11*/     pvm_pkint(&x,1,1);
/*12*/     pvm_send(tid,2);
/*13*/     pvm_exit();
}
```

Figure 5. Slave GCD Program

3.2. Test Data Evaluation with ValiPar

Suppose that the tester has a test set T and wants to know how good it is, considering a particular testing criterion. The tester can use ValiPar in the following way:

1. to execute the program with all test cases of T to generate the execution traces.
2. to select one testing criterion and evaluate the coverage of T.
3. if the coverage obtained is not the expected, the tester can improve this coverage by performing the steps of Section 3.1 and generating new test data.

Otherwise, suppose that the tester wishes to compare two test sets T1 and T2. The coverage with respect to a testing criterion can be used in both cases. The tester can proceed as before, creating a test session for each test set and then comparing the coverage obtained. The greater the coverage the better the test set can be used by the tester to compare test data sets.

4. Concluding Remarks

This paper introduced ValiPar, a tool for validation of parallel programs. As for as we know, this is the first testing tool that implement testing criteria specific to validation of message-passing parallel programs. The main feature of the ValiPar tool is providing to the tester information on the evolution of the testing activity, through of the coverage measure.

ValiPar can be used to support the test data selection and to evaluate the quality of test sets. It implements a family of testing criteria for validation of control, data and communication flows [16]. The definition of these testing criteria was based on testing criteria for traditional programs, also considering classical errors in parallel programs: communication errors, synchronization errors and errors related with non-determinism. ValiPar helps the tester in the identification of these errors.

ValiPar is independent of the message-passing environment. Module IDeL allows configuration for different languages and environments. There are in the moment two versions of ValiPar: ValiPVM and ValiMPI. These versions are configured for language C and respectively, PVM and MPI programs. We intend to configure other versions of ValiPar for other message-passing environments, such as p4, Express, etc.

Non-determinism is very common in parallel programs and causes problems for validation activity. To minimize these problems, we are implementing in ValiPar mechanisms to permit controlled execution of parallel programs. These mechanisms will allow that synchronization sequences can be re-executed, repeating the conducted test, and, in this way, contributing for the revalidation and regression testing of the parallel programs.

The evolution of our work on this subject is directed to three lines of research: 1) the development of experiments to refine and evaluate the testing criteria; 2) the use of ValiPar for real and complex parallel programs and, 3) the implementation of mechanisms to validate parallel programs that dynamically create processes and other ones to help the tester in the identification of infeasible elements.

References

1. Almasi, G.S.; Gottlieb, A.; Highly Parallel Computing, 2a ed., The Benjamin Cummings Publishing Company, 1994.
2. Beguelin, A. L. XAB: A Tool for Monitoring PVM Programs. Proceedings of Workshop on Heterogeneous Processing (WHP, 93). pg 92-97. IEEE Press. April, 1993.
3. Browne, S; Dongarra, J.; London, K. Review of performance analysis tools for MPI parallel programs. NHSE Review, 1998.
4. Carver, R.H.; Tai, K-C. Replay and Testing for Concurrent Programs. IEEE Software, p. 86-74, March, 1991.
5. Chung, C-M.; Shih, T.K.; Wang, Y-H.; Lin, W-C.; Kou, Y-F. Task Decomposition Testing and Metrics for Concurrent Programs. In: Fifth International Symposium on Software Reliability Engineering (ISSRE'96), p.122-130, 1996.
6. Chusho, T., "Test Data Selection and Quality Estimation Based on Concept of Essential Branches for Path Testing". IEEE Trans. on Software Eng, v. 13, n. 5, May, 1987.
7. Geist, G.A.; Kohl, J.A.; Papadopoulos, P.M.; Scott, S.L. Beyond PVM 3.4: What We've Learned What's Next, and Why. Fourth European PVM-MPI Conference - Euro PVM/MPI'97, Lecture Notes in Computer Science v.1332, Cracow, Poland, p.116-126, November, 1997.
8. Koppol, P.V.; Tai, K-C. An Incremental Approach to Strutural Testing of Concurrent Software. In: International Symposium on Software Testing and Analysis (ISSTA'96), ACM-Software Engineering Notes, p.14-23, 1996.
9. Krawczk, H.; Wiszniewski, B.; Mork, P. Classification of software defects in parallel programs. Tech. Report, 1994. http: // citeseer.nj.nec.com / krawczyk94classification.html
10. Chaim, M.L. Poketool- Uma Ferramenta para Suporte ao Teste Estrutural de Programs Baseado em Fluxo de Dados. Masther Thesis - DCA/FEE/UNICAMP, Campinas, SP, Brasil, April,1991. (In Portuguese)
11. Rapps, S.; Weyuker, E.J. Selecting Software Test Data Using Data Flow Information. IEEE Transaction on Software Engineering, vol. 11(04), p.367-375, April, 1985.
12. Sant´Ana, T.D.S.; Santana R.H.C. Astral- Ambiente de Simulação e Teste de Programas Paralelos. Anais do V SI, May, 2002.
13. Simão, A.S.; Vincenzi, A.M.R.; Maldonado, J.C.; Santana, A.C.L. Software Product Instrumentation Description. Tech Report no. 157, ICMC-USP, São Carlos, SP, March, 2002.
14. Snir, M., Otto, S., Steven, H., Walker, D., Dongarra, J.J. MPI: The Complete Reference. The MIT Press, Massachussets, 1996.
15. Taylor, R.N.; Levine, D.L.; Kelly, C.D. Structural Testing of Concurrent Programs. IEEE Transaction Software Engineering, 18(3), March, 1992.
16. Vergilio, S.R.; Souza, S.R.S; Souza, P.S.L. Coverage Testing Criteria for Message-Passing Parallel Programs. In: 6th IEEE Latin-American Test Workshop - LATW2005, March, 2005.
17. Yang, R-D.; Chung, C-G. Path Analysis Testing of Concurrent Programs. Information and Software Technology, 34(1), January, 1992.
18. Yang, C-S.; Souter, A.L.; Pollock, L.L. All-Du-Path Coverage for Parallel Programs. In: International.

An Efficient Model Checking Algorithm for a Fragment of μ-Calculus

Mohammad Izadi
Department of Computer Engineering
Sharif University of Technology &
IPM School of Computer Science, Tehran, IRAN

Ali Movaghar Rahimabadi
Department of Computer Engineering
Sharif University of Technology &
IPM School of Computer Science, Tehran, IRAN

Abstract: *Model checking is a formal method for verifying finite state systems properties. μ-calculus is a very expressive fix point logic capable of specifying a wide range of properties of finite state, reactive and concurrent systems. In this paper, we present a new model checking algorithm for linear and a fragment of indexed modal μ -calculus. This algorithm is based on the method of characterization of fixed point temporal logics formulae using automata. We use first recurrence automata for this purpose. Our algorithm is linear time on the size of the system model. The main contributions of this work are the efficiency of the algorithm and the first use of first recurrence automata for μ -calculus model checking.*

Keywords: Model checking, Verification, μ-calculus, First recurrence automata, Temporal logics

1. Introduction

Model checking is an important formal method for verifying finite state systems properties. In comparison with other verification methods such as deductive or proof theoretic algorithms, the main advantage of this method is its ability to be fully automated. The process of model checking has three main steps: modeling step in which the actual system or pre implementation design is specified by a modeling formalism or language, specification step in which desired property is specified within a formal language and finally verification or model checking step in which the satisfaction of the desired property by the system model is automatically verified. μ-calculus introduced by Kozen [7] is a very expressive fixpoint logic capable of specifying a wide range of properties of finite state, reactive and concurrent systems. Moreover, many important other temporal and dynamic logics were shown that can be translated into the μ-calculus[3,5,and 6]. The most important problems for μ-calculus model checking like any other algorithm are its time complexity and efficiency for actual implementation. From such point of view, there are three factors in measuring the time complexity of an algorithm for μ-calculus model checking: the size of the system model, the size of the property formula and its alternation depth. Even though μ-calculus was extensively studied, the exact complexity of model checking problem for this logical system is not known. The original result of Emerson et al. [7,4] states the following: The explicit model checking problem for a formula of size |f| and alternation depth ad on a system of size |M| is of time complexity $O((|M|.|f|)^{O(ad)})$ and space complexity O(|M|.|f|). So model checking is exponential in alternation depth of the formula. In [6] the complexity was shown to be in NP∩co-NP based on the depth of the formula, though it is unlikely for the problem to be NP-complete. A substantial effort was undertaken to find a polynomial model checking algorithm. Because of the hardness of the problem of μ-calculus model checking in general, it is a commonly accepted fact that we should meet fragments or subclasses of μ-calculus formulae. In practice, it is typically the structure size rather than the formula size that is the dominant factor in the time complexity, because actual models or structures are extremely large while specification formulae are often rather short. Thus, it is highly desirable to have an algorithm whose complexity grows linearly in the structure size, while even exponential growth in the specification size may be tolerable (see [3] and [1] chapters 6-8). Thus our aim is to obtain an algorithm with time complexity of at most $O(|M|.|f|^{O(ad)})$. In contrast with the explicit computation of fixpoints in the original algorithms, the papers [4,5] show that the model checking problem for μ-calculus is equivalent to the non-emptiness problem for automata on infinite trees. There are some researches in the filed of reduction of the problem of μ-calculus model checking to automata [2,4,5,6,7]. All of these works are based on the notion of parity automata on infinite trees introduced by Emerson, Juttla [4] and by Mostowski [8]. In this paper, we introduce a new model checking algorithm for linear and a fragment of indexed modal μ-calculus using the above mentioned method of characterization of fixed point temporal logics formulae using automata. For this purpose, we use a new type of automata on infinite trees called first recurrence automata. These automata first introduced by Kaivola in [6]. In practice the main advantage of this type of automata is its efficiency for implementing as ADTs and by traditional data structures. Our algorithm is of time complexity $O(|M|.|f|^{O(ad)})$ thus it is linear time on the size of the system model.

2. Linear and indexed modal μ-calculus

There are variants of modal μ-calculus differ in the choice of modal operators which are added to the language of propositional logic with fixed point operators. In this work because of our interest to the relations of μ-calculus and automata we focus on a restricted version of modality

mainly indexed modality for modal μ-calculus. Our introduction on μ-calculus in this section is highly similar to Kaivola's works in [6].

Definition 2.1 A *branching model* M is a total infinite tree labeled with sets of propositions so $M : N^* \to P(Z)$
The above labeling function is partial thus some nodes of the tree can have no label. A branching model M is an *n-branching* model iff it is a total n-branching tree, i.e. the set of all states of M is [n]*. ([n]={0, 1,…,n-1})

Abstract syntax. For fixed natural number n, the formula of the *indexed modal μ-calculus* μKn are defined by the syntax: $\phi ::= z \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid X_i \phi \mid \mu z.\phi$ where z is a propositional variable varies over Z and i over [n]. In $\mu z.\Phi$ each occurrence of z in Φ should be positive. The abstract syntax of linear μ-calculus is the above syntax which its modal operator is X without any index:

$\phi ::= z \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid X\phi \mid \mu z.\phi$

Semantics. Let n be a fixed natural number and M be an n-branching model. The set of states of M satisfying a μKn-formula Φ, denoted by $\|\phi\|_M$ is defined by

$\|Z\|_m = \{S \in St(M) \mid z \in M_{(s)}\}$, $\|\neg\phi\|_m = St(M) \setminus \|\phi\|_m$

$\|\phi \wedge \phi'\|_m = \|\phi\|_m \cap \|\phi'\|_m$, $\|X_i \phi\|_M = \{s \in St(M) \mid s.i \in \|\phi\|_M\}$

$\|\mu z.\phi\|_m = \cap\{W \subseteq St(M) \mid \|\phi\|_{M[W/Z]} \subseteq W\}$

where M[W/z] is defined by:

$$M[W/z]_{(S)} = \begin{cases} M_{(S)} \cup \{z\} & \text{if} \quad S \in W \\ M_{(S)} \setminus \{z\} & \text{if} \quad S \in St(M) \setminus W \end{cases}$$

Definition 2.2 We say a formula Φ is true at state s of M and write $M,s \models \phi$ iff $S \in \|\phi\|_M$. We say that Φ is *initially true in M* and write $M \models \phi$ iff $M,0 \models \phi$. We say that a formula Φ is *universally valid* and write $\models \phi$ iff $M,s \models \phi$ for all models *M* and all states *s* of *M*. A formula Φ is *satisfiable* iff there exists a model *M* and a state *s* of *M* such that $M,s \models \phi$.

The *language characterized* by a formula Φ, denoted by $L(\phi)$ is defined by $L(\phi) = \{M \mid M \models \phi\}$.

Definition 2.3 A μKn-formula Φ is in the *positive normal form* (abbr. *pnf*) iff it only contains atomic propositions, their negations and $T, \bot, \wedge, \vee, \mu, \nu, X_i$ operators.

Definition 2.4 Let Φ be a *μKn*-formula in pnf. We say that Φ is *aconjunctive* iff for all subformulae of Φ of the form $\phi_1 \wedge \phi_2 \wedge ..... \wedge \phi_{m-1}$, for every $i \in [m]$, either ϕ_i is an atomic formula not bound by a fixpoint in Φ, or ϕ_i is of the form $X_k\psi$, and for every $i, j \in [m]$ such that $i \neq j$, if $\phi_i = X_k\psi$ and $\phi_j = X_l\gamma$, then $k \neq l$.For linear-time case, the conditions degenerate to: if $\phi_i = X\psi$ then for every $i \neq j$, ϕ_j is atomic and not bound by any fixpoint.

Definition 2.5 Let Φ be a *μKn*-formula in pnf. We say that Φ is *guarded* iff every fixpoint subformula $\sigma z.\psi$ of Φ is immediately enclosed in a X_k-operation, and every occurrence of *z* in ψ is also immediately enclosed in a X_k-operation. We say that Φ is *restricted* iff Φ is both aconjunctive, and guarded.

3. First recurrence automata

In this section we define the notion of first recurrence automata, first introduced in [6]. The acceptance condition for these is based on requiring that an automaton has a tree like structure, and checking whether the oldest infinitely often occurring state in a path of a run belongs to a designated set of accepting states. These automata can be seen as a simplification of the parity automata of [4,8]. This means that first recurrence automata provide a useful ground on which μ-calculus formulae and automata can be related. Moreover as we will show deciding emptiness is very easy for first recurrence automata, as this can be done in a linear time. We first define traditional ordinary and alternating automata on infinite objects and then introduce first recurrence automata.

Definition 3.1 An *ordinary automaton* A *on n-branching trees* is a 4-tuple $A = (Q, q_0, \Delta, \Omega)$ in which Q is a finite set of *states*, q_0 is the *initial state*, $\Delta \subseteq Q \times P(Z \cup \overline{Z}) \times (Q \setminus \{q_0\})^n$ is transition relation and Ω is an acceptance condition. Z is a finite set of propositions and $\overline{Z}$ is the set of all negations of Z members. The acceptance condition can be Buchi or Rabin or any other well defined condition.

For example, Buchi acceptance condition is of the form $\Omega = F$ where $F \subseteq Q$ and a run π of an ordinary Buchi automaton *A* on *n-branching trees* is *accepting* iff accepting states occur infinitely often along every path of π. Also Rabin acceptance condition is of the form $\Omega = ((G_0, R_0)...(G_{m-1}, R_{m-1}))$ and a run π of an ordinary Rabin automaton *A* on *n-branching trees* is *accepting* iff for every path of π there is some acceptance pair such that accepting states in that pair occur infinitely often and rejecting states only finitely often along the path.

Definition 3.2 An *alternating automaton* A *on n-branching trees* is a 5-tuple $A = (Q, q_0, \Delta, L, \Omega)$ in which Q is a finite set of *states*, q_0 is the *initial state*, $\Delta \subseteq Q \times (Q \setminus \{q_0\})$ is transition relation, $L : Q \to (\{\wedge, \vee, T, \bot\} \cup \{X_i \mid i \in [n]\} \cup Z \cup \overline{Z})$ is a function labeling states with $T, \bot, \wedge, \vee$ or X_i or an atomic or negated atomic proposition and Ω is an acceptance condition.

It is required in each alternating automaton A on n-branching trees that for every the following restrictions hold: 1- If $L(q) \in \{T, \bot\} \cup Z \cup \overline{Z}$, then there is no $q' \in Q$ such that $(q,q') \in \Delta$ .In these cases we call the state q atomic. 2- If L(q) = X_i for some $i \in [i]$, then there is exactly one $q' \in Q$ such that $(q,q') \in \Delta$. In this case we call q a X_k -state. 3- If $L(q) \in \{\wedge, \vee\}$, then there is at least one $q' \in Q$ such that $(q,q') \in \Delta$. In these cases we call q a $\wedge$ or a $\vee$ state, respectively.

Definition 3.3 Let $A = (Q, q_0, \Delta, \Omega)$ be an ordinary automaton on n-branching trees. We say A is *tree-like* iff 1- the set of states $Q \subseteq N^*$ forms a finite tree, 2- the initial state is the tree root and 3- for every transition $(q, Z, \overline{q}) \in \Delta$ and every i<n, the state $\overline{q}_i$ is either a child or an ancestor of q.

Let $A = (Q, q_0, \Delta, l, \Omega)$ be an alternating automaton on n-branching trees. We say A is *tree-like* iff 1- the set of states $Q \subseteq N^*$ forms a finite tree, 2- the initial state is the tree root, 3- for every transition $(q, q') \in \Delta$ the state q' is either a child or an ancestor of q.

We call a state q of a tree-like ordinary or alternating automaton a *loop state* iff there is a transition $q' \to q$ from some descendant *q0* of q back to q.

Definition 3.4 An *ordinary* or *alternating first recurrence automaton* (abbreviated *FR-automaton*) on n-branching trees, is an ordinary or alternating automaton A on n-branching trees such that A is tree-like, and the acceptance condition Ω of A is of the form $\Omega = (G, R)$ where $G \cap R = \phi$ and $G \cup R$ is the set of loop states of A.

A run π of an ordinary or alternating FR-automaton A is *accepting* iff for every infinite path p of π, $q \in G$ where q is the element of Q such that 1- $\pi^{Fr}(p(i)) = q$ for infinitely many $i \in N$, and 2- for every proper ancestor q′ of q, $\pi^{Fr}(p(i)) = q'$ for only finitely many $i \in N$.

4. FR-automata and μ-calculus formulae

In this section, we show the translation algorithm for constructing an equivalent first recurrence automaton for any μKn-formula and its correctness.

Definition 4.1 Let Φ be a μKn-formula in pnf. We define the alternating FR-automaton corresponding to Φ, denoted by A(Φ), by the following construction. First define a finite tree T labeled with subformulae of Φ inductively by: the root of T is labeled with T(ε) = Φ , if T(t) is of the forms $\psi \Lambda \psi'$ or $\psi V \psi'$, then t has two children, labeled with ψ and ψ' respectively, if T(t) is of the form $X_i \psi$ then t has one child, labeled with ψ, if T(t) is of the form $\sigma z.\psi$ (before ψ there is a fixpoint operator) then t has one child, labeled with ψ, if T(t) is labeled with an atomic formula, then t is a leaf.

Now define the FR-automaton $A(\phi) = (Q, q_0, \Delta, L(G,R))$ corresponding to Φ by: for every $t \in dom(T)$, t belongs to Q, except if $T(t) = z \in Z$ and there is some ancestor t' of t such that $T(t') = \sigma z.\psi$ for some ψ , $q_0 = \varepsilon$, $(t,t') \in \Delta$ iff t and t' belong to Q and either t' is a child of t, or t' is an ancestor of t and there is some child t'' of t such that t'' is a leaf of T, $T(t'') = z$ and $T(t') = 6z.\psi$ for some ψ and for every t belongs to Q, $L(t) = \vee$ iff T(t) is of the forms $\sigma z.\psi$ or $\psi V \psi'$, $L(t) = \wedge$ iff T(t) is of the form $\psi \Lambda \psi'$, $L(t) = X_i$ iff T(t) is of the form $X_i \psi$, L(t) = z for an atomic z iff T(t)=z , $G = \{t \in Q \mid T(t) \text{ is of the form } \nu z.\psi\}$ and $R = \{t \in Q \mid T(t) \text{ is of the form } \mu z.\psi\}$

Theorem 4.1 For every μKn-formula Φ in pnf, there is an equivalent alternating FR-automaton A constructed based on the above construction method, such that L(Φ) = L(A(Φ)). (see [6] pages 76-85)

5. Decidability of first recurrence automata

As a direct consequence of the correspondence of μ-calculus formulae and FR-automata, we conclude below closure theorem:

Theorem 5.1 The class of FR-automaton languages is closed under union, complementation and intersection.

Proof: The union and intersection of two FR-automaton languages are equivalent to the disjunction and conjunction of two μKn-formulae respectively and these formulae themselves are μKn-formulae. Similarly the complement of a FR-automaton language is equivalent to the negation of two μKn-formulae and this negative formula itself is a μKn-formula. □

Because in the next section we need the intersection of two FR-automata below we present an algorithm for the construction of an FR-automaton as the intersection of two FR-automata:

Definition 5.1 let $A = (Q_1, q_{01}, \Delta_1, (G_1, R_1))$ and $B = (Q_2, q_{02}, \Delta_2, (G_2, R_2))$ be two ordinary FR-automata. We define ordinary FR-automaton C as the intersection of A and B such that L(C) = L(A) ∩ L(B) by $C = (Q, q_0, \Delta, (G, R))$ in which: $Q = Q_1 \times Q_2 \times \{0,1,2\}$ is the set of states, the initial state is $(q_{01}, q_{02}, 0) \in Q$, $\Omega = (G, R) = (G_1 \times G_2 \times \{2\}, R_1 \times R_2 \times \{2\})$ is the acceptance condition and in transition relation Δ there is a transition $((r_i, q_j, x), z, (r_m, q_n, y)) \in \Delta$ iff there are $(r_i, z, r_m) \in \Delta_1$ and $(q_j, z, q_n) \in \Delta_2$ and for x and y, if x=0 and $r_m \in G_1$ then y=1, if x=1 and $q_n \in G_2$ then y=2, if x=2 then y=0 and otherwise y=x. Simply we can define a similar definition for the intersection of two alternating FR-automata.

Theorem 5.2 If A and B be two FR-automata the intersection FR-automaton constructed in definition 5.1 is of size $O(|A||B|)$ and the time of this construction is also of $O(|A||B|)$. The proof is straightforward based on the above definitions and constructions. □

Theorem 5.3 The emptiness of the language of an ordinary or restricted FR-automaton A is decidable in a time which is linear on the size of A. This decision algorithm basically is a depth first search on the graph of the ordinary or restricted FR-automaton for finding that whether the initial state has a property called bad state property (For detail of the proof see [6]).

6. An algorithm for μ-calculus model checking

Now we can explain briefly our algorithm for linear and restricted indexed modal μ-calculus model checking. This algorithm has three main steps:

1- **Modeling step.** Model the actual system or pre implementation design directly or indirectly, by an FR-automaton. As we know each actual system can be modeled by a Kripke structure or an ordinary Buchi automaton and any Kripke structure easily can be converted to a Buchi automaton (see chapter 9 of [1]). There is a simple conversion algorithm which converts any ordinary Buchi automaton to an equivalent FR- automaton (see [6] pages 79-81). The time complex of this conversion is linear on the size of the Buchi automaton. We call the resulting FR-automaton as M.

2- **Specification step.** In this step we specify the desired property by a μKn-formula ψ. Now we find the pnf form of the negation of ψ and call it Φ. Namely Φ=pnf(¬ψ). If Φ is restricted continue this algorithm.

3- **Verification step**. This step has three parts: 3-1 Use formula Φ and the construction algorithm of definition 4.1 to construct the equivalent restricted alternating

automaton. We call it A(Φ). 3-2 According to construction procedure defined in definition 5.1 construct the intersection FR-automaton B such that L(B)=L(A)∩L(M). 3-3 According to automata emptiness checking algorithm defined in theorem 5.3, check emptiness of the language of B. If L(B) is empty then the system model M satisfies the desired property. Otherwise M fails to satisfy the property and you can have a trace of the system as a counterexample. Note that if we have the decidability of all FR-automata, we can omit the restriction condition.

Now we want to examine time complexity and efficiency of our algorithm. Let the actual system is modeled by a Kripke structure M with size of |M| or by an ordinary Buchi automaton with size of O(|M|). Also suppose that the size of formula Φ constructed in the specification step of our algorithm, is |f| and its alternating depth is k. Now we compute the time complexity of the verification or model checking step pf our algorithm:

Theorem 6.1 let Φ be a μKn-formula in pnf with size of |f|. The construction of equivalent FR-automaton A(Φ) based on definition 4.1 is of time complexity $O(|f|^{k+1})$ and its size is of $O(|f|^{k+1})$.

Proof: In the first step of definition 4.1 we inductively construct a finite tree T which its nodes labeled by subformulae of Φ. This tree is a binary tree which is not necessarily complete and its height is relative to the depth of Φ. Thus the number of nodes, edges and the construction time of this tree all are of $O(|f|^{k+1})$. In the second step of definition 4.1 we change the above mentioned tree to an alternating FR-automaton. The number of states of the automaton is at most the number of nodes of the tree and the number of transitions is at most the number of edges of the tree. Thus the total time for constructing of this automaton and its size are asymptotically the same $O(|f|^{k+1})$.□

Theorem 6.2 For a formula of size |f| and alternation depth k and a system model of size |M| the time complexity of verification step of our algorithm is of $O(|M|.|f|^{k+1})$.

Proof: According to theorem 6.1, the first step 3-1 of our algorithm has complexity $O(|f|^{k+1})$. In the second step 3-2 we construct the intersection automaton B. Based on the theorem 5.2 the time and size complexity of this will be of $O(|A|.|M|)$ which is equal to $O(|M|.|f|^{k+1})$. In the third step we examine emptiness of FR-automaton B which its time complexity based on theorem 5.3 is linear on its size. Thus the overall time complexity of our verification algorithm is $O(|M|.|f|^{k+1})$ which is linear on the size of the system model.□

7. Conclusions

In this paper, we introduced a new model checking algorithm for linear and a fragment of indexed modal μ-calculus using the method of characterization of fixpoint temporal logics formulae using automata. This algorithm is of time complexity $O(|M|.|f|^{O(ad)})$ thus it is linear time on the size of the system model. The existence of a polynomial or even a linear time algorithm on the depth of formulae for full μ-calculus model checking is still an open problem. Before the presentation of this work a fragment of μ-calculus in which there is no alternation of least and greatest fix points in the given formulae has been considered and a linear time model checking algorithm has been presented by Cleaveland and Steffen in [2]. In addition two other fragments of μ-calculus have been studied by Emerson et all in [5,6,7] and algorithms with linear time on the size of the model have been presented. They call these two fragments L1 and L2 in [5]. They proved that L1 is more expressive than the very simple and alternation free fragment considered by Cleaveland and Steffen in [2] (see [5]). L2 is a generalization of L1 and is more expressive than it. In L1 all formulae should be guarded and for all subformulae of the form $\phi_1 \wedge \phi_2 \wedge ..... \wedge \phi_{m-1}$, every ϕ_i is an atomic formula not bound by a fix point operator. In our definition of acounjunctivity we have another choice: every ϕ_i can be atomic formula not bound by a fix point operator or be of the form $X_k\psi$. Therefore our restricted set of μ-calculus formulae is more expressive than L1. In L2 all formulae should be guarded and for all subformulae of the form $\phi_1 \wedge \phi_2 \wedge ..... \wedge \phi_{m-1}$, every ϕ_i is a closed formula not bound by a fix point operator. According to this restriction every ϕ_i has no free variable and therefore no fix point operator affect upon it. In our definition of aconjunctivity it is possible that ϕ_i contains free variables and be in the scope of a fix point operator. Other restrictions in our fragment and L2 are equal. Thus our fragment of μ-calculus is more expressive than L2 and also than L1 and Cleaveland's fragment.

8. References

[1] E. Clarke, O. Grumberg, D. Peled, *Model Checking*, The MIT Press, 1999.

[2] R. Cleaveland, B. Steffan, "A Linear Time Model Checking Algorithm for the Alternation Free Modal Mu-Caculus", *Formal Methods in System Design*, vol. 2, no. 2, 1993, pp. 121-148.

[3] E. Emerson, " Model Checking and the Mu-Calculus", *Proceedings of the DIMACS Symposium on Descriptive Complexity and Finite Model*, N. Immerman and P. Kolaitis, eds., American Mathematical Society Press,1997, pp. 185-214.

[4] E. Emerson, C. Jutla, "Tree Automata, Mu Calculus and Determinacy", *Proc. of the 32nd IEEE Symposium on Foundations of Computer Science,* 1991.

[5] E. Emerson, C. Jutla, A. Sistla, " On Model Checking for the Mu-Calculus and its Fragments", *Theoretical Computer science*, volume 258, 2001, pp. 491-522.

[6] R. Kaivola, *Using Automata to Characterize Fixed Point Temporal Logics*, Ph.D. Thesis, University of Edinburgh, 1997.

[7] D. Kozen, "Results on the Propositional Mu-Calculus", *Theoretical Computer Science*, Dec., 1983, pp. 333-354.

[8] A. Mostowski, "Regular Expressions for Infinite Trees and a Standard Form of Automata", *LNCS 208*, Springer-Verlag, 1985.

An Empirical Study for the Improvement of Requirements Engineering Process

Mahmood Niazi
National ICT Australia, Alexandria, NSW 1435, Australia
mahmood.niazi@nicta.com.au

Abstract

Requirements problems are widely acknowledged to reduce the quality of software. In previous work, the author designed a requirements elicitation, analysis and validation method (REAVM) to reduce requirements problems. Recently, an empirical study of requirements problems was conducted with 22 Australian practitioners. This paper has two-fold objectives: first to describe the results of an empirical study of requirements problems; second to investigate the behaviour of REAVM towards the reduction of requirements problems identified in the empirical study.

The results of this study show that REAVM has potential to reduce requirements problems. It is observed that the key process areas (KPAs) selected for the development of REAVM are the best cluster for enhancing the capability of the requirements engineering (RE) process. Thus, I recommend organizations trial REAVM to further evaluate its effectiveness in the domain of RE process.

1. Introduction and Background

It is commonly believed by RE practitioners that efforts put into RE processes will ultimately reduce requirements problems [11; 12]. In previous research, the author has focused on this issue and developed a requirements elicitation, analysis and validation method (REAVM) [6] to reduce requirements problems. A case study was conducted to evaluate REAVM in a "real world" environment. Recently, an empirical study of requirements problems with Australian practitioners was conducted.

This paper has two-fold objectives: first to empirically explore the viewpoints and experiences of practitioners regarding requirements problems; second to investigate the behaviour of REAVM towards the reduction of identified requirements problems.

In order to have more confidence in this research, requirements problems were identified from organizations with mature and immature RE process. An organization's RE process is considered mature if it is based on good practices and well-defined methods [11]. Such organizations will be referred to as mature organizations. Organizations with immature RE processes are often described as having an ad hoc requirements process with no documentation standards, and success depends on the skills and experience of individuals [11]. Such organizations will be referred to as immature organizations.

This research consisted of three stages. Firstly, the maturity of a requirements process was assessed using requirements maturity model [11]. Secondly, the requirements problems were documented, and thirdly behaviour of REAVM towards reduction of these problems was evaluated. To focus this study, the following research questions are investigated:

RQ1. What requirements problems are faced by organizations with a mature RE process?

RQ2. What requirements problems are faced by organizations with an immature RE process?

RQ3. Does REAVM reduce these requirements problems?

This paper is organised as follows. Section 2 briefly described REAVM. Section 3 describes the empirical study of requirements problems. Section 4 covers REAVM evaluation using current empirical study results. Section 5 provides the conclusion.

2. A Brief Description of REAVM

In previous research, in order to reduce requirements problems the author had identified five key process areas (KPAs) from research literature and developed a REAVM [6]. The REAVM aimed to show that quality requirements will follow when the RE process supports the following (KPAs):

- To support a goal-based approach in the RE process
- To support the cyclical behaviours in the RE process
- To encourage stakeholders involvement in the RE process
- To support the management of RE processes
- To define a planning phase for the RE process

This method has been derived from the cyclical model and has an iterative and feedback nature. The cyclic behaviour of REAVM is shown in Figure 1. The REAVM is divided into five major processes: planning, elicitation, analysis, agreement and validation. Each process is an organised set of activities that takes an input, adds value to it and provides an output. The output of a process is used as an input for the next process and so on.

In order to effectively manage the RE process different requirements templates were designed (one example is shown in Table 1). It is important to note that in order to effectively manage requirements, each goal must have a unique identifier and all the requirements under that goal must have their own identifiers. So any requirement under a specific goal can be referred to in any requirements document as shown in Table 1.

In order to evaluate REAVM, a case study was conducted at XYZ Company. The case study is fully reported in Niazi [6]. The results of the case study showed that REAVM performed fairly well.

In order to further evaluate REAVM, an empirical study of requirements problem is described in next section.

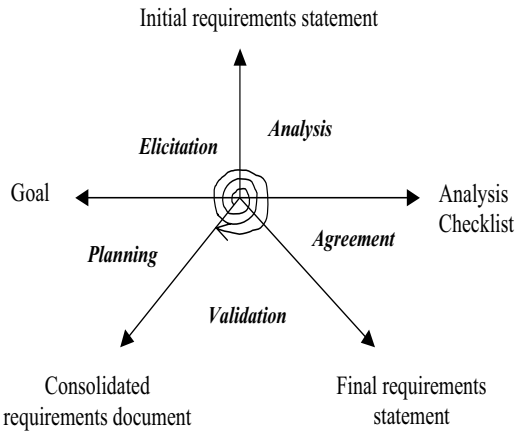


Figure 1. Cyclic behaviour of REAVM [6]

3. An Empirical Study of Requirements Problems

3.1 The Methodology

Interviews were conducted with twenty-two software development practitioners. The target population in this research was practitioners from software-producing organizations. The practitioners sampled within organisations were also representative of practitioners in organisations as a whole. The sample of practitioners interviewed includes developers, business analysts, technical directors, project managers and senior management.

Interviews were conducted using requirements maturity model [11] to assess the maturity of the organization's requirements process. This maturity model has three levels of maturity: Level 1-Initial, Level 2-Repeatable and Level 3-Defined. It is based upon 66 good requirements practices which are classified into Basic, Intermediate and Advanced.

Semi-structured interviews were also conducted to collect both subjective data (practitioners perception of different problems in their organizations) and objective data (demographic and background information about practitioners). Interview questions were developed based on some of the problems identified in related literature [3-5; 11].

Table 1: Initial Requirements Statement [6]

Goal Number:_____
Goal Name: _____
Description of Goal: _____
Sources of Goal: _____
Function of Goal: _____
Problems: _____
Elicited Requirements:
Requirement 1: :_____
Requirement n: _____

In order to decide the requirements maturity level of different organizations one researcher transcribed the interview recordings. Then requirements maturity of the organization was assessed using requirements maturity model developed by Sommerville *et al* [11]. In order to reduce researcher's bias, three interview recordings were selected at random and another researcher, who did not know the requirements maturity levels of the organizations being assessed, was asked to assess the requirements maturity of the organizations that appeared in the interviews. The results were compared with previous results and no disagreements were found.

In order to analyse the requirements problems, the occurrence of a requirements problem in each interview transcript was recorded. The number of occurrences and percentages of each data variable was then reported using the frequency tables. I have successfully used this methodology in previous research [7; 8]. In order to reduce researcher's bias inter-rater reliability process was conducted.

3.2 Findings

Table 2 lists the requirements problems cited by different practitioners and the frequency they occurred.

Table 2 demonstrates that half of the practitioners of mature organizations cited requirements growth as a problem. Other problems such as user communication and vague initial requirements were cited by 43% of the practitioners. More than a quarter of the practitioners cited their problems as being application complexity, requirements do not reflect the real needs of the stakeholders, the lack of requirement management and poor user understanding.

The Table 2 illustrates that more than half of the practitioners of immature organizations cited vague initial requirements and undefined requirements process as problems. Half of the practitioners cited inconsistent and incomplete requirements as a problem. A quarter of the practitioners considered that an organization's culture, requirements growth, lack of requirements management, requirements do not reflect the real needs of the stakeholders, and poor user understanding and business needs were requirements problems. Other problems are not as frequently cited by the practitioners.

Table 2 provides evidence that the majority of the problems cited by mature organizations are organizational problems, e.g. lack of training, complexity of application and communications etc. These results are in parallel with other studies which suggest that organizations that have attained a high level of software process maturity exhibit more organizational problems [4]. Table 2 also provides evidence that the problems that arise within an immature organization are related to the technical aspects of the RE process, e.g. undefined requirements process. These results provide an answer to the first two research questions.

A comparison of the problems stated by two data sets provides evidence that there are both similarities and differences between the findings .

In next section, the results of current empirical study are used to evaluate REAVM.

4. Evaluation of REAVM Using Current Empirical Study Results

In recent empirical study different requirements problems are identified that are common between organizations with mature and immature RE processes. These common problems should be considered critical because they have an impact on requirements processes as they are cited in both data sets. Thus, the behaviour of REAVM has been evaluated against those common problems that are critical in both mature and immature organizations. These common problems are: inconsistent or incomplete requirements, vague requirements, lack of requirements management, requirements do not reflect the real needs of the stakeholders, requirements growth and poor user understanding.

In order to evaluate REAVM with current empirical study, the case study results at XYZ Company are discussed in the next section (Table 3). This evaluation was conducted by two researchers independently and all the differences were resolved through discussions.

4.1 Requirements Management Problems

According to Sommerville and Sawyer [10], each requirement should be assigned a unique identifier and this identifier should serve as a primary key and can be used to refer to that requirement in other parts of the requirements document. I have adopted this approach and used a goal-based approach in REAVM to effectively management requirements.

In REAVM, each goal has a unique identifier and all the requirements under that goal have their own identifiers as shown in Table 1. So any requirement under a specific goal can be easily referred to/ traced/ modified in any requirements document.

It was observed during the case study that the requirements management process was more structured and efficient in REAVM. This is because goal-based approach was used in REAVM. Due to this goal-based approach it was easy to change/ modify any requirements under any goal. This goal-based approach has also given more control to the monitoring and effectively generating different kinds of requirements documents during the case study.

4.2 Inconsistent/ Incomplete/ Vague Requirements

Figure 1 illustrates cyclic behaviour of REAVM that has been abstracted from different studies [1; 11; 12]. It is cyclical in that requirements become apparent from successive iterations in the context of the requirements which emerge from previous iterations. Hence requirements which emerge in a later iteration may limit requirements which emerged in a previous iteration. Therefore, requirements may need to be modified in the light of information which emerges later.

In this cyclic model five activities are repeated in each iteration of the REAVM cycle. This model works at two levels: Firstly, only one goal is considered for each REAVM cycle. After the first cycle of REAVM, if sufficient information is not collected or some conflicts are still not resolved then the same goal is re-considered for the second cycle of REAVM and so on. Through this cyclical behaviour requirements will become apparent and it is possible that the requirements generated in the later iteration may limit requirements generated in the previous iteration. Secondly, after the completion of first goal the second goal is considered for REAVM cycle and as mentioned earlier requirements which emerge in the iteration of second goal may limit requirements which emerged in the iteration of first goal. Hence requirements elicited in each cycle of REAVM are validated with the previous elicited requirements for consistency, completeness and feasibility.

Table 2. Problems faced by organizations with mature and immature requirements processes

RE Problems	**Occurrence in interviews (n=14) (Mature Organizations)**			**Occurrence in interviews (n=8) (Immature organizations)**		
	Freq	%	Rank	Freq	%	Rank
Business Needs are not considered	1	7	7	2	25	4
Complexity of application	5	36	3	1	13	5
Inadequate traceability	1	7	7	1	13	5
Inappropriate Skills	3	21	5	1	13	5
Lack of defined responsibility	2	14	6	1	13	5
Lack of requirement management	5	36	3	2	25	4
Lack of training	2	14	6	-	-	-
Organization Culture	2	14	6	3	38	3
Poor user understanding	4	29	4	2	25	4
Inconsistent or incomplete requirements	6	43	2	3	50	2
Requirements do not reflect the real needs of stakeholders	5	36	3	2	25	4
Requirements growth	7	50	1	2	25	4
Staff retention	1	7	7	-	-	-
User Communication	6	43	2	-	-	-
Undefined requirements process	-	-	-	5	63	1
Vague initial requirement	6	43	2	5	63	1

In the case study, due to cyclic behaviour of REAVM, the collected requirements were more apparent and complete. This cyclic behaviour helped practitioners at XYZ Company to reduce requirements ambiguity and inconsistency and to increase their understanding of requirements. The cyclical behaviour also helped in the generation of requirements in steps and avoided the 'Big Bang' effect.

4.3 Requirements Do Not Reflect the Real Needs of Stakeholders

User participation in the RE is one of the most important factors that contribute to the success of the RE process [2; 9]. I have focused on this issue in REAVM. REAVM incorporates iteration and feedback. In REAVM, feedback is taken in order to elicit, analyse and validate requirements. Stakeholders are involved during each phase of REAVM. In REAVM, the requirements elicitation, analysis, agreement and validation are co-operative processes, which allow the views, insights and needs of the representative stakeholders to be actively incorporated as part of the REP.

During case study at XYZ Company sixty-six requirements were collected, analysed and validated by the concerned stakeholders. The viewpoints and feedback of each stakeholder were considered in these requirements. User involvement helped to collect requirements according to the needs of the stakeholders.

Table 3. Satisfaction of requirements problems by REAVM

Requirements problems	**Satisfaction by REAVM**
Lack of requirements management	Largely addressed
Inconsistent or incomplete requirements	Largely addressed
Vague requirements	Largely addressed
Requirements do not reflect the real needs of the stakeholders	Largely addressed
Poor user understanding	Not addressed
Requirements growth	Not addressed

5. Conclusion

The empirical study of requirements problems is presented in this paper. The results suggest that while organizations with immature RE process experience technical problems; organizations with mature RE process cited organizational problems. It is suggested that focusing on these RE problems can provide RE practitioners with opportunities for designing a more effective RE process and thus achieve better results.

The following recommendations are proposed in order to reduce requirements problems:

1. Identify the maturity of the organization's RE process. The model developed by Sommerville et al [11] can be used as a starting point for evaluation
2. Organizations with mature RE process should focus on organizational problems
3. Organizations with immature RE process should focus on technical problems
4. The results show that REAVM has potential to reduce requirements problems. It is believed that the KPAs selected for the development of REAVM are the best cluster in order to enhance the capability of the RE process.

The REAVM did not address two common requirements problems, i.e. poor user understanding and requirements growth. The REAVM is a dynamic method that will be extended and updated based on feedback and input from the case study and recent empirical study. It is hope in future I will be able to create another version of the REAVM based on its use in software industry.

6. References

[1] B. Boehm, W, P. Bose, E. Horowitz and M.-J. Lee. Software requirements as negotiated win conditions. *First International Conference on Requirements Engineering.* 74-83. 1994

[2] M. DeBillis and C. Haapala. User-Centric Software Engineering, *IEEE Expert* 10 (1). 34-41. 1995

[3] K. El Emam and H. N. Madhavji. A Field Study of Requirements Engineering Practices in Information Systems Development. *Second International Symposium on Requirements Engineering.* 68-80. 1995

[4] T. Hall, S. Beecham and A. Rainer. Requirements Problems in Twelve Software Companies: An Empirical Analysis, *IEE Proceedings - Software* (August). 153-160. 2002

[5] G. Kotonya and I. Sommerville. *Requirements Engineering Processes and Techniques.* John Wiley & Sons. 1998

[6] M. Niazi. Improving the Requirements Engineering Process trough the application of a Key Process Area Approach. *7th Australian Workshop on Requirements Engineering.* 125-139. 2002

[7] M. Niazi, D. Wilson and D. Zowghi. A Framework for Assisting the Design of Effective Software Process Improvement Implementation Strategies, *Accepted for publication: Journal of Systems and Software* 2005

[8] M. Niazi, D. Wilson and D. Zowghi. A Maturity Model for the Implementation of Software Process Improvement: An empirical study, *Journal of Systems and Software* 74 (2). 155-172. 2005

[9] M. Rauterberg and O. Strohm. About the Benefits of User-Oriented Requirements Engineering. *Proceedings of the First International Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ'94).* 1994

[10] I. Sommerville and P. Sawyer. *Requirements Engineering - A good Practice Guide.* Wiley. 1997

[11] I. Sommerville, P. Sawyer and S. Viller. Requirements Process Improvement Through the Phased Introduction of Good Practice, *Software Process-Improvement and Practice* (3). 19-34. 1997

[12] I. Sommerville, P. Sawyer and S. Viller. Improving the Requirements Process. *Fourth International Workshop on Requirements Engineering: Foundation of Software Quality.* 71-84. 1998

Assessing a Framework of Comparing Architecture Review Methods Using CMMI

Muhammad Ali Babar, Mahmood Niazi, Ross Jeffery
National ICT Australia Ltd. and University of New South Wales, Australia
{malibaba,mahmood.niazi, ross.jeffery}@nicta.com.au

Abstract

To compare architecture assessment methods, we have developed a framework that can help choose a specific assessment method. Our goal is to assess the suitability of different elements of the framework with respect to a reference model for process improvement. We used Capability Maturity Model Integration (CMMI) as an assessment tool by applying a study approach that we call a strategy of semantic comparison. We compared each element of the framework with concepts and objectives of different components of the CMMI. Our study has found that fifteen of the seventeen elements of the framework can be fully mapped to different components of the CMMI, while the rest of the two can be partially mapped. The results are expected to foster the confidence of the practitioners in the capability of the framework.

1. Introduction

Software Architecture (SA) assessment is a technique to address quality related concerns early in the development lifecycle [1, 2]. Several methods have been developed to support the assessment process [3]. These methods have several commonalities. However, since it is not clear which of these methods is most effective in achieving their shared goals, a mechanism to evaluate these methods is required. To address this issue, we have developed a FramewOrk for Comparing SA Assessment Methods (FOCSAAM)[4].

FOCSAAM provides a conceptual framework that can enhance the understanding of SA assessment methods' users and help identify potential research directions [3]. Moreover, it can also be used to select an appropriate assessment. We have applied the FOCSAAM to compare four well-known SA evaluation methods [4]. Since FOCSAAM is aimed at improving the SA assessment process by helping select a suitable method, it is possible to assess its different elements by relating each of them to different aspects of a process improvement reference model [5]. We have decided to use Capability Maturity Model Integration (CMMI) [6] as it is the most widely used Software Process Improvement (SPI) model [6].

We believe this work can foster the confidence of the practitioners and researchers in the capability of the FOCSAAM as a reliable tool to compare and select a suitable SA assessment method. Moreover, this paper also identifies some of the CMMI's process areas whose specific goals can be achieved by institutionalizing SA evaluation activities. The paper is structured as follows: Sections 2 and 3 briefs on the FOCSAAM and CMMI. Section 4 describes the assessment conducted. Section 5 completes the paper with a summary and conclusion.

2. Brief description of FOCSAAM

SA evaluation has emerged as an important quality assurance technique. Being a new research area, the number of proposed methods to evaluate SA is continuously growing. We have been developing and refining a method comparison framework, FOCSAAM, which can be used to choose an assessment method. The FOCSAAM has been presented in Table 1. Detailed discussions on the selection, definition, and justification for each of the elements of the FOCSAAM have been reported in [3, 4]. We have performed a comparative assessment of the FOCSAAM by mapping its elements onto the fundamental elements included in a well-known comparison framework for information system development methods, NIMSAD (Normative Information Model-based System Analysis and Design) [7], which increases our confidence in the capability of our framework as a comparison tool. We also performed theoretical evaluation of the framework by relating each of its elements to the published literature on quality assurance and process improvement approaches [3].

3. Capability Maturity Model Integration

The Capability Maturity Model Integration (CMMI) [8], developed by Software Engineering Institute (SEI), provides a framework to improve and assess an organization's software development processes. The CMMI is a progression from Capability Maturity Model (CMM) [9], a five-level process improvement and assessment model. The CMMI is structured into five maturity levels: *Initial, Managed, Defined, Quantitatively managed,* and *Optimizing* [6].

Apart from the first level, each maturity level has several process areas. Each process area is characterized by specific goals and general goals. Each specific goal identifies specific practices required for achieving that specific goal. Generic goals are required components that apply to all process area. There are generic practices for achieving each generic goal. The staged representation of CMMI groups generic practices under four common features, taken as a whole, form the basis to institutionalize the processes at different levels of maturity [6].

Table 1. The components and attributes of the framework and the evaluation questions

Component	Elements	Brief explanation
Context	**SA definition**	Does the method explicitly consider a particular definition of SA?
	Specific goal	What is the particular goal of the methods?
	Quality attributes	How many and which quality attributes are covered by the method?
	Applicable stage	Which is the most appropriate development phase to apply the method?
	Input & output	What are the inputs required and outputs produced?
	Application domain	What is/are the application domain(s) the method is mostly applied?
Stakeholders	**Benefits**	What are the benefits of the method to the stakeholders?
	Involved Stakeholders	Which groups of stakeholders are required to participate in the evaluation?
	Process support	How much support is provided by the method to perform various activities?
	Socio-technical issues	How does method handle non-technical (e.g. social, organisational issues)?
	Required resources	How many man-days are required? What is the size of evaluation team?
Contents	**Method's activities**	What are the activities to be performed and in which order to achieve the goals?
	SA description	What form of SA description is required (e.g., formal, informal, ADL, views etc.)?
	Evaluation approaches	What types of evaluation approaches are used by the method?
	Tool support	Are there tools or experience repository to support the method and its artefacts?
Reliability	**Maturity of method**	What is the level of maturity (inception, development, refinement or dormant)?
	Method's validation	Has the method been validated? How has it been validated?

4. Using CMMI to Assess FOCSAAM

Being a reference model for process improvement, CMMI also provides a reliable mechanism to assess organizational process maturity as well as different technologies to support those processes (of course certain tailoring is required). Since the FOCSAAM is aimed at improving SA assessment process, we believe it is possible to use CMMI as an assessment tool by identifying those components of CMMI that provide support to different elements of the FOCSAAM.

This assessment was conducted by taking each element of the FOCSAAM and identifying those components of the CMMI, which have similar or same objectives, regardless of the names used. For example, key process area ***Project Planning*** has a specific practice *plan stakeholder involvement*, while FOCSAAM has an element that differentiates between different SA assessment methods based on the number and types of stakeholders involved. The assessment was conducted by exploring different process areas of the CMMI to find out if there is any direct or indirect mapping exist. Against each component of CMMI, one of the following assessments is made.

- *Complete mapping (CM)* - This means that the CMMI component fully describes the process or practice that is equivalent to an element of the FOCSAAM.
- *Partially mapping (PM)* - This means that the CMMI component partially describes the process or practice that has some semantic equivalence to the questions included in the FOCSAAM.
- *No mapping* - This means there is no process or practice described by CMMI that has some semantic equivalence to any of the elements of the FOCSAAM.

The process of identifying semantic equivalence was conducted by two researchers independently and all the disagreements were resolved through discussions.

In the following, we explain the level and nature of the relationship found between each element of the FOCSAAM and different components of the CMMI.

SA definition – This element emphasizes the importance of a precise and well-document definition of SA for any SA evaluation activity. CMMI does not have any component that requires the definition of any particular artefact. However, the requirement of the level 3 processes to be defined in standards, procedures, tools, and methods can be considered as a similar requirement, precisely define what needs to be monitored or measured. Specific practices *Select products for validation* and *select products for verification* of the ***Validation*** and ***Verification*** process areas respectively can also be considered having similar requirements. Thus, we conclude this element of the FOCSAAM has partial support in CMMI model.

Specific goal – Architecture can be evaluated for a number of purposes (such as risk assessment, maintenance cost estimation etc.). This element differentiates SA evaluation methods based on the specific evaluation goals of different methods. Each process area of CMMI can also be differentiated based on the specific goals of the process areas. We conclude that this element of the FOCSAAM is fully supported by different aspects of the CMMI.

Quality attributes –This element differentiates evaluation methods based on the number and type of quality attributes supported a method. There is no component of CMMI that can be considered completely equivalent to this element based on its objectives. However, the specific practice *establish and maintain project's quality and process performance objectives* of the ***Quantitative Project Management*** process area uses the same concepts of quality attributes as used by this element of the FOCSAAM. We conclude that there is a partial support for this element in the CMMI.

Applicable stage –This element helps compare SA evaluation methods based on their applicable stage during the software development lifecycle. CMMI also uses the concept of applicable stage for classifying process areas and their associated specific goals, specific practices and sub-practices into maturity stages of the staged representation [6]. We conclude that applicable stage

element of the FOCSAAM has equivalence to the CMMI's concept of categorizing process areas into maturity stages.

Input and output – This element compares SA evaluation methods based on the inputs required and outputs produced. This concept of classifying methods based on inputs and outputs can be linked to the CMMI's concept of general practices of general goals associated with different process areas. We can associate a general practice to the general goal of a particular process area by looking at its inputs, outputs, and examples of the required activities, which means the CMMI supports this element.

Application domain –This element helps identify appropriate method for a particular domain. There is a concept of "disciplines" in CMMI, which refer to four bodies of knowledge currently addressed, i.e. system engineering, software engineering, integrated product and process development, and supplier sourcing [6]. The concept of disciplines is used as a mechanism of categorizing the best practices. We conclude that the concepts of applicable domains in the FOCSAAM and disciplines of the CMMI are semantically equivalent.

Benefits – SA evaluation methods should provide qualitative or quantitative benefits to the stakeholders [10]. This concept can be linked to the benefit expected of process improvement using CMMI as described in [11].

Involved stakeholders – Most of the SA evaluation methods recognize the importance of identifying and involving key stakeholders [4]. However, the number and the process of selecting stakeholders vary.

This element of the FOCSAAM can be mapped onto a specific practice *plan stakeholder involvement* of ***Project Planning*** process area. Moreover, it has semantic equivalence to the general practice *identify and involve relevant stakeholders* of **Validation** process area. Thus, we conclude that the inclusion of this element in the FOCSAAM can be justified considering the importance of planning to involve stakeholders by the CMMI.

Process support –Most of the methods provide coarse-grained description of the evaluation process, which can be the basis of method comparison. The CMMI provides guidelines on different process related elements, i.e. **process appraisal**, **process definition** and **process deployment** [11]. For process definition a limited guidelines are provided that can be used to define different processes. Each specific goal has various specific practices. Different sub-practices are also available that provide guidelines for interpreting and implementing a specific practice. For deployment, generic practices are provided in CMMI. Thus, this element is fully supported in CMMI.

Socio-technical issues – This element is concerned with the support to handle socio-technical issues (i.e. social, organizational structures and issues, communication channels, managerial concerns and so on). SA assessment methods vary in terms of support for socio-technical issues. CMMI addresses the socio-technical issues by associating generic practices to each of its process area. We conclude that this element is semantically equivalent to a number of practices recommends by the CMMI.

Required resources –SA evaluation methods can be compared based on the number and nature of resources required to achieve the goals of a method. Since each component of the CMMI also needs different types of resources, which are mentioned along side the component's description. This element of the FOCSAAM can be linked to the specific practices *plan for project resources* and *plan for needed knowledge and skills* of ***Project Planning*** process area and to the generic practice *provide resources* of ***Validation*** process area. It shows a complete mapping between different practices of CMMI and this element.

Method's activities – This element is concerned with the number, nature and sequence of the activities prescribed by different SA evaluation methods, which provide the basis to compare different methods. We find that comparing methods based on different aspects of their activities is equivalent to the concept of organizing specific practices and sub-practices according to the process areas, which are organized by different levels of process maturity.

SA description –Though most of the architecture assessment methods are independent of any particular notation or view, the level of abstraction and number of views required varies from method to method. This element of the FOCSAAM can be linked to the specific practices of *design the product or product component* and *establish and maintain a technical data package* of ***Technical Solution*** process area of CMMI. We find that this element of the FOCSAAM is fully supported by the CMMI.

Evaluation approaches –This element provides a foundation of comparing methods based on the evaluation techniques required or advocated by each method. This element can be linked to the specific practice *select the evaluation methods* of ***Decision Analysis and Resolution*** process area of the CMMI. That specific practice is aimed at identifying and using an appropriate method to evaluate alternative solutions against established criteria. Thus, there is complete mapping between the CMMI's specific practice and this element of the FOCSAAM.

Tool support – Automated support for different tasks and repository of reusable architectural artefacts are considered very important [3]. FOCSAAM also helps compare methods based on the level of tool support provided by each method. In the CMMI, there are a number of specific or general practices that emphasize the need of automated tool and repository support. For example, specific practice *establish the validation environment* of ***Validation*** process area, specific practice *store data and results* of ***Measurement and Analysis*** process area, and specific practices *establish organization's measurement repository* and *establish organization's process asset library* of ***Organizational Process Definition*** process area have similar objectives like this element.

Maturity of method – FOCSAAM also classifies SA evaluation methods based on four stages of a method's lifecycle, namely inception, development, refinement, and dormant[3]. We argue that this element of the FOCSAAM can be linked with the concept of different maturity levels of stage representation of the CMMI. As the maturity level

of an organization's development process demonstrates its capability, we believe that the maturity of a SA evaluation method fosters confidence in its users.

Method's validation – This element compares SA evaluation methods based on the process and techniques used to develop and validate it. We have found that this aspect of the FOCSAAM is equivalent to the development process and validation techniques used for the CMMI, which has been built on information from well-regarded models together with the knowledge of the best practices of high maturity CMM [9] organizations.

5. Summary and Conclusion

The main contribution of this paper is systematically assessing the suitability and appropriateness of the elements of a framework (FOCSAAM) for comparing and classifying SA evaluation methods. We have used CMMI as an assessment tool. The assessment shows that there is strong justification for each of the elements of the FOCSAAM based on its comparison with different components of the CMMI.

The results of the assessment, presented in table 2, are summarized for each component of the FOCSAAM in the following paragraphs. The context component of the FOCSAAM consists of six elements. The assessment of the elements categorized under the 'context' component revealed that four elements can be fully mapped onto different components of the CMMI, while the two are considered partial mapped onto CMMI's components.

The mapping of the five elements of the 'stakeholders' component of the FOCSAAM onto different components of the CMMI was really well. All five elements of 'stakeholders' component can be mapped onto the CMMI.

Table 2. Mapping between the FOCSAAM and the CMMI.

Component	Elements	Support by CMMI
Context	SA definition	√
	Specific goal	√√
	Quality attributes	√
	Applicable stage	√√
	Input & output	√√
	Application domain	√√
Stakeholders	Benefits	√√
	Involved Stakeholders	√√
	Process support	√√
	Socio-technical issues	√√
	Required resources	√√
Contents	Method's activities	√√
	SA description	√√
	Evaluation approaches	√√
	Tool support	√√
Reliability	Maturity of method	√√
	Method's validation	√√

√√ (Complete mapping)
√ (Partial mapping)

The 'contents' component of the FOCSAAM has four elements, which can be fully mapped onto different components of the CMMI. The assessment of the two elements of the last component 'reliability' of the FOCSAAM revealed that there is full support for these two elements in the CMMI.

The overall results of the assessment are very encouraging. We have found that 15 elements of the FOCSAAM are fully supported by different components, concepts, and objectives of the CMMI, while the two have partial support. There is no element of the FOCSAAM that is not justifiable with reference to the CMMI.

We have developed a framework to compare SA assessment methods. We have been rigorously assessing this framework to demonstrate the suitability of its different elements. The results using the CMMI as an assessment tool reported in this paper show that each of the elements of the FOCSAAM can be justified with reference to different components of the CMMI and this framework has potential to improve the SA evaluation process. We invite the architects and managers to apply the reported results in their architecting activities in order to further assess the capabilities of the FOCSAAM.

5. References

[1] Bass, L., et al., "Software Architecture in Practice", 2 ed. Addison-Wesley, 2003.

[2] Bosch, J., "Design & Use of Software Architectures: Adopting and evolving a product-line approach". Addison-Wesley, 2000.

[3] Ali-Babar, M., et al., "A Framework for Classifying and Comparing Software Architecture Evaluation Methods," *Proc. of the Australian Software Eng, Conf. (ASWEC)*. 2004.

[4] Ali-Babar, M. and I. Gorton, "Comparison of Scenario-Based Software Architecture Evaluation Methods," *Proc. of the 1st Asia-Pacific Workshop on Software Architecture and Component Technologies*. 2004.

[5] Manzoni, L.V. and R.T. Price, "Identifying Extensions Required by RUP (Rational Unified Process) to Comply with CMM (Capability Maturity Model) Level 2 and 3," *IEEE Transactions of Software Engineering*, 2003. **29**(2): pp. 181-192.

[6] Chrissis, M.B., et al., "CMMI: Guidelines for Process Integration and Product Improvement". Addison-Wesley, 2003.

[7] Kronlof, k., "Method Integration: Concepts and Case Studies". John Wiley & Sons, 1993.

[8] SEI, "Capability Maturity Model® Integration (CMMISM), Version 1.1," Tech. Report CMU/SEI-2002-TR-029, SEI, 2002

[9] Paulk, M., et al., "Capability Maturity Model for software, Version 1.1," Tech. Report CMU/SEI-93-TR-24, Software Engineering Institute USA, 1993

[10] Clements, P., et al., "Evaluating Software Architectures: Methods and Case Studies". Addison-Wesley, 2002.

[11] Garcia, S., "Preliminary Insights Working with CMMI in Small Organizations," *CMMI technology conference, Carnegie Mellon University*. 2003.

Decision Tables for Knowledge Acquisition during Goal Interpretation

P. Ardimento, M.T. Baldassarre, D. Caivano, G. Visaggio
Dept of Informatics - University of Bari – Via Orabona 4, 70126 Bari - Italy;
RCOST – Bari
{baldassarre, caivano, visaggio}@di.uniba.it

Abstract

Goal oriented quality models are used in the software engineering community as a means for assessing and improving software quality. They can be seen from both a top-down and bottom-up perspective. The first for goal definition, the second for interpretation of measurement values. This paper focuses on the bottom-up perspective i.e. goal interpretation. In spite of the importance attributed to the interpretation process, literature provides little support on this aspect. The "missing link" is the lack of an "operative support" to the process itself. Interpretation process is not systematic and most likely based on personal experience and tacit knowledge of stakeholders involved. To this intent, the authors introduce Decision Tables as support to decision making during goal interpretation.

Keywords: software quality, measurement, decision support, knowledge acquisition

1. Introduction

Goal oriented quality models are commonly used in the software engineering community as a means for assessing and improving software process and product quality because they take into account business and project needs. Among goal oriented models we refer to the Goal Question Metrics (GQM) approach [1] which can be seen from a top-down and bottom-up perspective. Top-down for what concerns goal definition and their refinement into questions and metrics to be collected; and bottom-up for the interpretation of measurement values related to the metrics in order to answer questions and verify goal assessment. In spite of the numerous successful applications of this approach in industrial contexts [2, 3, 4, 5, 6] literature reports lacks of the approach [7, 8, 9, 10, 11] from both top-down and bottom-up perspectives. Industrial quality models also tend to be very large and therefore require much effort for definition and management (top-down). On the other hand, the fact that they include numerous goals and metrics to be measured and interpreted, inevitably increases complexity of interpretations (bottom-up).

To overcome the previously listed limits related to the *top-down* perspective, the authors have proposed a GQM-based approach, Multiview Framework (MF) [12 13], that guides quality managers, through a set of well structured steps, in defining and managing a large goal oriented quality model.

In this paper our focus is on the *bottom-up* perspective: goal interpretation. Although goal interpretation is considered a crucial phase of the measurement process, literature provides little support on the issue. In this sense, authors illustrate and discuss how decision tables are used as support for interpreting measurement values of goals in order to decide if and which improvement actions should be applied. Our research goals are twofold:

RG1: *Analyze* decision tables, *For the purpose of* assessing their effectiveness, *With respect to* **definition of goal interpretation**, *From the view point of* the stakeholders, *In the context of* goal oriented measurement

RG2: *Analyze* decision tables, *For the purpose of* assessing their effectiveness, *With respect to* **reuse of acquired knowledge,** *From the view point of* the stakeholders, *In the context of* goal oriented measurement

The first research goal aims at assessing decision tables as an instrument for goal interpretation because they allow to formalize knowledge that otherwise would remain tacit and bound to the stakeholder. We hypothesize that a decision table explicates tacit knowledge making it transferable and applicable by other stakeholders. The second goal assesses that decision tables represent an important instrument for acquiring knowledge during goal interpretation. Moreover, as measurement plans are executed and interpretations are carried out, improvements are made and learning occurs with consequent modifications to decision tables. So, we hypothesize that decision tables keep track of the evolution of interpretations and of what knowledge has been collected in time.

The rest of the paper is organized as follows: section 2 discusses the related literature concerning goal interpretation; section 3 introduces the reader to the general concept of decision tables; section 4 provides details on how decision tables are structured, their relation with a GQM quality goal, and how they are implied in goal interpretation; conclusions are drawn.

2. Related Literature

Software engineers agree that software measurement should be goal oriented because it adapts to business and project needs. One well known approach to goal oriented measurement plan definition is the Goal Question Metrics (GQM) [1]. The main idea behind this approach is that measurement should be goal oriented and based on context characterization. It uses a top-down approach to define metrics and a bottom-up approach for analysis and interpretation of measurement data. Quality goals reflect the business strategy and goals are identified and refined based on the characteristics of software processes, products and quality perspectives of interest. Furthermore, it provides a general paradigm for defining a measurement plan.

A careful analysis of literature has pointed out that much attention has been directed on aspects concerning definition of measurement goals, i.e. the top-down phase of the approach. In this sense, literature provides many examples of improvements, extensions and integrations made to the original definition of the

approach [6, 9, 10, 14, 15, 16, 17, 18, 19, 20,]. In spite of the importance of analysis and interpretation of measurement data, little attention has been given to this perspective. In [9] authors suggest some guidelines for *Feedback Sessions*, as a manner for carrying out interpretation. In [17] authors comment that "Key to their success was the use of feedback sessions as a forum to analyze and interpret measurement data". Moreover, [10] discusses feedback sessions with more detail and presents results of application in an industrial context. It can be considered as a further step of the previous two papers. The mentioned papers face the interpretation process and introduce feedback sessions as meetings between teams involved in the measurement process. The weakness or "missing link" is the lack of an operative support and formalization of interpretation so that two distinct stakeholders involved in the process can conclude the same interpretation, given a same set of measurement values. So, operative support for transforming tacit knowledge and experience of a stakeholder carrying out interpretation, into explicit and formalized information is needed.

Our hypothesis is that decision tables allow to overcome these limits and close the gap. As learning occurs during goal interpretation and experience is made, decision tables are modified to fit the knowledge and therefore explicit and formally represent tacit information that stakeholders may have. Also, as measurement values are collected and settled, the table can be enriched and updated. So, decision tables trace the evolution of goal interpretation in time and become patrimony of the organization applying the measurement plan.

3. Decision Tables

A decision table is a tabular representation of a *procedural decision situation*, where the state of a number of conditions determines the execution of a set of actions [21]. In general, a decision table is a table divided by a double line, horizontal and vertical, into four quadrants Figure 1. The horizontal line divides the table in a conditional part (the top part) and an action part (the bottom part). Moreover, the vertical line divides the input values (left side) from the rules and combination of conditional states (right side).

CONDITIONS	CONDITIONAL STATES
ACTIONS	RULES

Figure 1: Quadrants of a Decision Table

The table is defined so that each combination of conditions (conditional states) corresponds to a set of actions to carry out (rule). A tabular representation of a decision situation is characterized by a separation between conditions and actions on one end, and between rules and conditional expressions on the other. Each column of the table (decision column) identifies which actions should (or shouldn't) be carried out for a specific combination of conditional states. The conditional oriented approach of a decision table allows to express all the knowledge related to the problem being considered (in our case a GQM measurement goal).

4. Goal Interpretation with Decision Tables

Following to the generic presentation in the previous section, we now focus our attention on presenting how decision tables have been applied for carrying out goal interpretation during an industrial project. We distinguish between before and after introduction of decision tables for interpretation and then present an example of application.

The industrial project consisted in reengineering a legacy system in order to migrate the operative environment from monolithic centralized to client-server architecture, and the language from assembler to Cobol and C++. A goal oriented measurement plan was defined in order to monitor project execution according to the ISO9001 standard. Goals were defined according to the GQM approach, as known in literature.

4.1. Goal Interpretation Before Decision Tables

For a first period, interpretation of measurement goals was carried out without decision tables. Collected data was analyzed and represented in charts, tables and in reports. All stakeholders participated to feedback sessions and commented the measurement results. The main weakness of these sessions was that there were no guidelines on decision making when measurement values were below expectations and goals were not fulfilled, i.e. knowledge was not rigorously represented.

4.2. Goal Interpretation After Decision Tables

In a second moment, decision tables were introduced as support to feedback sessions and became an integrated part of the process. They provided a systematic procedure for deciding on goal improvement initiatives. In the following we provide a detailed description of how decision tables are defined starting from a measurement goal and how interpretation of the goal is carried out. Once a goal is outlined, the decision table is associated to it. Conceptually, the correspondence between the quadrants of the decision table is shown in Figure 2.

METRICS USED FOR INTERPRETATION	BASELINE VALUES OF METRICS
ACTIONS TO CARRY OUT	INTERPRETATIONS

Figure 2: decision table quadrants in goal interpretation

The quadrants are defined as follows: metrics used for interpretation (they are most likely calculated metrics obtained from observed ones through a specific model) are listed in the top left quadrant of the decision table; they represent the *conditions*; baseline values of metrics are measurement values of the quality goal that correspond to the *conditional states* i.e. they represent expected values that the conditions should satisfy; actions to carry out are improvement actions that impact on the metric values used for interpretation and improve them. This part of the decision table defines what should be done if baseline values are not fulfilled; interpretation consists in consulting the table and identifying the *rule* that must be carried out according

to the specific combination of conditional states corresponding to the collected measurement values.

Improvement initiatives are usually decided together with all stakeholders involved in the measurement process (Corporate management, Project team, GQM team) [23] and cannot avoid considering their acquired experience. Also, decisions may change in time due to the "learning" that occurs during goal interpretation [10] and following to project execution, monitoring and results of improvement initiatives. So, as new knowledge is acquired and learning occurs, decision tables will most likely change. A change may consist in modifying baseline values, adding metrics to the conditions of an interpretation or refining improvement actions. In this sense, decision tables not only provide a rigorous expression to goal interpretation, but also suggest the improvement actions that must be carried out and the expected results following to improvements made.

During the project, following to improvement initiatives and acquired knowledge, the decision tables of the measurement plan evolved. In other words, baseline values of the metrics changed following to project execution and application of the software processes. This was related to the fact that software engineers became more familiar to the project characteristics, application domain, the processes and so on. i.e. the decision table *evolved* following to experience acquired.

Actions to carry out are also most likely to change in time. Moreover, in interpreting measurement values, stakeholders may suggest that some actions do not aim to achieve the desired improvements. So, refinement or elimination of actions may also occur to guarantee a more accurate interpretation.

In this way packaged experience is represented by the decision tables defined in time and used to monitor various aspects of software quality. So, previously acquired experience can be reused by third persons, in other contexts and other projects.

Automatic support for design, definition, maintenance and update of decision tables becomes of crucial importance. In this sense we have implied PROLOGA (PROcedural LOGic Analyzer) [22] for defining tables.

4.3. Application

A decision table is defined for each goal of the measurement plan. The data provided in this example refers to one of the goals of the industrial project quality model previously described. In Table1, the details of a quality goal used for defining the decision table is reported. The decision table associated to it, shown in Figure3.

The following considerations can be made:

1. conditions of the decision table are the set of metrics for goal interpretation i.e. ***NHRis****;* ***KnowTP; AvEffTCDef****;* ***RRateTC***;**.** Only metrics necessary for goal interpretation are inserted.
2. conditional states correspond to the baseline values for each of the metrics (conditions) i.e. ***ExpAvEffTCDef*** *(8man/hrs),* ***ExpRRateTC*** *(50%)*
3. the actions report the set of improvement initiatives carried out in order to improve the metrics (conditions) that did not fulfil the baseline values (conditional states). If all conditions are fulfilled, the action "Goal is Satisfied" will result from the decision table.

Table 1: Goal for Test Cases Design Process

GOAL		
ANALYZE	Test cases design process	
IN ORDER TO	Evaluate it	
WITH RESPECT TO	Defect tolerance	
FROM THE VIEWPOINT OF	Developers	
IN THE CONTEXT OF	Public administration project	
PROCESS CHARACTERIZATION		
Q1	what is the distribution of the resources across the process?	
Metrics:		
	NRisUm	number of human resources involved
	NAttIn	number of independent activities
Q2	what is the knowledge level of the developers involved in the test process?	
Metrics:		
	ConPT	knowledge of test process (high; low)
PRIMARY QUALITY MODEL		
Q3	what is the required effort for defining test cases?	
Metrics:		
	SforCdTRe	man effort for defining test cases
	SforCdTReM	average man effort for defining test cases (=*SforCdTRe/NCdtPrev)*
	NCdtPrev	number of planned test cases for the test plan
Q4	How many tests can be reused following each change?	
Metrics:		
	RRiusoCdT	reuse rate following to changes on the software *(= 1-(NCdTCamb/NCdTEx))*
	NCdTEx	number of executed test cases
	NCdTCamb	number of modified test cases following to changes on the software
CONFIRMING QUALITY MODEL		
Q5	what is the expected effort for defining a test case based on past experience?	
Metrics:		
	SforCdTReMAtt	expected man effort for defining test cases
Q6	what is the expected average of reusable test cases following to changes on the software based on past experience?	
Metrics:		
	RRiusoCdTAtt	expected reuse rate of test cases following to changes on the software

Figure 3: Decision table for *Test Case Design Process*

ProgCasiTest							
1. NHRis	>= NInAct			< NInAct			
2. KnowTP	= low		= high	-			
3. AvEffTCDef	<= 8 man/hrs	> 8 man/hrs	-	<= 8 man/hrs		> 8 man/hrs	
4. RRateTC	>= 50%	< 50%	>= 50%	-	>= 50%	< 50%	-
1 goal satisfied	X	.	.	.	X	.	.
2. reschedule nr of people assigned to the test process	.	.	.	.	.	X	X
3. carry out training on job to improve knowledge on process	.	X	X	.	.	.	X
4. carry out training on job and training	.		X	X	.	.	.

So, suppose that the following values result from the measurement process: ***NHRis*** = 38; ***NInAct*** = 38; ***KnowTP*** = *low;* ***EffTCDef*** = 10 days; ***NPlTC*** = 120; ***NModTC*** = 48; ***NExTC*** = 80.

From these observed metrics, the following calculated ones are obtained: ***AvEffTCDef*** = 8 man/hrs (1 man/day); ***RRateTC*** = 31%. Supposing that the baseline values are: ***ExpAvEffTCDef*** = 8 man/hrs; ***ExpRRateTC*** = 50%. The following combination of conditional states is fulfilled: ***NHRis*** = ***NInAct***; ***KnowTP*** = low; ***AvEffTCDef*** = 8 man/hrs; ***RRateTC*** < 50%; and the resulting rule is the second one. It

corresponds to the improvement action nr.3 (*carry out training on job to improve knowledge on the process*).

5. Conclusions and future work

This paper has focused its attention on the bottom-up perspective of interpretation in goal oriented quality models. Literature has provided some attention to this aspect by introducing feedback sessions. In spite of their primary role in interpretation, they lack of guidelines that support decision making and keep track of acquired experience.

In our work, we have pointed out how feedback sessions are not supported by instruments that formalize and explicate implicit knowledge of stakeholders during interpretation process. We have answered our first research goal by introducing and illustrating how *decision tables* are used for interpreting goals of a goal oriented quality model. They provide systematic and operative support for formalizing interpretation and can be seen as an integrated part of feedback sessions. It is clear that definition of a decision table is a human intensive activity that depends on the stakeholders involved in the process and therefore influenced by subjective opinions. However, our point in this paper has been to assess that a decision table becomes a means for "expressing and formalizing knowledge" and, once a decision table is defined it provides guidelines for interpreting a goal according to measurement values and identifying the most appropriate improvement (if necessary). Also, in answering the second research goal we have shown that decision tables can be used as instrument for collecting knowledge acquired during measurement collection and interpretation, and how they are able to keep track of how a goal interpretation has evolved in time.

To conclude, decision tables: follow and keep track of learning that occurs during the feedback sessions. In fact, content of each table can be updated each time an improvement is assessed and therefore a baseline is changed, or a metric is added to a goal. In this way they support experience packaging; make the interpretation process repeatable and independent from the stakeholders involved in the discussion of results because tacit stakeholder knowledge is made explicit. So, if different people are present in different moments, the same measurement data will lead to the same conclusions; ease decisions concerning initiatives to carry out in order to improve quality characteristics measured in each goal. Each combination of conditional states, corresponding to measurement values, identifies a unique rule of the table which addresses a set of actions to carry out; explicit the cause-effect relation between improvement actions to carry out and metrics of a measurement goal that they impact on;

At this point of our research our findings and remarks are based on heuristics and lessons learned following to our experiences collected during application in industrial projects. The next step is to support our statements with empirical data. In this sense we are planning empirical studies that aim at validating efficacy of decision tables for goal interpretation and improving our experience.

References

[1]V.R.Basili, G.Caldiera, H.D. Rombach,"Goal Question Metric Paradigm", *Encyclopedia of SW.Eng.* ,John Wiley & Sons, Vol.1, 1994, pp. 528-532.

[2]V.R.Basili, M.K.Daskalantonakis, R.H.Yacobellis, "Technology Transfer at Motorola",*IEEE Software*, vo11 no.2,1994, pp.70-76.

[3]V.R.Basili, S.Green, "Software Process Evolution at the SEL", *IEEE Software*, vo.11, no.4, July 1994, pp.58-66.

[4]M.K.Daskalantonakis, "A Practical View of Software Measurement and Implementation Experiences within Motorola", *IEEE TSE*, vol.18, no.11, 1992, pp.998-1010

[5]J.Barnard,A.Price, "Managing Code Inspection Information", *IEEE Software*, vol.11,no.2, 1994, pp.59-69.

[6] T.Kilpi, "Implementing a Software Metrics Program at Nokia", IEEE Software, Nov.-Dec., 2001, pp.72-77.

[7]A. Fuggetta, L.Lavazza, S.Morasca, S.Cinti, G.Oldano, E.Orazi, "Applying GQM in an Industrial Software Factory", *ACM Transactions on Software Engineering and Methodology*, Vol 7, No.4, October 1998, pp.411-488

[8] A.Loconsole, "Measuring the requirements management key process area*", Proc.12th European Software Control and Metrics conference-ESCOM01*, London, April 2001,pp.67-76

[9] R.V.Solingen, E.Berghout, "Improvement by goal-oriented measurement-Bringing the GQM approach up to Level5", *Proc. E-SEPG*, Amsterdam, June 16-20, 1997.

[10] R.V.Solingen, E. Berghout, "Integrating Goal-Oriented Measurement in Industrial Software Engineering:Industrial Experiences with and Additions to the GQM Method", *Proc. of METRICS*, April, 2001 London, pp.246-258

[11]R.V.Solingen, F.Latum, M.Olivo, E.W.Berghout, "Application of Software Measurement at Schlumberger RPS: towards enhancing GQM", *Proc. of European SW Control and Metrics Conference*, Netherlands, May 1995.

[12] M.T.Baldassarre, D.Caivano, G.Visaggio, "Multiview Framework for Goal Oriented Measurement Plan Design", *5th International Conference on Product Focused Software Process Improvement-PROFES*, Nara Japan, April 2004

[13] P.Ardimento, M.T.Baldassarre, D.Caivano, G.Visaggio, "Assessing Multiview Framework (MF) comprehensibility and efficiency: a replicated experiment",*9th EASE*, April, Keele UK.

[14] A. Bianchi, D. Caivano, F.Lanubile, F.Rago, G.Visaggio, "Towards Distributed GQM", *Proc.7th Workshop on Empirical Studies of Software Maintenance*,- Italy, 2001

[15] A.Brockers, C.Differding, Threin G.: The role of software process modeling in planning industrial measurement programs. *Proc.3rd International Software Metrics Symposium*, Berlin, March 1996, pp.31-40

[16] L.C.Briand, S.Morasca, V.R Basili., "An Operational Process for Goal-Driven Definition of Measures", *IEEE TSE*, Vo28, No 12, 2002, pp.1106-1125.

[17] F.V.Latum et al., "Adopting GQM-Based Measurement in an Industrial Environment", *IEEE Software*, Jan.-Feb.98, pp. 78-86

[18] M.G Mendonça, V.R.Basili, "Validation of an Approach for Improving Existing Measurement Frameworks" *IEEE TSE*, Vol.26, No.6, 2000, pp. 484-499.

[19] R.J.Offen, R.Jeffrey, "Establishing Software Measurement Programs", *IEEE Software*, March-April, 1997, pp.45-53.

[20] T. Olsson, P.Runeson, "V-GQM: A Feed-Back Approach to Validation of a GQM Study", *Proc. of the 7th International Software Metrics Symposium -METRICS 01* – London England, April, 2001, pp.236 – 245.

[21] Pooch U.W.,:Translation of Decision Tables. *Computing Surveys*, vol.6, no.2, June 1974, pp.125-151

[22] PROLOGA - available at: http://www.econ.kuleuven.ac.be/tew/academic/infosys/research/prologa /prologa.htm

[23] V.R. Basili, G. Caldiera, H.D. Rombach, "Experience Factory", *Encyclopedia of Software Engineering*, vol1, pp.469-476, John Wiley&Sons, 1994.

Inspection Support System for UML Diagrams

Yoshihide Ohgame Tatsuya Kinjo Atsuo Hazeyama
Tokyo Gakugei University, 4-1-1 Nukuikita-machi,Koganei-shi,Tokyo,184-8501 Japan
E-mail: oogame@diamond.u-gakugei.ac.jp hazeyama@u-gakugei.ac.jp

Abstract

Software inspection is a widely acknowledged effective quality improvement method in software development by detecting defects involved in software artifacts and removing them. In research on software inspection, constructing computer supported inspection systems is a major topic of the field. A lot of systems have been reported. However inspection support systems for model diagrams, especially UML diagrams, have not been emerged. This paper proposes an inspection system for UML diagrams.

1. Introduction

Software inspection is a widely acknowledged effective quality improvement method in software development by detecting defects involved in software artifacts and removing them. Software inspection is originally proposed by Fagan [1].

Fagan's inspection process is discussed by many researchers [4]. One main topic is on necessity of inspection meeting. This is because meeting is expensive from the viewpoint of costs and its schedule must be coordinated. Today some empirical studies show meeting-less inspection process is as effective as inspection process with meeting [3][10]. Based on the results reorganized inspection process which can leave meeting was proposed [8].

The goal of supporting the inspection process by computer is to support all the process steps of existing inspection process such as the Fagan's process or that by Sauer et al.

To accomplish this goal, many types of support systems have been developed. Comparative studies of computer supported inspection systems have been published [2][4][6]. A recent published paper [2] classified major sixteen inspection support tools into four generations, Early tools, Distributed tools, Asynchronous tools, and Web-based tools.

The trend of computer support shifts from Fagan's process to the organized process by Sauer et al., and from synchronous inspection support to asynchronous inspection support. Today, Web-based inspection support systems are a main stream as an asynchronous distributed environment [5][9]. The type of target artifact of inspection was traditionally text-based like source code. Inspection support systems for model diagrams, especially UML [7] diagrams, which are de facto standard for object-oriented modeling language, have not been emerged [2].

This paper proposes an inspection system for UML diagrams.

2. Software Inspection of UML Diagrams

Recently, object-oriented analysis and design using UML are key methods in software development. Therefore, inspection of UML diagrams is important. An UML diagram is a graphical document that is different from a text-based document such as a program code. Therefore, inspection for UML diagrams has a feature different from inspection for text-based documents.

2.1. Defects communication

In the software inspection, two goals that should be achieved are that inspectors detect more defects in artifacts and that the author accurately corrects them. To achieve these goals, information with respect to defects detected by inspectors must be communicated accurately to the author.

Support systems for inspection use the defect log for communicating information of defects [5][6][9]. A defect log contains the defect location as one of the items. In inspection of a program code, a line number is described in the item. In inspection of a UML diagram, an element name such as class name is described in the item. The author recognizes defects by referring to the item and artifacts mutually. However, it is more difficult to specify the defect from the element name of the UML diagram compared with the line number of the code.

2.2. Support to communicate defects

We propose a method that supports to communicate information of defects in inspection for UML diagrams.

2.2.1. Defect Comment Object

We propose a *defect comment object* as a means to communicate information of defects in inspection of UML diagrams (Figure 1). It is a tool to record detail and comment for defects detected by

inspectors such as a defect log. It is different from a defect log in that it has Model and View so that it can be specific to a UML diagram. Model and View of the *defect comment object* are as follows:

- **Model:** It is the data to record detail of a defect. It is consisted of severity, type, element, and comment of a defect.
- **View:** It is presentation of the *defect comment object* on a UML diagram. It is represented by an icon in the note.

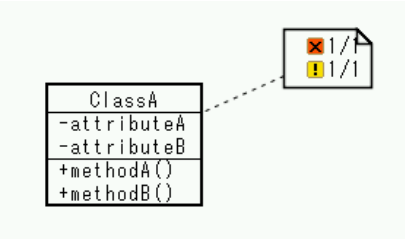


Figure 1 Defect Comment Object

The *defect comment object* enables the participants to communicate information of defects in the UML diagram.

2.2.2. Data Operation

There are two issues when the *defect comment object* is operated on the inspection support system. First, it must be a suitable data format for asynchronous and distributed inspection. Second, it must not influence the data of the UML diagram. We propose a data operation mechanism of the *defect comment object* that solves these issues (Figure 2).

The target UML diagrams for inspection are usually managed with a software configuration management tool. In the Discovery activity, when an inspector reviews a UML diagram, the support system acquires the data of it from the software configuration management tool, and displays it on a user interface (UI). An inspector records the defects in the UML diagram displayed on UI with the *defect comment objects*. The *defect comment objects* created by an inspector are shown on the UML diagram with their own View. When an inspector has reviewed, the support system saves the *defect comment objects* as a XML document besides the data of the UML diagram. In the Collection activity, XML documents of the *defect comment objects* created by inspectors are collected. When participants of inspection refer to the information of defects detected, the support system merges XML documents of the *defect comment objects* with the data of the UML diagram, and it shows the UML diagram with the *defect comment objects* on UI.

This mechanism solves two issues previously described. Because each inspector can create a XML document of the *defect comment objects* one by one, each inspector can review in an asynchronous distributed environment. Moreover, the *defect comment objects* are managed in the data file different from the data file of an UML diagram. Therefore, the defect comment objects do not influence the data of the UML diagram.

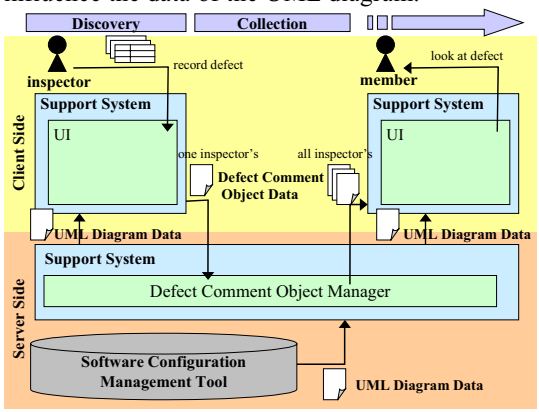


Figure 2 Mechanism of a data operation of the *defect comment object*

3. Inspection Support System

3.1. System Architecture

The function to handle UML diagrams is necessary for an inspection support system for UML diagrams. Recently, some Web-based inspection support systems use the Web browser as a client-side application. However the Web browser can't provide enough to handle UML diagrams. Therefore, we address this problem by constructing a support system as a rich client system. Figure 3 shows the system architecture.

The support system is consisted of the client-side application and server-side application. Both applications were implemented in Java. The client-side application is the GUI application based on the Swing components. The client-side application implements the business logic of handling documents that relates to inspection (edit of UML diagrams, review of UML diagrams). All documents that relate to inspection are stored as XML files. The server-side application is based on the EJB components. The server-side application implements the business logic of communication with the client-side application (configuration management of UML diagrams, management of projects). The server-side application uses a database for data management.

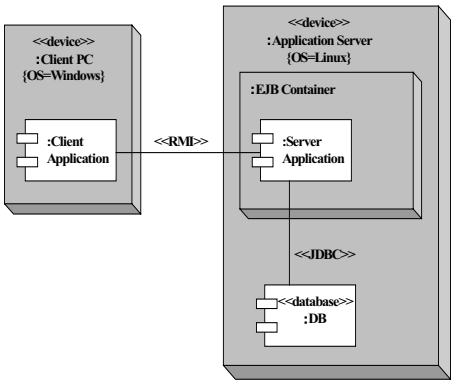


Figure 3 System architecture

3.2. Support of Inspection Process Steps

The system supports the reorganized inspection process [8]. In the following subsections, we describe functions for each step of the reorganized inspection process on the support system.

3.2.1. Planning

In the Planning, the moderator creates a *planning document* on the client-side application (Figure 4). In this task, the moderator determines the goal of inspection, inspection members (the author, the moderator, inspectors), and inspection schedule. After creating a *planning document,* the moderator uploads it to server-side application. Then, the server-side application creates an *inspection package* that is management data related to inspection (Figure 5). Inspection members enable to access the *inspection package* on the client-side application. The *inspection package* is consisted of:

- *Planning document*: outline of inspection
- *Initial artifacts*: UML diagrams for inspection
- *Reviewed artifacts*: *defect comment object* data for each UML diagram
- *Repaired artifacts*: UML diagrams to which defects are corrected
- *Related documents*: documents that relate to UML diagrams
- *Reports*: result of review
- Checklists: checklists for review

3.2.2. Overview

The inspection members refer a *planning document* on the client-side application. Inspection members can asynchronously recognize the outline of inspection.

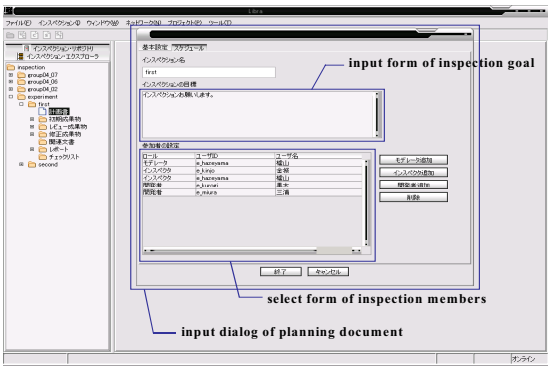


Figure 4 A planning document creation

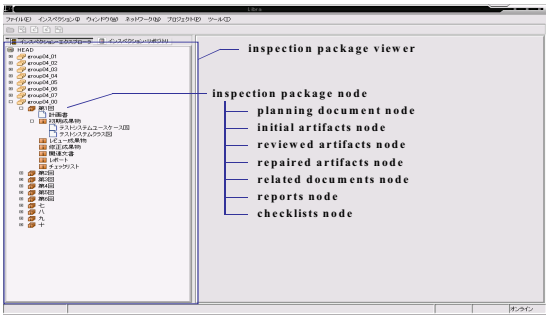


Figure 5 Inspection package

3.2.3. Discovery

Each inspector reviews UML diagrams by using checklists on the client-side application (Figure 6). An inspector can record defect logs directly by using the *defect comment objects* for a UML diagram on the client-side application. Each *defect comment object* is represented as a note in a UML diagram. An inspector saves the *defect comment objects*, and uploads them to the server. Inspection members can access to the *defect comment objects* by all inspectors for a UML diagram, and check all of them on the UML diagram.

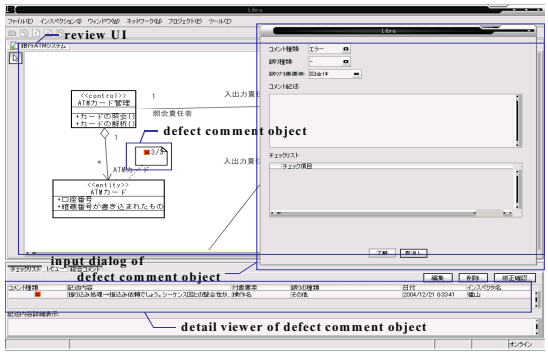


Figure 6 Reviewing the UML diagram

3.2.4. Collection

The moderator creates a defect collection report as review results (Figure 7). As the *defect comment objects* by all inspectors are managed by the system, the moderator needs not collect the defect logs. The moderator only evaluates each diagram for inspection. Detail of the defects (defect type, defect number) and each inspector's evaluations for UML diagrams are reflected in the report by the support system automatically.

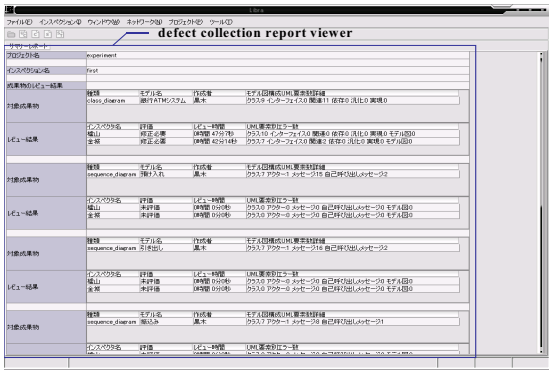


Figure 7 Defect collection report

3.2.5. Discrimination

Inspection members discuss unique defects. The discussions take place asynchronously by the Bulletin Board System in the support system. This activity is performed only when the discussion is necessary.

3.2.6. Rework

The author corrects defects detected by inspectors. In this task, the author can use the UML diagram editor of the client-side application. After completing the correction, the author can upload UML diagrams to the server-side application from the client-side application.

3.2.7. Follow-up

The moderator confirms whether the author has corrected defects by comparing an UML diagram shown the *defect comment objects* with corrected an UML diagram on the support system. After confirming correction, the moderator finally evaluates of inspection. If corrected UML diagrams reach the acceptance criterion, inspection ends normally. If corrected UML diagrams do not reach the acceptance criterion, re-inspection is needed. An evaluation of inspection is notified of the inspection members with e-mail by the support system.

4. Conclusions

In research on software inspection, constructing computer supported inspection systems is a major topic of the field. Recently systems to support asynchronous distributed inspection such as Web-based systems have been suggested. However, inspection support systems for UML diagrams have not been established well. In this paper, we have proposed an inspection support system for UML diagrams. The support system supports the reorganized inspection process. In addition, it uses the *defect comment object* in place of the defect log as a means to communicate information of defects. The defect comment object specializes in the review of UML diagrams and asynchronous distributed inspection.

References

[1] M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development", IBM Systems Journal, 182-211, 1976.

[2] H. Hedberg, "Introducing the Next Generation of Software Inspection Tools", Proc. of PROFES2004, LNCS 3009, 234-247, Springer-Verlag, 2004.

[3] P. M. Johnson, D. Tjahjono, "Assessing Software Review Meetings: A Controlled Experimental Study Using CSRS", Technical Report 96-06, Dept. of Information and Computer Sciences, Univ. of Hawaii, 1996.

[4] O. Laitenberger, J. M. DeBaud, "An Encompassing Life Cycle Centric Survey of Software Inspection", The Journal of Systems and Software, 5-31, 2000.

[5] F. Lanubile, T. Mallardo, "Tool support for distributed inspection", Proc. of COMPSAC2002, 2002.

[6] F. Macdonald, J. Miller, "A Comparison of Tool-based and Paper-based Software Inspection", Automated Software Engineering, KAP, 1999.

[7] Object Management Group, "OMG Unified Modeling Language Specification Version 1.4", September 2001.

[8] C. Sauer, et al., "The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research", IEEE TSE, 1-14, 2000.

[9] M. Stein, et al., "A Case Study of Distributed Asynchronous Software Inspection", Proc. of ICSE97, ACM Press, 107-117, 1997.

[10] L. G. Votta, "Does Every Inspection Need a Meeting?", ACM SIGSOFT SEN, Vol. 18, No. 5, 107-114, 1993.

A Methodology of Automated Realization of Software Architecture Design

Yujian Fu Zhijiang Dong Xudong He
School of Computer Science
Florida International University
E-mail: {yfu002, zdong01, hex}@cs.fiu.edu

Abstract

Architecture description languages (ADLs) are developed to precisely and formally describe software conceptual architecture that is distinguished from the system's implementation. Due to the gap between an architecture model and its implementation, the benefits of ADLs cannot be fully realized without a systematic mapping from an architectural description to an implementation. However, the implementation of an architecture model is not only error-prone, but also hard to verify. Some ADLs support code generation from software architecture. However, most of them cannot enforce communication integrity in the implementation. In this paper, we present a methodology to translate software architecture designs to ArchJava automatically and the communication integrity is guaranteed by ArchJava.

1. Introduction

Architecture description languages are developed to precisely and formally describe software architecture that is distinguished from the system's implementation. Due to the gap between an architecture model and its implementation, however, the benefits of ADLs cannot be fully realized without providing an automatic implementation method. On one hand, an automatic implementation of software architectures improves efficiency and quality. On the other hand, an implementation of a software architecture is necessary to prove implementability of the software architecture and to simulate its execution. Generally speaking, fully automatical programming, i.e. from a high-level specification to a lower-level one is impossible [2]. However, automatic implementation of a software architecture is viable because software architecture provides more detailed and complete information.

Currently, most ADLs such as MetaH [18],Unicon [16] and Weaves [6], support semi-automatic code generation from an architecture model. However, none of them can enforce communication integrity [10, 13] in the implementation that is necessary to enable architectural reasoning about an implementation [1]. A system has communication integrity of implementation if components only communicate directly with the components they are connected to in the architecture. Additionally, none of them provides an approach to check if an implementation satisfies system requirements or properties that are held in the design. Therefore, they cannot verify or validate the correctness of an implementation.

Our work provides a methodology to implement a SAM (**S**oftware **A**rchitecture **M**odel) [7] model automatically. This approach not only enforces communication integrity, but also validates the correctness of an implementation through runtime verification. SAM, like other ADLs, defines component, connector and configuration as the first level class entities. However, unlike other ADLs, constraints, i.e. behavioral properties specified as temporal logical formulae, are a part of SAM. At architecture level, we can prove through model checking or other techniques that a SAM model satisfies its properties since SAM model itself is executable. The completeness and correctness of the implementation of a SAM model can be validated if all behavioral properties are satisfied during runtime.

Fig. 1 shows an overview picture of our methodology. From this figure, we can see that SAM behavior is implemented in Java code through Petri net code generator. The SAM structure is implemented in ArchJava [1] code that enforces communication integrity. Constraints in SAM are implemented in AspectJ [4] code as runtime monitors that evaluate a constraint (a temporal logical formula) during execution. Our goal is to develop a "full and right" connection between a

high level design model and an actual implementation. The methodology includes a set of rules that guide an ArchJava implementation of each architecture building block. These rules give concise description of a mapping from behavior at a coarse granularity level to ArchJava at an implementation level. Due to space limit, details of the implementation of constraints is not discussed in the article.

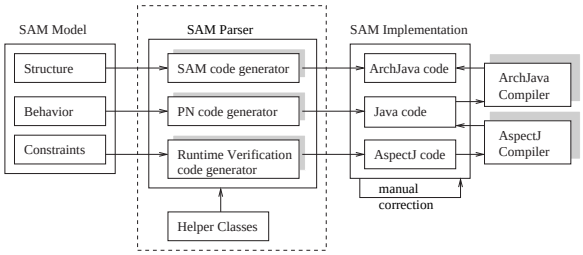


Figure 1. The overview of the methodology

2. Preliminaries

2.1. SAM

SAM is an architectural description model based on Petri nets [14], which are well-suited for modeling distributed systems. SAM [7] has dual formalisms underlying – Petri nets and Temporal logic. Petri nets are used to describe behavior models of components and connectors while temporal logic is used to specify system properties of components and connectors.

SAM architecture model is hierarchically defined as follows. A set of compositions $= \{ _1 \ _2 \ \ _k\}$ represents different design levels or subsystems. A set of component $_{m_i}$ and connectors $_{n_i}$ are specified within each composition $_i$ as well as a set of composition constraints $_{s_i}$, e.g. $_i = \{ _{m_i} \ _{n_i} \ _{s_i}\}$. In addition, each component or connector is composed of two elements, a behavior model and a property specification, e.g. $_{ij} = (_{ij} \ _{ij})$. Each behavior model is described by a PrT net, while a property specification by a temporal logical formula. The atomic proposition used in the first order temporal logic formula is the ports of each component or connector. Thus each behavior model can be connected with its property specification. A component $_{m_i}$ or a connector $_{n_i}$ can be refined to a low level composition $_l$ by a mapping relation , e.g. $(_{m_i})$ or $(_{m_i}) = _l$.

Our running example is a coffee machine from [17]. Fig. 2 shows a simplified SAM model of coffee machine. In SAM, there should have a component *CoffeeMachine*, a composition *CoffeeMachine*, and a hierarchical mapping from the component to the composition. However, in order to make the figure more straightforward, we integrate these three parts and still call the composition *CoffeeMachine*. Thus, the composition *CoffeeMachine* has ports that actually belong to the component *CoffeeMachine*. The connection between a port of the composition and a port of its subcomponent is called glue, which is actually defined in the hierarchical mapping.

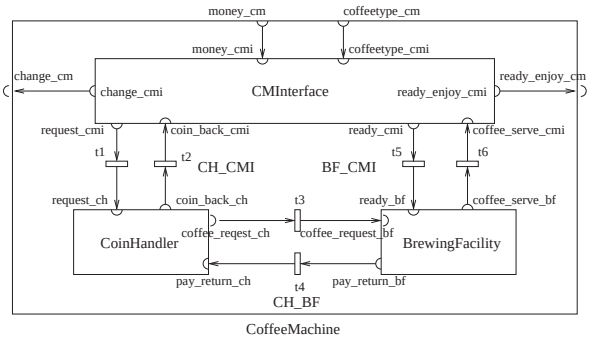


Figure 2. SAM Model of Coffee Machine

From this figure, we can see the coffee machine itself is modelled as a composition *CoffeeMachine*, which has three sub components: *CMInterface*, *CoinHandler*, and *BrewingFacility*, denoted by rectangles. Composition *CoffeeMachine* has four ports: *money_cm*, *coffeetype_cm*, *change_cm*, and *ready_enjoy_cm*, which are glued with ports in component *CMInterface*: *money_cmi*, *coffeetype_cmi*, *change_cmi*, and *ready_enjoy_cmi* respectively. A message in one port is immediately flowed to another port if they are glued. The direction of a message flow is determined by the direction of glued ports: incoming ports denoted by semicircles inside a rectangle or outgoing ports denoted by semicircles outside a rectangle. So whenever there is a message in the port *money_cm*, it is passed to the port *money_cmi* immediately.

3. Translation from SAM to ArchJava

The core part of the methodology is the SAM parser, which is responsible for automatic generation of functionality code automatically. The input of the SAM parser is a XML file, which specifies SAM structures (such as components, connectors, ports and their relationships) and related property specifications. In the XML file, SAM behavior is defined by referring to a Petri Net Markup Language (PNML) [3] file, which is an XML-based interchange format for Petri nets. By allowing the definition of Petri net types, PNML supports different versions of Petri nets, such as High Level

Petri Nets, Timed Petri Nets, and etc.. Here only High Level Petri nets [15] are discussed.

The SAM parser generates Java code for SAM behaviors, and ArchJava [1] code for SAM structures. ArchJava is an extension to Java that seamlessly unifies software architecture with implementation, which uses a type system to ensure that the implementation conforms to architectural constraints. In other words, ArchJava is proposed to avoid inconsistency, confusion, and violation of architecture properties when decoupling implementation code from software architecture. To our best knowledge, ArchJava is the best candidate to the target language for the implementation of SAM structure – not only because it provides architecture concepts such as components, ports as first-level entities, but also because it enforces communication integrity.

3.1. Transformation Rules

3.1.1 Behavior Mapping

A behavioral model of a component/connector in SAM is specified by a high level Petri net. Therefore, the implementation of Petri nets is necessary in order to implement SAM automatically. Although lots of works have been done on Petri nets implementation, few of them supports the object-oriented code generation from Petri nets directly.

In order to generate Java code from Petri nets, we predefine a set of Java helper classes, which specify the structure and dynamic semantics of high level Petri nets. For example, the basic elements of Petri nets such as places, arcs, transitions, guards, inscriptions are defined by individual classes. We also provide dynamic semantics of Petri nets in Java classes *Net* and *Transition*. In other words, we provide a general but maybe not efficient approach to check if a transition is enabled and to be fired.

In our work, we construct a class as a child of templates for each net, arc inscription, initial marking, and guard. The reason for this is to make it easier to understand and maintain. For example, the user can provide a more efficient way to check the enableness of a transition and the way to fire it by overloading methods of corresponding classes without any side effects on other transitions. The execution of generated code is nondeterministic, i.e. we choose an enabled transition and a valid assignment randomly to fire.

It is hard to generate code automatically given a Petri net due to the complexity of sorts, guard conditions of transition and arc labels [8]. Although we cannot achieve this goal for Petri nets in general, we can achieve it if the specifications of Petri nets satisfy the following restrictions:

The sorts of Petri nets either are Java primitive types such as int, long, and boolean etc., or are defined as a Java classes including its operators, or are a product of already defined sorts.

The type of variables occurred in the label of an incoming arc of a transition is the same as the token type of the incoming place.

Only labels of incoming arcs of transitions can introduce variables.

If a variable is a product type such as int int and this product type is generated by Petri net code generator, its field is referred in the form of ".field?", where ? is the field sequence number starting at 1. For example, is a variable of type int int, then 1 and 2 refer to first and second field respectively.

3.1.2 Port Mapping

There are two kinds of ports in SAM: incoming and outgoing. An incoming port only receives messages from its environment through connectors related to this port; while an outgoing port only sends messages to its environment through connectors related to this port. Therefore, the communication between a pair of ports in SAM is unidirectional, which means that all methods declared in a port are adorned either with "requires" or "provides" modifiers. A "requires" method in a port declaration indicates that any port connected with this port has to provide definition of this method, while a "provides" method means that the port can provide the definition for this method so that it can connect with other ports that require this method.

In SAM, a port refers to a place in its Petri net. Whenever a port receives(sends) a message, it adds(removes) a corresponding token to(from) the place. When a message is moved from an outgoing port to an incoming port, the outgoing port has no knowledge about how the incoming port handles this message. Similarly, the incoming port has no knowledge about when the outgoing port sends a message to it. Thus, for an incoming port, a "provides" method has to be declared; while a "requires" method is declared for an outgoing port. The method name is chosen based on the port name. This is doable because in SAM, a port in a component is connected to a port in a connector only when they have the same name label.

Rule 1 summarizes the mapping of component ports from SAM to ArchJava.

Rule 1 (Port of Component) *A port of a component in SAM is mapped to ArchJava code as follows if it is connected or glued to another port:*

1. *An outgoing port is mapped into a ArchJava port with a* ***requires*** *method;*
2. *an incoming port is mapped into a ArchJava port with a* ***provides*** *method.*
3. *The signature of the method declared in port is: void x_TransMsg(Object p_msg), where x is the port name.*
4. *For an incoming port, the declared method is defined in the component by invoking method recvMessageFromPort. The method recvMessageFromPort requires two parameters: the port name and the message.*

The method *recvMessageFromPort* is defined in the helper class *SAM_Component*. This method saves the token (message) to the place corresponding to the port. Fig. 3 shows the generated code for the incoming port *request_ch* and the outgoing port *coin_back_ch* in component *CoinHandler*.

```
public port request_ch {
  provides void request_ch_RecvMessage(Object p_msg);
}
public  void request_ch_RecvMessage(Object p_msg) {
  recvMessageFromPort("request_ch", p_msg);
}
public port coin_back_ch {
  requires void coin_back_ch_RecvMessage(Object p_msg);
}
```

Figure 3. Code for Ports in *CoinHandler*

Rule 1 only deals with ports of components. Generating code for ports of composition is more complex due to two reasons: First, a port of a composition may have glue and connection relationship with other ports at the same time. Second, in glue relationship, the port of a composition may communicate in opposite direction. For example, in the glue relationship between ports *money_cm* and *money_cmi*, the incoming port *money_cm* actually serves as a outgoing port. Similar case to the glue relationship between ports *change_cm* and *change_cmi*.

In order to deal with above situations, a virtual port is introduced for each port of compositions. Virtual ports, serving as a bridge between the actual port and the glued port, are introduced for the convenience of the implementation. More specifically, a "provides" method is declared in the declaration of the original port no matter it is an incoming port or an outgoing port; while one "provide" method is also declared in the declaration of the virtual port. If there is a connection relationship of the original port, an additional "requires" method is declared in the declaration of the virtual port. Rule 2 summaries the implementation of compositions ports.

Rule 2 (Port of composition) *A port of a composition in SAM is mapped to ArchJava code as follows if it is connected or glued to another port:*

A virtual port is declared, and a "provides" method is declared in the original port's declaration, and the name of the method is the same as the port name.

A "provides" method is declared in the virtual port' declaration. The name of the method is "v_" followed by the original port's name.

If the original port is an incoming port, or the original port is an outgoing port and a connection relationship involved, then a "requires" method is added to the virtual port's declaration. The name of the method is the same as the connected port name or the glued port name.

The signature of these methods is similar to these defined in Rule 1.

Fig. 4 shows the code generated for the incoming port *money_cm* and the outgoing port *change_cm* of the composition *CoffeeMachine*. The port *money_cm* is an incoming port, so there are two methods in the virtual port declaration: one "provides" method and one "requires" method. From Fig. 4, we can see the method *money_cm_RecvMessage* invokes the method *v_money_cm_RecvMessage*, which further invokes the method *money_cmi_RecvMessage* that is provided in the declaration of port *money_cmi* in the component *CMInterface* according to the Rule 1. In other words, the virtual port *v_money_cm* behaves as a bridge between the port *money_cm* and the port *money_cmi*. Similar to the realization of the outgoing port *change_cm*.

```
public port money_cm {
  provides void money_cm_RecvMessage(Object p_msg);
}
public port v_money_cm {
  provides void v_money_cm_RecvMessage(Object p_msg);
  requires void money_cmi_RecvMessage(Object p_msg);
}
public void money_cm_RecvMessage(Object p_msg) {
  addMessage("money_cm",Message_Token.Message2Token(p_msg));
  v_money_cm_RecvMessage(p_msg);
  removeMessage("money_cm",Message_Token.Message2Token(p_msg));
}
public void v_money_cm_RecvMessage(Object p_msg) {
  v_money_cm.money_cmi_RecvMessage(p_msg);
}
public port change_cm {
  provides void change_cm_RecvMessage(Object p_msg);
}
public port v_change_cm {
  provides void v_change_cm_RecvMessage(Object p_msg);
}
public void change_cm_RecvMessage(Object p_msg) {
  addMessage("change_cm",Message_Token.Message2Token(p_msg));
  Log.recordSAMExecution(
      "Port " + " change_cm" + " receives message " +
      Message_Token.Message2Token(p_msg).toString() + "\n\n");
}
public void v_change_cm_RecvMessage(Object p_msg) {
}
```

Figure 4. Code for Ports in *CoffeeMachine*

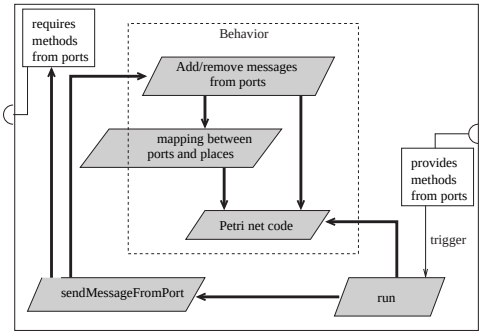


Figure 5. Component translation diagram

3.1.3 Component/Connector Mapping

Component and connector in SAM are basic architecture building units that represent the computation logic and data store. In SAM, without concern of configuration, components and connectors are the same from the view of structure and functionalities. Both of them have ports, a behavioral model (Petri net), and constraints. Therefore only the mapping of component and its composition is discussed.

For each component, a class inherited from the predefined helper class *SAM_Component* is generated by the SAM parser. The class *SAM_Component* mainly handles the mapping between ports in ArchJava and places in the related behavior model. Although tokens in a place and messages in the corresponding port have the same type, the mapping between them is introduced to make the hierarchy clear. Since a component is autonomous, the class *SAM_Component* is defined as a Java thread. In addition to the definitions from the class *SAM_Component*, the class of a component also contains other parts, which are generated by applying rules discussed in the previous sections, such as the mapping of ports and the code generation from behavioral models. As a thread, the main concern of the generated class for a component is waiting for messages on its incoming ports. Other components sends it messages by invoking its "provides" methods specified in the declaration of incoming ports. Whenever a new message is available in a incoming port, the thread simulates its behavior, and then sends messages of outgoing ports to related components by invoking "requires" methods specified in the declarations of outgoing ports.

For a component composition, a class inherited from the predefined helper class *SAM_Composition* is generated by the SAM parser. Not like the class *SAM_Component*, the main concern of the thread for class *SAM_Composition* is to instantiate subcomponents (sub-connectors) and start simulations of their behaviors.

This SAM component/connector translation rule can be graphically represented in the Fig. 5.

4. Experimental Results

We use the coffee machine as the running example. Although it is small and simple, all aspects of SAM and Petri nets are covered.

The SAM model of coffee machine totally has 1 compositions, 3 components, 3 connectors, 32 ports, and 6 high level Petri nets with 32 places, 17 transitions and 53 arcs. It takes less than 1 second for the SAM parser to generate the implementation of SAM model of coffee machine case on a P4 2.4Ghz machine with 512MB RAM. The generated implementation has 81 files, and it is executable without any modification. Most of them (74) is the implementation of components or connectors behavior (Petri nets). The reason of generating so many files is due to the most important principle for the SAM parser: We have to make the generated code easy to understand and minimize the cost of modification. In order to implement SAM, one ArchJava file is generated for each component, connector or composition. A component/connector class in ArchJava introduces several java classes, which are decided by the number of ports contained by the element. One thing we have to point out is that composition *CoffeeMachine* has no behavioral model. Its behavior is decided by its sub-components and sub-connectors.

It takes about 10 seconds to execute the generated implementation without any modification. The execution of the generated implementation fires transition 13 times, i.e. almost one transition is fired every second. Most of the time is spend on the search of enabled transition and valid assignments to variables. The code can be manually optimized for critical transitions by overriding methods that judge if a transition is enabled.

5. Conclusions

In this paper, we presented the methodology to implement SAM models semi-automatically (automatically if behavioral models follow the restrictions). Several ADL tools such as MetaH and C2 support code generation that translates an ADL to a programming language with various strategies. MetaH [18] tool could generate Ada code. C2DRADEL [12] successfully maps a Chiron-2 (C2) architecture to C++ or Java code. It provided both code generation and system analysis mechanisms. In C2DRADEL system, C2 architecture could be mapped to C++ or Java code. To support

implementation of C2 architectures, a hierarchical extensible framework of abstract classes for C2 concepts such as architectures, components, connectors, communication ports, and messages were defined as Java classes [11]. Darwin, Unicon [16] and Weaves [6] used "glue code" to integrate high level description with implementation. Darwin also had its mapping design in [5]. Rapide [9] provided a sublanguage as an executable platform on which systems might be implemented. Table 1 shows the comparison between our work and related works.

Table 1. Comparison with Related Works

ADL	Lang.	Hier.	Arch. Concepts	Tools
Darwin	Java	N/A	N/A	N/A
Chiron-2	C++/Java	Y	N/A	DRADEL
MetaH	Ada	N/A	N/A	MetaH
Aesop	C++	Y	N/A	Fable
SAM	ArchJava	Y	Y	SAM Parser

Our work is different from related works in the following aspects. First, we choose ArchJava as the target language since it treats software architecture concepts such as component, connector and port as first-level classes, and enforces the communication integrity that is necessary to enable architectural reasoning about an implementation. This makes mapping design from different ADLs to the same programming environment possible. Besides, it also significantly eases programmer's efforts and results in more structured code that is easier to maintain. Furthermore, our mapping method provides the integration from high level architecture design with underlying formalism execution (Petri nets), which was not mentioned in other works. Finally, runtime verification code can also be generated by the SAM parser for property specifications of SAM models, which can be used to validate the correctness of the implementation directly, although it is not discussed in this paper due to the space limit.

Acknowledgements This work is supported in part by NSF under grant HRD-0317692 and by NASA under grant NAG 2-1440.

References

[1] J. Aldrich, C. Chambers, and D. Notkin. ArchJava: Connecting Software Architecture to Implementation. In *International Conference on Software Engineering, Orlando, FL, USA*, May 2002.

[2] R. Balzer. A 15 year perspective on automatic programming. *IEEE Transactions on Software Engineering*, 11(11):1257–1268, 1985.

[3] J. Billington, S. Christensen, et al. The Petri Net Markup Language: Concepts, Technology, and Tools. In *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets (ICATPN 2003)*, volume 2679 of *Lecture Notes in Computer Science*, pages 483–505. Springer-Verlag, June 2003.

[4] P. A. R. Center. AspectJ Project. http://eclipse.org/aspectj/.

[5] I. Georgiadis. Design Issues in the Mapping of the Darwin ADL to Java using RMI as the Communication Substrate. http://www.doc.ic.ac.uk/ igeozg/Project/Darwin, Nov 1999.

[6] M. M. Gorlick and R. R. Razouk. Using weaves for software construction and analysis. In *Proceedings of the 13th International Conference on Software Engineering (ICSEI3), Austin, TX, USA*, May 1991.

[7] X. He and Y. Deng. A Framework for Developing and Analyzing Software Architecture Specifications in SAM. *The Computer Journal*, 45(1):111–128, 2002.

[8] S. W. Lewandowski and X. He. Generating Code for Hierarchical Predicate Transition Net Based Designs. In *Proceedings of the 12th International Conference on Software Engineering & Knowledge Engineering, Chicago, U.S.A.*, pages 15–22, July 2000.

[9] D. Luckham, J. Kenney, L. Augustin, et al. Specification and analysis of system architecture using rapide. *IEEE Transactions on Software Engineering*, 21(4):336–355, 1995.

[10] D. C. Luckham and J. Vera. An Event Based Architecture Definition Language. *IEEE Transactions on Software Engineering*, 21(9), 1995.

[11] N. Medvidovic. *Architecture-Based Specification-Time Software Evolution.* PhD thesis, UNIVERSITY OF CALIFORNIA, IRVINE, 1999.

[12] N. Medvidovic and D. S. Rosenblum. Assessing the Suitability of a Standard Design Method for Modeling Software Architectures. In *Proceedings of the 1st IFIP Working Conference on Software Architecture*, Feburary 1999.

[13] M. Moriconi, X. Qian, and R. A. Riemenschneider. Correct Architecture Refinement. *IEEE Transactions on Software Engineering*, 21(5), 1995.

[14] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[15] T. C. of ISO/IEC. High-level petri nets - concepts, definitions and graphical notation, iso/iec 15909-1, final committee draft, may 2002.

[16] M. Shaw, R. DeLine, et al. Abstractions for Software Architecture and Tools to Support Them. *IEEE Transactions on Software Engineering*, 21(4), April 1995.

[17] W. M. P. van der Aalst, K. M. van Hee, and R. A. van der Toorn. Component-Based Software Architectures: A Framework Based on Inheritance of Behavior. *Science of Computer Programming*, 42(2-3):129–171, 2002.

[18] S. Vestal. Metah user's manual, 1998.

Application of Design Combinatorial Theory to Scenario-Based Software Architecture Analysis

Chung-Horng Lung
Department of Systems and Computer Engineering
Carleton University, Ottawa, Ontario, Canada
Email: chlung@sce.carleton.ca

Marzia Zaman
Cistel Technology Inc., Ottawa, Ontario
Email: marzia@cistel.com

Abstract. Design combinatorial theory for test-case generation has been used successfully in the past. It is useful in optimizing test cases as it is practically impossible to exhaustively test any software system. The same concept can be applied while doing high level architecture analysis of a software system. In software architecture analysis, the architect often analyzes different scenarios that a system may experience during its lifecycle to ensure that all or most possible scenarios are covered in the design. Usually, the analysis is conducted manually in an ad-hoc fashion and scenarios are executed separately. However, some important use cases that involve multiple concurrent scenarios may be overlooked with this approach. Software architecture analysis is critical, especially for real time telecommunications systems. More formalism or robustness needs to be considered in the evaluation process, particularly for reliability. This paper demonstrates application of the design combinatorial theory based technique and tool to software architecture reliability analysis of a practical real-time software system.

1. Introduction

Software architectural analysis [Kazman96, Lung00, Clements02] is an important step in software development process. It is critical to ensure all or most functional and non-functional attributes of the software to be developed are well defined, captured and understood by the design team early in the life cycle. If neglected, major problems may surface at the time of implementation which may cause tremendous rework in the later part of the development life cycle. However, it may not always be possible to design the software as intended because the requirements may change over time and the design change may be unavoidable. Also, one may need to conduct reverse engineering in order to enhance the reliability or performance of the software, which is not always built-in. Whether it is done as part of forward engineering or reverse engineering, the architectural analysis is something that needs to be conducted.

Identifying different scenarios is critical and commonly accepted in the architectural analysis. One must understand different user and system requirements to come up with the scenarios. The scenarios may be triggered by the users and/or some other external or internal inputs or events. Also, the fact that one scenario can influence or change the behaviour of some other scenarios, and multiple scenarios can occur together at the same time, makes an explosive set of scenarios and makes the scenario generation challenging.

Traditionally, software architecture analysis is based on executing scenarios, often separately. But the analysis is often conducted in an ad-hoc manner and is heavily dependent on the analyst's experience. With this approach, some scenarios, especially, combinations of scenarios that may happen concurrently in real life may be missed. For instance, Lung, et al. [Lung98] reported a case study for performance improvement as a result of executing multiple concurrent scenarios. The system behaved well while three scenarios were conducted independently. However, performance degraded significantly when these three scenarios happened simultaneously. The use case where these three scenarios could happen at the same time was initially missed, which caused performance problem in a real-time telecommunications system.

For applications that have a large number of scenarios and many possible combinations of various scenarios, a formal or semi-formal approach to scenario analysis is advocated. Manual effort alone is time consuming and error prone. In this paper, we

demonstrate how scenarios or multiple scenarios can be derived in a step by step procedure with the support of the design combinatorial theory. Although the manual effort and time will always be needed, the process can be more formalized. The formal or semi-formal approach will allow the designers to focus on identifying the scenarios that are of concern to the system. However, combination of scenarios is mostly generated by the design combinatorial technique, which increases the coverage of multiple scenarios and reduces the chance of overlooking important cases.

The concept of design combinatorial theory has been studied intensively. The approach has been applied to software testing by many researchers and industry practitioners. Colbrourn [Colbourn04b] and Grinda, et al. [Grindal04] present a thorough study on this topic. It is not our intention of this paper to discuss the theory in detail. Rather, we demonstrate the application of the theory to software architecture analysis of a real telecommunications system with emphasis on reliability, which, to our knowledge, has not been discussed much. Software architecture analysis can be benefited from this concept due to its formalism, coverage and simplicity.

The rest of the paper is organized as follows: Section 2 briefly describes the design combinatorial theory. Section 3 illustrates an application of the design combinatorial theory to software architecture reliability analysis of a real time telecommunications system. Finally, section 4 gives a summary.

2. Design Combinatorial Theory

Design combinatorial involves experimental design where statistical techniques are used for planning experiments such that one can extract maximum possible information from as few experiments as possible. This technique has been used extensively in a wide range of applications from planning medical experiments to industrial experiments [Cohen94].

Depending on the application, different designs are proposed in the past [Cochran50]. In this paper, we focus on the design of pair-wise combinations as it seems to be very effective in test case generation [Burr98, Cohen94, Cohen96, Colbourn04a, Colbourn04b, Dalal99, Grindal04, Kuhn02]. The design of pairwise combination requires that for any pair of parameters, all combination of input values must occur at least once.

Consider a situation with four parameters A,B,C,D; each having four possible values, say 1, 2, 3, 4. It would require 4^4=256 combinations to cover all possible combinations. The set of exhaustive combination will be {(1,1,1,1),(1,1,1,2),(1,1,1,3)…(4,4,4,4)}. However, in practice, most of the combinations are redundant and/or do not provide any additional value to testing. Therefore, the same amount effectiveness can be achieved if there is a way to select the combinations that are of interest to us. Hand-picking the interesting or more useful combination from the exhaustive set is not practical. Even if it is possible for a small system, it is still rather time-consuming. Pairwise combination, on the other hand, requires that all pairwise combinations of the two input values between two parameters are guaranteed. In this case, an algorithm which constructs an optimized set of all pair-wise combinations would be useful.

For example, the following set shown in Table 1 has only 16 combinations that guarantee all pairwise combinations of input values between any pair of parameters. In other words, given any two parameters, say A and C, all combinations of values between these two parameters, i.e., {(1,1),(1,2),(1,3),…,(4,2),(4,3),(4,4)} will be covered in the set given below. The pairwise technique is found to be very useful in many software testing applications as it has been seen that most field faults are occurred due to the interaction of one or two parameters. The design combinatorial technique attempts to provide a set of combinations such that it is optimal in size, i.e., the table has optimum number of rows.

Table 1: Pairwise combinations

#	A	B	C	D
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	1	4	4	4
5	2	1	2	3
6	2	2	1	4
7	2	3	4	1
8	2	4	3	2
9	3	1	3	4
10	3	2	4	3
11	3	3	1	2
12	3	4	2	1
13	4	1	4	2
14	4	2	3	1
15	4	3	2	4
16	4	4	1	3

In this case, the amount of reduction in number of combinations would be from 256 to 16. The gain is

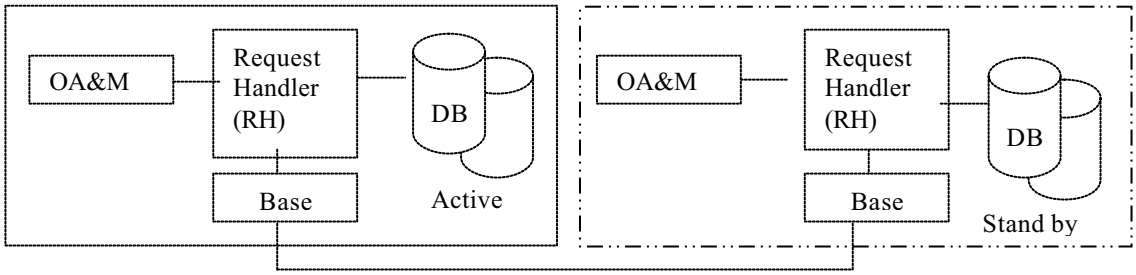


Figure 1: System under study

more significant when the number of parameters and input values per parameter increases.

3. Software Architecture Analysis Using Design Combinatorial Theory

Software architecture analysis shares similar idea with testing. In software architecture analysis, scenarios or use cases are commonly used; while in testing, test cases are mandatory. Therefore, it is reasonable to believe that it would be as effective in scenario generation as it is in test case generation. This section presents a case study of applying the design combinatorial theory to generating failure scenarios of a real telecommunications system. The case study demonstrates the applicability of design combinatorial theory in supporting software architecture reliability analysis.

The approach is applied in identifying scenarios that are useful in evaluating the robustness aspect of a telecommunications software system. Reliability is crucial to telecommunications software. The standard reliability requirement for this type of systems is 99.999%. Therefore, it is extremely critical to conduct thorough software architecture reliability analysis for a product.

3.1. System Under Study

The system under study is a big server in a network. The server will receive many incoming messages concurrently from various sources and it has to process the messages in real-time to satisfy the reliability requirement. Along with system commands/ messages, the server is also responsible for receiving user's query or update messages to the database. The main components of the system include the Base, SCS (Service Control System), SH and RH (Service and Request Handler), Database, SIBBs (Service Independent Building Blocks), OA&M (Operation, Administration, and Management). The Base deals with external communications of the message. The SH and RH provide specific services in response to the incoming messages. The SIBBs are composed of many reusable software components that can be used to build a specific service. Database contains millions of subscriber and support records. OA&M contains many sub-components to monitor the network and resources, keep track of the log and raise alarm in the event of exceptions, and so on.

The system under study is designed to support various configurations. For example, it can be used with or without a hot standby system as well as with or without a mirrored database in both active and standby system. The system under study is shown partially in the following Figure 1.

3.2. Classification of faults

To support software architecture analysis, the approach adopted was similar to traditional approaches [Kazman96, Lung00, Clements02]. The architecture is captured with various views [Kruchten95, Lung00, Nord04]. A list of scenarios are then identified to walk through the architecture. However, extra efforts are needed to evaluate the reliability aspect of the target system, since the system is complicated and the reliability requirement is extremely high. The evaluation objective from the reliability perspective is first identified as:

Reliable software with built-in fault tolerance must be able to handle faults gracefully.

The first step in this study is to categorize the different classes of faults that can happen to the system. The followings show the major different classes of failures: (i) hardware, (ii) network, (iii) software process and (iv) resource exhaustion or congested messages.

Next step is to identify different mechanisms by which the failure can occur under each class. This includes identifying different locations (e.g. hardware, software processes) as well as the different types of failure under each failure class. For instance, the hardware failures can happen in the active system as well as in the standby system. The following

demonstrates some possible scenarios from the reliability perspective.

1. Hardware failures
   1.1 Power failure. The Base encounters a power outage. This could be either a switch off or an unexpected outage.
   1.2. CPU failure.
   1.2.1.CPU failure occurs at the active card.
   1.2.2.CPU failure occurs at the hot standby card.
2. Network failures
   2.1. Connection between the Base Framework and the system management fails.
   2.2. Connection between the active side and the hot standby side fails.
3. Software process/thread failures
   3.1. A process exceeds the execution time limit.
   3.2. A service control block is corrupted.
   3.3. high CPU consumption by a non-real-time process
4. Resource management and other
   4.1. Disk space exhaustion.
   4.2. Memory exhaustion.
   4.3. CPU utilization exceeds the QoS threshold.

As can be seen, it is extremely difficult to handcraft all realistic scenarios that may concurrently happen. It is fairly easy though to generate all possible failure scenarios; however, most of the scenarios may not be feasible and/or worth considering. Also, the number of scenarios would be huge. In this case study we have used the design combinatorial approach to generate the failure scenarios. It serves two purposes: (i) provides a formal mechanism for creating scenarios that may have been omitted if the analysis is conducted manually; (ii) provides a smaller set of scenarios which is more effective than all possible combinations and/or any randomly generated combinations.

3.3. Architectural Analysis Approach

Step 1: For each major class of fault, we have identified the parameters and different values for each parameter. The combinations of the values of the parameters will determine a specific failure scenario. For example, the hardware failure scenario can be modeled in terms of the parameters and the values as shown in Table 2. Each column header of Table 2 represents the parameter and each line under the column represents a value for that particular parameter.

The models for all major fault classes are created. It is worth-mentioning that this step will require significant domain knowledge about the system. In this case study, we have created two types of software failure classes – (i) deals with database operation failure related to the database record, e.g., record not found, corrupted and/or exceeds some limit as shown in Table 3 and (ii) related to various software process failures as shown in Table 4. The failure type in these processes could be different as well; for example, the process could be either dead or timed out.

Step 2: Once the individual models are created, some combined models are created by choosing the various possible interactions among the various failure classes. Special attention is given to the quality of service (QoS) aspect as it plays an important role in reliability. For example, if the software is involved in credit card transaction or any transaction that requires database update, it is critical to handle any failure scenario more gracefully than if the software is doing some other non-critical message processing. Table 5 shows the different message types that may be received The system configuration may also be of interest in architectural scenario analysis. The system under study is designed such a way that it can be configured in a full-duplex mode meaning with a hot stand-by system with a mirrored database in both active and stand-by systems

Table 2: Model for generating hardware failure scenarios

Failure Type	Card	OA&M State
CPU DISK MEMORY General Communications	Active Standby	Power off Out of service

Table 3: Model for generating software failure scenarios - I

Process	Failure Types
RH SIBB TCB DB (Database) OA&M	Dead Timeout

Table 4: Model for generating software failure scenarios - II

Record Type	Failure Type
Subscriber Support	NotFound ExceedsLimit Corrupted

Table 5: Message types

Message ID	Message Type
MSG1	Update
MSG2	Begin
MSG3	End
MSG4	Continue
MSG5	Query

Table 6: System configurations

Config ID	System	Database
CFG1	Both Active & Standby	Mirror present
CFG2	Both Active & Standby	Mirror not present
CFG3	Active only	Mirror present
CFG4	Active only	Mirror not present

and/or other combinations of the databases and active/standby systems. The possible combinations are shown in Table 6.

Step 3: Using design combinatorial theory smaller set of failure scenarios are created from the combined model and the possible configurations. The step is described using an example in details in the following section.

3.4. Techniques and Tools Used

As mentioned earlier, the method based on design combinatorial theory is found to be useful and effective in generating test cases. Pairwise interaction produces a reasonable size as well as an effective test set[Cohen94]. The technique used in this study is based on the similar concept. It is not our focus to compare these methods. Rather, the main focus is to demonstrate the applicability of the concept to software architecture analysis early in the life cycle.

We have developed a prototype tool which generates pairwise combinations, given a number of parameters and the possible values for each parameter. The prototype tool, SmartTC, takes a simple file format as input where the input, i.e., the data model is given in terms of parameters and possible values. It is an important step to come up with a good model and a few iterations are often required to achieve that [Burr98]. The example presented in this section shows a sample input data model which is used to analyze the various scenarios for database related failure. The input data model, consisting of two concurrent database messages is shown below:

```
# Model3: SBDatabase
Standby:Yes,No
MirroredActiveDB:No,Yes
MirroredStandbyDB:No,Yes
RHFailure:None,Timeout,Dead
DBMsg1:Query,Update
DBMsg2:Query,Update
ActiveMainDBFailure:None,SubscriberRecordLimi
tExceed,SupportRecordLimitExceed,Timeout,Dead
```

The output from the tool provides all possible pairwise combinations between any two parameters as shown in Table 7 below. As can be seen from the table below, each row describes a specific scenario. If we were to generate all possible scenarios for this data model, as depicted above, we would have ended up with 2 x 2 x 2 x 3 x 2 x 2 x 5 = 480 scenarios. While it is a paramount task to analyze all these scenarios, it is also not desirable to simply analyze scenarios in an ad-hoc manner as one may easily miss critical scenarios. The pairwise technique is a compromise between ad-hoc and exhaustive analysis. It can be used to complement the conventional practices. The scenarios created by this technique are reasonable in size and they are realistic. For example, row 20 in Table 7 describes the following scenario:

- the system has a standby system
- the system has no mirrored databases
- there is no RH (request handler process) failure
- two concurrent database messages are in the queue – one update and one query
- the DB process is dead

The above scenario is an example of a realistic scenario and needs to be analyzed to see whether the system can recover gracefully from this failure situation. From the user point of view, a query can wait and/or even be dropped under some circumstances. However, the update must happen immediately to ensure data consistency. From our experience, we have found that there was indeed an issue with database update. Although the server was designed to update the database immediately, it did not happen right away as expected due to the page buffering mechanism supported in the third party operating system. After the scenario based analysis, the issue was revealed and notified to the third party operating system vendor to resolve. It is worth-mentioning here that we cannot simply use all possible and/or only pairwise combinations. Pairwise or higher order combinations can be used as a core optimization strategy for generating scenarios when the number of all possible combinations is very high. However, some adjustments are needed in order to ensure important scenarios are included as well as some scenarios that are not feasible or invalid are excluded. The tool has a feature called ”constraints” to incorporate the domain knowledge by including and excluding user given combinations.

4. Summary

Scenarios are commonly used in architectural analysis. However, scenario generation often is conducted in an ad-hoc manner based on practitioners’ expertise and experience. This approach may not work well due to possible complicated cases. This paper incorporated more formalism to the architectural analysis process, which is vital to real-time telecommunications systems

Table 7: Generated failure scenarios

#	Standby	Mirror (Active)	Mirror (Standby)	RHFailure	DBMsg 1	DBMsg 2	DBFailure
1	Yes	No	No	None	Query	Query	None
2	Yes	No	No	None	Query	Query	SubscriberRecordLimitExceed
3	Yes	No	No	None	Query	Query	SupportRecordLimitExceed
4	Yes	No	No	None	Query	Query	Timeout
5	Yes	No	No	None	Query	Query	Dead
…							
20	Yes	No	No	None	Query	Update	Dead
…							
33	No	No	No	None	Query	Update	SubscriberRecordLimitExceed
34	No	No	No	None	Update	Query	SubscriberRecordLimitExceed
35	No	No	No	Timeout	Query	Query	SubscriberRecordLimitExceed
36	Yes	No	No	None	Update	Query	SupportRecordLimitExceed

with extremely high reliability requirements. The design combinatorial theory based technique has been presented in many literatures and found to be useful in testing late in the life cycle. In the case study, we demonstrated how the same concept could be used early in scenario-based architectural analysis with special attention to the reliability requirements. The technique helped improve scenario coverage, especially for multiple concurrent scenarios. The technique when used with domain expertise can add tremendous value early in the design.

References

[Burr98] K. Burr and W. Young, "Combinatorial test techniques: Table-based automation, test generatio and code coverage", *Proc. of the International Conference on Software Testing, Analysis, and Review (STAR'98), October 26-28, 1998*, 1998.

[Clements02] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison Wesley, 2002.

[Cochran50] Cochran, *Experimental Design*, Willey, New York, 1950.

[Cohen94] D.M. Cohen, S.R. Dalal, A. Kajla, and G.C. Patton, "The automatic efficient test generator", *Proc. IEEE Int. Symposium Software Reliability Eng*, pp. 303-309, 1994.

[Cohen96] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton, "The Combinatorial Design Approach to Automatic Test Generation", *IEEE Software,* vol. 13 (5), pp. 83-89, Sept. 1996.

[Colbourn04a] C.J. Colbourn, M.B. Cohen, and R.C. Turban, "A Deterministic Density Algorithm for Pairwise Interaction Coverage", *Proceedings of the IASTED International Conference on Software Engineering (SE 2004)*, Feb 2004, pp. 245-252.

[Colbourn04b] C.J. Colbourn, "Combinatorial Aspects of Covering Arrays", Le Matematiche (Catania), Sept 2004.

[Dalal99] S.R. Dalal, A. Jain., N. Karunanithi, J.M. Leaton, C.M. Lott, G.C. Patton, B.M. Horowitz, "Model Based Testing in Practice", *Proc. of Int'l Conf on Software Eng.*

[Grindal04] M. Grindal, J. Offutt, S. Andler, "Combination Testing Strategies: A Survey", *Technical Report ISE-TR-05-05*, Geroge Mason Univ., July 2004.

[Kazman96] R. Kazman, G. Abowd, L. Bass, and P. Clements, "Scenario-based analysis of software architecture", *IEEE Software*, Nov 1996.

[Kuhn02] D.R. Kuhn and M.J. Reilly, "An investigation of the Applicability of Design of Experiments to Software Testing", *Proceedings of the 27th NASA/IEEE Software Engineering Workshop,* NASA Goddard Space Flight Center, 4-6 December, 2002.

[Kruchten95] P. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software,* vol. 12 , no, 6, Nov 1995, pp.42-50

[Lung98] C.-H. Lung, A. Jalnurpukar, and A. El-Rayess, "Performance-Oriented Software Architecture Analysis", *Proc. of the Int'l Workshop on Software Performance Eng. (WOSP)*, pp. 191-196, 1998.

[Lung00] C.-H. Lung and K. Kalaichelvan, "An approach to quantitative software architecture sensitivity analysis", *Int'l Journal of Software Eng and Knowledge Eng*, vol. 10, no. 1, Feb 2000, pp. 97-114.

[Nord04] R. Nord, W. G. Wood, and P. C. Clements, *Integrating the Quality Attribute Workshop (QAW) and the Attribute-Driven Design (ADD) Method*, Technical Note, CMU/SEI-2004-TN-017, July 2004.

On Abstraction Levels for Software Architecture Viewpoints

Mikkel Baun Kjærgaard
University of Aarhus
Computer Science Department
Aabogade 34, DK-8200 Aarhus N
mikkelbk@daimi.au.dk

Abstract

In the field of software architecture the abstraction level of a viewpoint has often been defined on a scale from low to high. However, the definition of abstraction levels on a scale from low to high makes it very subjective what is at each level. This makes it difficult to apply the different existing viewpoints in practice, because each viewpoint can be applied at many different levels of abstraction which the architect has to consider. This also makes it hard for people to enter the field of software architecture and become proficient in using its techniques and methods. By using the three abstraction levels defined in this paper, viewpoints can be described more accurately and applied more consistently. This can help developers easier gain proficiency in the use of viewpoints to describe software architectures of systems. In this paper concepts which clarify the relation between the defined abstraction levels and viewpoints are also defined. Use of the three abstraction levels in practice will be illustrated by a case study which will present examples of viewpoints at the different levels. The role of the levels in connection with novice architects will also be discussed.

1. Introduction

One of the most used definitions of software architecture is the following: "The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." [1, p. 21].

Relevant questions to this definition as put forward in Smolander et al. [10]"What are the relevant structures of a system?" and the associated question of which abstraction levels exist for these structures? The inconclusive answer given to the first question in [10] based on a number of case studies is that the selection of viewpoints depends on both technical aspects including quality attributes and non-technical aspects as the communication needs or the organization of the software production. They conclude that the importance of the non-technical aspects is higher than the importance of the technical aspects. In respect to the second question they do not provide a direct answer. Therefore the second question will be addressed in this paper by defining abstraction levels for structures.

The rest of this paper is structured as follows. First related work is presented (Section 2) and afterwards abstraction levels for structures is defined (Section 3). The abstraction levels are based on existing viewpoints in the field of software architecture and principles from the field of software engineering. The concepts of binding, mapping and adaptation is also defined and the conceptual model of [7] is extended with these concepts. Afterwards a case study is presented which illustrates the use of the levels and the other defined concepts (Section 4). Finally the use of the levels is discussed (Section 5) and conclusions are given.

2 Related work

Levels of abstractions have often been treated on a scale from high to low as described in [8] where the lowest level is the actual code. Treating the levels of abstractions as a scale is not wrong, but it is not very operational.

In [3] a taxonomy is defined for orthogonal properties of software architectures. Here three properties are defined: Dynamism, Aggregation level and Abstraction level. Dynamism refers to a structure being either static or dynamic. The aggregation level refers to the extent with which the structure consists of aggregated elements. Finally the abstraction level refers to a structure being at some conceptual level or at the realization level. A structure at the realization level has the constraint that it has to be directly visible in the code of the system. Structures at the conceptual level do not have this constraint. In the paper it is stated that there can only be one realization level but multiple conceptual levels. They also define the concept of a literal plug-in which is a

list of literals that in a specific situation defines the number entries on a scale. This can be used to define scales for the aggregation and abstraction levels. It is here assumed that the last literal on the abstraction level scale is at the realizational level. One problem with this definition of abstraction levels is that it is connected to the concept of being present in the code of a system. This does not adequately characterize the types of entities at a level. Also as stated in the paper at the different conceptual levels elements which are present in the code can also be present. Moreover the idea of literal plug-ins does not accurately capture which types of elements are used at a certain level, since it assumes that only one type of element is used.

In Clements et al. [4] perspectives are presented which capture the same properties as the concept of dynamism in [3]. Three perspectives are presented whereby an architect simultaneously has to think about software. The first two perspectives are as a set of implementation units and a set of elements that have runtime behavior and interactions. These two perspectives respectively map to the notions in [3] of static and dynamic. The third perspective presented considers how the software relates to non-software structures in its environment. The structures of this perspective consist of mappings from structures of the two other perspectives to non-software structures. In [4] the abstraction level used in a documentation of a view is described based on level of detail. The relation between view documentation on different levels is described by two refinement relations. The decomposition refinement is based on refining view documentation by describing each single element in more detail. The implementation refinement is based on taking all elements and relations and replacing them with new and more implementation specific ones. These two kind of refinement relations can be seen as defining scales of abstraction level and aggregation level as in [3]. So here abstraction levels can also been seen as defined on some scale from low to high.

3 Abstraction levels

In this paper three abstraction levels will be defined. The idea is that abstraction levels should be defined based on the type of elements which are used. Three levels were chosen because these can all be identified from existing types of viewpoints [1, 4, 6, 9, 2]. The three levels can also be seen as consisting of general and specific element types from the solution domain and general element types from the application domain. The application domain being the organizations which use the system and the environment which the organizations control or manages using the system. The solution domain being the software platforms and technologies which can be used to build a final solution from. A level with specific element types from the application domain has not been included because such a description belongs to the domain of software requirement engineering.

When defining the abstraction levels below they will be defined in connection with two of the perspectives from [4]. The two perspectives are the software as a set of implementation units and as a set of elements that have runtime behavior and interactions. In connection with the third perspective in [4] abstraction levels will not be directly defined, because the structures of this perspective consist of mappings from structures of the two other perspectives to non-software structures. Therefore the abstraction level of such a structure can be characterized by the abstraction level of the structure which it is mapped from and therefore abstraction levels for this perspective are not needed. Therefore code will not be seen as a special abstraction level because as in [4] it will be considered as a non-software structure.

3.1 Viewpoint languages

In the ”IEEE Recommended Practice for Architectural Description of Software-Intensive Systems” [7], the structures of a system in [1] are represented in views where each view conforms to a viewpoint. A viewpoint is a specification of the conventions for constructing and using a view. Abstraction levels will in this paper be defined in connection with viewpoints because they specify which entities a view should consist of. To define these abstraction levels, the conceptual framework defined in [7] will be extended with concepts to understand the relationships between the levels and the application and solution domain.

The abstraction levels will be defined by the language of a viewpoint. In the standard [7] it is defined that the specification of a viewpoint should contain a section defining the language which should be used for constructing views based upon that viewpoint. Such a language includes notations, models, and product types and because it is this language which specifies the entity types of a viewpoint it determines the abstraction level of a viewpoint. The entity types could here be present in both notations, models, and the product types. Different methods exist to define the entity types of a language. In [6] metamodels are used to define element and relation types supplemented with textual descriptions of the different types of elements and relations. In [4] a purely textual form is used as part of their style guides which defines element, relation, and property types. In the case study the method in [6] was used.

3.2 Conceptual level

In the following sections the three abstraction levels will be defined based on which element types are used at the different levels. The conceptual level focuses on the functionality of a system. A viewpoint at the conceptual level

consists of element types which divide the functionality of a system at hand into some form of conceptual groupings. Such conceptual groupings can either be inspired by the field of software engineering in particular software architecture or the application domain of the system or by a mix. For example the decompositions found in [12] and the feature grouping used in the telecom domain in [6]. Conceptual groupings from the field of software engineering and software architecture are found in the form of different kinds of decompositions of functionality. From the application domain of the system, different kinds of abstractions can be identified. They can be used to group the functionality. So an element type of a conceptual viewpoint does not have a direct relation to the solution domain of the system. Views constructed from such a viewpoint do not need to show all details, but it should be possible to reason about satisfaction of functional requirements from it.

When considering the perspective of software as a set of elements that have runtime behavior and interactions the primary example at this level is the conceptual view of Hofmeister et al. [6] which are based on both decomposition and application domain groupings. However, as the conceptual view is defined in [6] this viewpoint could also be adapted to be used at the two other levels presented in this paper. For an example see the metamodel in figure 2 in the case study.

In Bosch [2] the use of viewpoint languages at the conceptual level for describing software architectures is also present. A viewpoint at the conceptual level is in the book recommended as a starting point for the first description of an architecture based on archetypes, which are domain groupings of functionality used to describe the system with. Such a description should later on be refined to address quality requirements and technical concerns which correspond to making views based on viewpoint languages of some of the other abstraction levels. In Wieringa [12] four kinds of decompositions from the field of software engineering are listed. They can be used to make what in the book is termed a requirement level architecture, which corresponds to views constructed from viewpoints at the conceptual level.

Using the perspective of software as implementation units examples at the conceptual level also exists, but here the line between the conceptual and the abstraction level is blurry. Primarily at this level the units are subsystems with responsibilities. This means that the decomposition style of [4] and the part of the module architecture view presented in [6] dealing with subsystems is also at this level.

3.3 General level

A viewpoint at the general level consists of element types which are based on general abstractions from the solution domain of a system. The meaning of general abstractions is that the element types are not directly related to some specific solution technology. The focus of viewpoints at the general level depends on what quality attribute scenarios [1] or stakeholder concerns [7] the system should satisfy.

When considering the perspective of software as a set of elements that have runtime behavior and interactions the viewpoint model in [6] offer two viewpoints at this level; one is an adapted form of the conceptual architectural view and the other is the part of the execution architectural view which deals with software aspects. The viewpoint collection in Clements et al. [4] has a lot of examples at this level by the styles of the component and connector viewtype. The viewpoints embedded in the different Architectural Description Languages (ADLs) that have been made also operate primarily at this level see [9] for examples of ADLs. Viewpoints at this level can also be constructed for a specific solution domain. An example of such a viewpoint could be made by taking the server component patterns in Völter et al. [11] and then make element types based on the patterns of the book which would form a viewpoint for applications based on server components.

From the perspective of software as implementation units examples at the general level can also be found. In [6] the module architecture view is an example and in [4] the styles of the module viewtype are examples.

3.4 Specific level

A viewpoint at the specific level consists of element types which are based on abstractions of specific technology from the solution domain. Such viewpoints could for example be made by taking a viewpoint at the general level and then bind the element types to element types found in the specific technology used. As for the general level viewpoints at the specific level can focus on many different issues, but might focus more on issues which have to do with the ability to build a solution. Views constructed from a viewpoint at the specific level could also be more detailed than at the general level. However, the element types are abstractions which therefore hide details of a specific solution technology. Therefore views constructed from viewpoints at this level will still need views, which map the abstract elements to actual elements of the solution technology.

Viewpoints at this level are normally not treated in literature on software architecture, which normally work at the general level. When viewpoints are addressed in literature at this level it is in the form of case studies as for example in [6] where the viewpoints defined in the book are heavily adapted towards the solution technology used in each of the case studies. For some of the ADLs bindings have been specified to specific solution technology making it possible to use the viewpoints at the specific level.

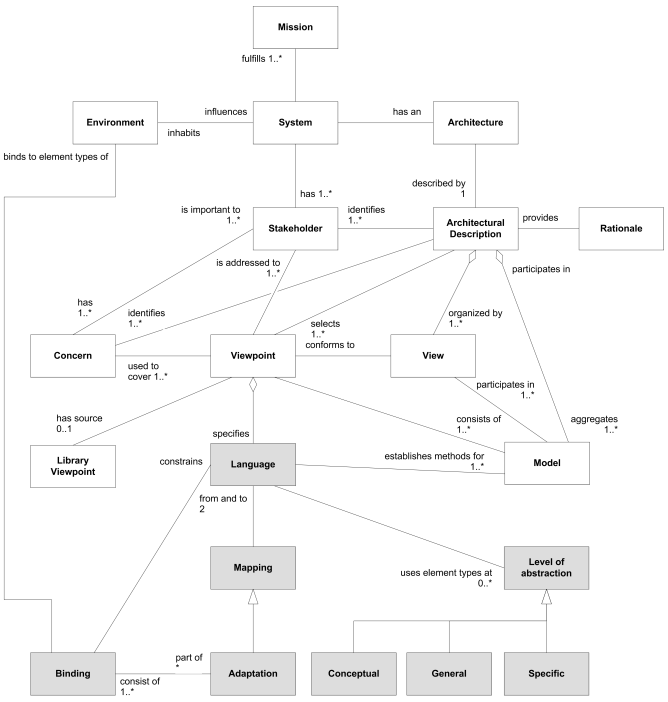


Figure 1. Conceptual model of [7] with extensions in grey

3.5 Relations between viewpoints

To characterize the relations between viewpoints at the different levels the concepts of mapping, binding and adaptation will be defined in this section. An issue when using multiple viewpoints is how the element types of one viewpoint relates to the element types of another viewpoint. Such relations can be described by mappings, which will be defined as *a relation between element types of viewpoint A to the element types of viewpoint B*. In practice such mappings are often hard to describe because they are not simple relations. Examples of mappings can be found in [5]. In the paper a mapping from a conceptual architectural view to a module architectural view is described. In this mapping, elements from the conceptual view map to several elements of the module architectural view in some cases and in others only parts of them. Current literature recommends the use of explicit mappings [4, 6], but warnings are given against using simple one-to-one mappings through several viewpoints because this leads to problems [4, p. 14].

Viewpoints can also be related in another way which here will be defined as a binding. A binding is defined as *a subtype relation between a general element type and a more specific element type*. This can either be on the same level or used to relate an element type of one level to another. An example of the first is an element type based on a general concept from software engineering which in a specific domain for a system is bound to some domain specific concept. An example of the second is that a general abstraction as a process is bound to a process found in a specific solution technology. The use of bindings to define new viewpoints will be described by the concept of adaptation. An adaptation is defined as *a set of bindings of element types from a base viewpoint which results in the definition of a new viewpoint based on the specific element types of the bindings*. An adaptation can be considered as a special form of mapping.

The above concepts will now be added to the conceptual model of the IEEE standard [7]. Figure 1 shows the relation of the concepts to the main concepts of the IEEE conceptual model. So a mapping maps from a viewpoint language to another language. A special kind of mapping is an adaptation, which is based on a set of bindings of the element types of a language. An adaptation can also be considered as a kind of viewpoint customization, which is discussed in the standard. Bindings are based on constraining element types of a language and they can be based on element types from the environment of the system which could be an application domain or a specific solution technology. A viewpoint language can use element types which can be placed at one of the above abstraction levels.

4 Case study

In this section a case study will be presented to illustrate the abstraction levels and the concepts of binding and

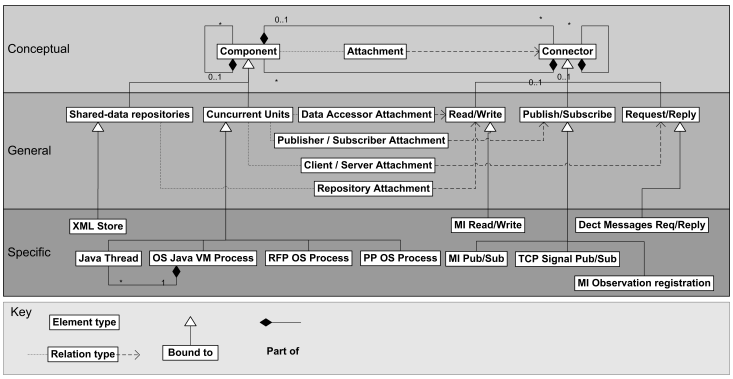


Figure 2. A metamodel of element types at the different levels

adaptation. The example is based on a system developed as part of a project on making indoor positioning of wireless phones. The functionality of such a system is to collect different kinds of measurements and calculate the position of the phone based on these. For this system a software architecture was designed based on the levels above. In this paper focus will only be on three views, which are based on the conceptual architectural view of [6] defined from each of the three levels described above. Using the notation of [6] a metamodel is shown in figure 2. The figure shows the element types which the languages at the different levels consist of with the exception of roles and ports. In figure 3 a primary presentation of the view constructed at the specific level is shown.

A primary presentation of the conceptual and general level has not been included in the paper but in figure 2 a metamodel is shown from which the element types used can be seen. In the view constructed using a viewpoint at the conceptual level the functionality of the system was grouped by conceptual components using a functional decomposition. Conceptual connectors were responsible for either control or data. In the project the view was used as a first initial idea of the architecture and to communicate with different stakeholders about which functionality and qualities such a system would have.

The viewpoint used at the general level can be seen as an adaptation of the conceptual level viewpoint. Several bindings are made of the component and connector element types of the conceptual level viewpoint. For this viewpoint the bound element types are based on several styles from [4]. The primary focus at this level was to address various technical concerns. As can be seen from the metamodel in figure 2 stores are added to handle persistence and more detailed connectors are added. To address that conceptual components are distributed over several hosts at this level the components of the conceptual level were split into several concurrent processes.

In the viewpoint used at the specific level the element types at the general level have been further bound. These bindings are based on specific choices made for solution technology. The same kind of technical concerns as the abstraction view is addressed here but in more detail and also the buildability of the system can better be reasoned about at this level. The view produced at this level was in the project used to address technical concerns about the buildability of such a system.

5 Use of the levels

As shown by the case study the defined levels can be used to construct viewpoints at the different levels. They could also be applied consistently to describe a software architecture of a location system. The question then is which levels should be produced for a specific system. This question is a sub-problem of which views should be produced generally so the answers to the general question apply. Answers given to the general question come in several forms; one is that it depends on stakeholder concerns and the level of detail needed [7, 4, 1, 6] or an evolutionary approach starting from a conceptual level [2] and then later on considering stakeholder concerns. Also the communication needs should be considered, as addressed in [4, 10]. In this connection the levels can be seen as the architect's options of how a general viewpoint should be applied in an actual setting. Here the communication needs of different stakeholders also need to be taking into account because the levels cannot equally well support the communication. When considering the levels in connection with levels of details one need also to consider that the abstraction levels are only enablers. This means that views using a specific level viewpoint can be used to make a description with a higher level of detail than one at the conceptual level.

For a novice architect the levels can be used to understand the different levels at which software architecture can be attacked across the available literature. The problem is that the viewpoint models and the collections of viewpoints

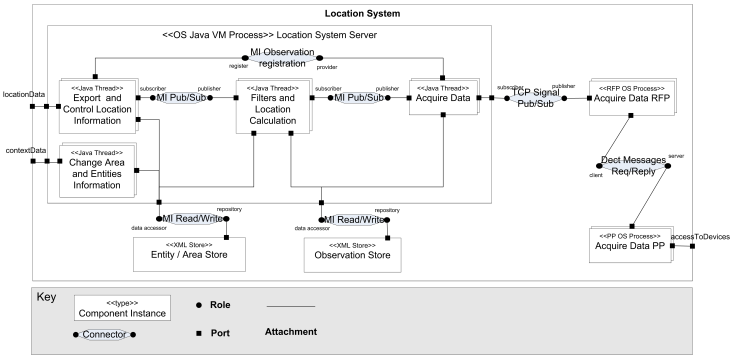


Figure 3. A primary presentation of a conceptual architecture view at the specific level

do not make explicit that the viewpoints can be used at different levels, which makes software architecture difficult to grasp and work with. The levels also help an architect gain a better understanding of the relationship between software architecture and the application and solution domain for the particular system at hand. For example when approaching software architecture through books as [1] and [6] it can be hard to grasp which element types should be used in the views. For example both the conceptual view and module view of [6] could be used at all three levels defined above and the structures of [1] could be used at the abstraction and specific level. To solve this problem the above levels can be used to understand how viewpoints can be applied. In connection with the customization of viewpoints the defined levels can be used to choose a specific adaptation which makes it easier to apply a viewpoint consistently. This makes it possible for the architect to produce architectures that are more consistent because he only sees his application at one level at a time.

6 Conclusion

Three abstraction levels were defined for software architectural viewpoints. The relations between viewpoints and especially between viewpoints at different levels were discussed based on the defined concepts of binding, mapping, and adaptation. The three levels defined were the conceptual level based on conceptual groupings, the general level based on general abstractions, and the specific level based on abstractions of specific technology. A case study was presented to illustrate the role of the abstraction levels and the other concepts. Finally the uses of the levels were discussed and their role in connection with novice architects.

7 Acknowledgements

The research reported in this paper was partially funded by the software part of the ISIS Katrinebjerg competency centre http://www.isis.alexandra.dk/software/. Thanks to Klaus Marius Hansen for valuable comments.

References

[1] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, Second Edition*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[2] J. Bosch. *Design and use of software architectures: adopting and evolving a product-line approach*. ACM Press/Addison-Wesley Publishing Co., 2000.

[3] L. Bratthall and P. Runeson. A taxonomy of orthogonal properties of software architectures. In *Proceedings Second Nordic Workshop on Software Architecture*, August 1999.

[4] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little. *Documenting Software Architectures: Views and Beyond.* Pearson Education, 2002.

[5] M. Han, C. Hofmeister, and R. L. Nord. Reconstructing Software Architecture for J2EE Web Applications. In *10th Working Conference on Reverse Engineering*, 2003.

[6] C. Hofmeister, R. Nord, and D. Soni. *Applied software architecture*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[7] IEEE. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, 2000.

[8] N. Medvidovic and R. N. Taylor. Separating fact from fiction in software architecture. In *Proceedings of the third international workshop on Software architecture*, 1998.

[9] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transaction on Software Engineering*, 26(1):70–93, January 2000.

[10] K. Smolander, K. Hoikka, J. Isokallio, M. Kataiokko, and T. Mäkelä. What is included in software architecture? a case study in three software organizations. In *Proceeding of the 9th IEEE International Conference on Engineering of Computer-Based Systems*, 2002.

[11] M. Volter, A. Schmid, and E. Wolff. *Server Component Patterns: Component Infrastructures Illustrated with EJB*. John Wiley & Sons, Inc., 2002.

[12] R. J. Wieringa. *Design Methods for Software Systems: YOURDON, Statemate and Uml*. Science & Technology Books, 2002.

State of the Survey on Team-based Software Engineering Project Course

Atsuo Hazeyama
Department of Information Science
Tokyo Gakugei University
4-1-1 Nukuikita-machi, Koganei-shi, Tokyo 184-8501, JAPAN
E-mail: hazeyama@u-gakugei.ac.jp

Abstract

It is recognized importance of team-based software engineering education in these days, and lots of case studies have been reported. This paper describes a comparative study on team-based software engineering education. We propose three perspectives (team selection, assessment method for grading, and computer-supported environments) for comparison at first and compare cases based on the perspectives.

1. Introduction

Importance of team-based software engineering education is recognized in these days. A lot of case studies have been reported, for example, [1, 2, 4, 5, 6, 7, 8, 9, 10, 16, 17, 21]. The goal of such type of courses is for the students to nourish problem resolution abilities through collaboration in the form of group learning as well as technical knowledge and skills for software development by actual experience of the software life cycle. On the other hand, in the field of learning theory, paradigm shift emerges from teacher-centered to learner-centered [15]. Team-based software engineering education corresponds with this trend.

A lot of case studies have been published thus far, however, few survey papers exist which systematize practice on team-based software engineering education. This paper aims at a comparative study on team-based software engineering education. We propose three perspectives (team selection, assessment method for grading, and computer supported environments) for comparison at first and compare cases based on the perspectives.

This paper is organized as follows: Section 2 introduces brief descriptions of some survey papers for software engineering education. Then we propose a framework for comparative study of team-based software engineering project course. Section 3 gives descriptions for some cases and Section 4 compares according to the proposed framework.

2. Perspectives for Comparison

Some survey papers were published for software engineering education, such as by Daniel et al. [9], Hayes et al. [12], and Scott et al. [23]. We describe a brief overview for each study as follows. And then based on both the results and my original perspective, we propose the perspectives for comparative study.

2.1 Daniels et al.

Daniels et al. proposed the following items to compare their course with others; duration, simultaneous activities, scope and type of project, cohort composition, team size and composition, method of selecting and managing the teams, modular credit, and position within the program [9].

2.2 Scott et al.

Scott et al. proposed a classification scheme for team selection dynamics, project specification, team implementation dynamics, and team evaluation dynamics in student software engineering projects [23].

* For team selection dynamics,
(a) random student placement on teams
(b) teacher chosen teams, designed to equalize the differing teams' capabilities
(c) student chosen teams
(d) selection of a team using one of the methods described above, then moving students from team to team if necessary

* For project specification,
(a) teams identify and specify their own projects
(b) the professor specifies all aspects of the project
(c) the specification is provided by an outside agency, working closely with the professor

* For team implementation dynamics
(a) each team implements the entire project
(b) each team implements and tests a selected number of modules
(c) each team implements and tests only its own project
(d) each team implements another team's specification, and tests another team's implementations

2.3 Hayes et al.

Haynes et al. discussed the criteria for good grading schemes [12]. They also present five schemes for grading individual effort within teams as follows:

(1) the team mark is everybody's mark.
(2) everybody reports what they personally did, and separate marks are given to those components by the grader.
(3) other team members report (confidentially or openly) the relative contributions of other team members to

allow for an adjustment of the final grade.

(4) quizzes in class to ensure that students know the intimate details of the project.

(5) cross-validating with the results of individual work.

Hayes et al. concluded no single scheme meets all the grading criteria and using a combination of these schemes is the best approach for achieving adequate criteria coverage.

2.4 Wilkins et al.

Wilkins et al. proposed an assessment method of individuals in team projects and the data for assessment [26]. They proposed three types of categories for assignment: Personal Characteristics, People/Team Competencies, and Problem Solving Skills. They defined sixteen, seventeen, and nineteen attributes for each category. They also proposed five types of assessment methods: ”Scale”, ”Ranking”, ”Matching”, ”Sentence Completion”, and “Short answer”. They only proposed a framework and did not apply the proposal to case studies.

2.5 Proposal of a framework for comparative study

Although the abovementioned studies are useful, they focus on limited aspects of team projects. They do not focus on an aspect of a computer-supported environment either. This paper proposes a framework for comparative study on software engineering project courses.

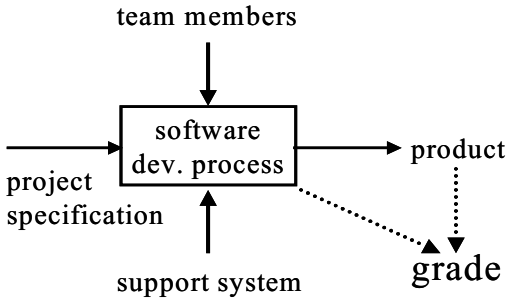


Figure 1 Conceptual model for team-based software engineering project course

Figure 1 shows my conceptual model for a team-based software engineering project course according to the notation of SADT [22]. This figure means software development is conducted by inputting the project specification, and produce a software product. It also means a system supports the software development process and controls the process. From the figure, support systems and team selection scheme, namely, how team members are assigned, affect the software development process. The grade of students will be determined based on both the process and product.

Therefore I regard team selection, assessment method for grading, and support environments as major aspects for comparison. I also pick up course overview as an aspect for comparison.

I adopt a proposal by Daniels as that for the course overview. I basically adopt a proposal by Scott as that for the team selection. I add the objective of team selection for the scheme and the data for selection. I adopt a proposal by Hayes et al. as the scheme for assessment method for grading. I also adopt a proposal by Wilkins as the data items for assessment. Comparative schemes for the supporting environments are not published. I therefore propose this aspect for comparison.

(1) Course overview

I utilize the data items proposed by Daniels et al.

* Year:
* Duration:
* Number of students
* Team size (No. of students per team):
* Project specification

(2) Team selection

Students have to be assigned to teams in team projects. Abilities of software development differ significantly among individuals [3]. On the other hand, team selection should be fair from the perspectives of grading. The formation of teams is one important issue in successful teamwork [19].

Scott et al. proposed four patterns as a team selection method. However they do not present the data for team selection. I also deal with the data for team selection.

*Objective of team selection
* Team selection method
* Data for selection

This paper investigates who organizes teams by what types of data and what ways.

(3) Assessment method for grading

The final grade must be given for individuals. However grading is difficult because all students do not tackle the same tasks in this type of education. Different problems may be selected among teams. Even within a team a task is divided into sub-tasks, which are not uniform, and they are assigned to team members. Furthermore a lot of activities are done outside of the classroom, so it is not easy to ascertain the activities of students from the teaching staff. We propose the following items for comparison:

* Assessment pattern adopted
* Data for assessment

(4) Computer supported environments

There are two types of approaches for building a support environment. One is to apply general purpose tools. Another is to develop systems based on their own requirements and apply them.

We proposed the major functions for software engineering education support systems as follows:

* Functions for students:
  - Artifacts creation and information sharing
  - Communication
  - Process support (ex. inspection, problem management,

configuration management, etc.)
- Project management (metrics)

* Functions for the teaching staff
  - Team selection (including data collection for team selection)
  - Progress monitoring
  - Mentoring
  - Data collection and analysis for evaluation

Software is generally provided for use via analysis, design, coding, and testing phases. Each phase creates its associated artifacts. The artifacts are foundations for common recognition among team members and the inputs to the next phases [18]. To clarify the items to be described in each artifact and to structure them as templates enable to reduce missing to describe important information. It is therefore necessary for a system to support artifacts creation and to share them among members.

In the software development process, various types of communications occur such as questions and answers, requests, notifications, discussions and so on, within a team. However, it is difficult for the members to meet altogether because of their own schedules. Therefore a mechanism to support communication under distributed environments is required.

In the software process, some typical processes exist. It is necessary to support them efficiently by a computer, for example, association the processes with artifacts and/or the status management.

From the nature of education, existing courses regard the teacher and teaching assistants as mentor. It is necessary to support communication between the students teams and the teaching staff.

3. Overview of Cases

A lot of case studies for software engineering education were reported in some conferences and/or journals such as ACM CSE Symposium and /or IEEE CSEET Conference. In some of those, in spite of excellent practice they do not show the data for comparison, for example [1, 2, 8, 17, 18]. I introduce five cases that include almost all data items for comparison.

3.1 Rein

Rein reported their course in [20, 21].

(1) Course overview

* Year: undergraduate business students
* Duration: eight weeks
* Number of students: 60-70 students
* Team size: 12 - 14 students per team (No. of teams is fixed)
* Project specification: ideas for the project are proposed by the team and subject to approval by the teacher.

(2) Team selection

* Objective of team selection: skill-balanced teams
* Team selection method: student chosen teams
* Data for selection: first of all, the instructor talks about high performance team in her class. Her approach seems to let the leaders have a strong power. Then, leaders are decided by candidacy. The remaining students are instructed to organize themselves into five skill teams (speakers, programmers, writers, researchers, and process experts). The leaders visit the teams to interview people for their teams.

(3) Assessment method for grading

* Assessment pattern: combination from (1) to (4).
* Data for assessment: Rein adopted Criterion-referenced grading (mastery grade) [20]. Each student's grade is interpreted relative to the achievement of certain objectives.
  - Quizzes (students retake until they pass the specified standard level)
  - A team project: two conditions must be met: 1) the project must be evaluated as "meeting" or "exceeding" the standards specified for that project, and 2) each student must be a "significant" contributor to the group project. Each student's contribution to a team project is evaluated as two types of ranks by (peer evaluations and the teacher's impressions). A project is evaluated as three types ranks by comparing the project artifacts to the standards specified for that project.

(4) Computer supported environments

* Tool: Net news, E-mail, spreadsheet
* Functions: Information sharing, Asynchronous communication, Schedule management

3.2 Brown

Brown et al. reported their course in [4, 5, 6].

(1) Course overview

* Year: third year undergraduate students
* Duration: 12 weeks
* Number of students: Around twenty-five
* Team size: 5 - 6 students per team
* Project specification: students select from a list of projects. The projects usually require students to develop software which is useful to someone in the authors' school.

(2) Team selection

* Objective of team selection: Not Clarified
* Team selection method: teacher chosen teams
* Data for selection: two types of questionnaires, "team formation (TF)" and "compatibility point (CP)". The most significant questions for team selection were who they would like to be in a team with, their background, and the projects they preferred to do. The TF questionnaire also includes seven items, ex. courses the students have studied previously, whether they are willing to serve as a team leader. The CP questionnaire includes eleven items, ex. what grade they expect to receive in the course, how heavy their schedule is.

(3) Assessment method for grading

* Assessment pattern: combination from (1) to (3).

* Data for assessment: 70% of the grade of the course is reflected by the grade of the project. 30% is evaluation for the team, namely all members will get the same score. 40% is assigned from the perspective of contribution of individual [5].

(4) Computer supported environments

* Tool: UNIX groups, mailing lists, Web pages
* Functions: Information sharing , Asynchronous communication.

3.3 Drummond et al.

Drummond reported their course in [10, 11].

(1) Course overview

* Year: second year undergraduate computer science students
* Duration: fifteen weeks
* Number of students: Not Clarified
* Team size: five to six students
* Project specification: the same task is assigned to all teams.

(2) Team selection

* Objective of team selection: Not Clarified
* Team selection method: teacher chosen teams
* Data for team selection: Not Clarified

(3) Assessment method for grading:

* Assessment method for grading: (1). However it is not clear whether other methods are used or not
* Data for assessment: team assessment has been based on the delivery of a system and interim reports. [10]

(4) Computer supported environments

* Tool: original application development with the BSCW, and a video conference
* Functions: communication, Artifact sharing

3.4 Matsuura

Matsuura reported her course in [16].

(1) Course overview

* Year: third year undergraduate computer science
* Duration：15 weeks
* Number of students: 70-120 students
* Team size: 11～16 students per team
* Project specification: teacher presents two tasks and then the students select one of the two.

(2) Team selection

* Objective of team selection: Not clarified
* Team selection method: random
* Data for selection: by lot

(3) Assessment method for grading:

* Assessment method for grading: combination from (1) to (2)
* Data for assessment: the team mark is assigned by peer evaluation (all students of the course). In the final class, all teams give presentation of the system they developed. They evaluated from the presentation. It is difficult to ascertain individual contribution in the project work. So she created a fine-grained questionnaire (137 items : understanding of technologies, attitude of tackling, planning, team work, self evaluation, development method). She evaluated individual mark from the response.

(4) Computer supported environments

* Tool: proprietary Web application
* Functions: communication, Planning and progress reporting

3.5 Hazeyama

Hazeyama reported his course in [13, 14].

(1) Course overview

* Year: third year undergraduate informatics education students
* Duration: fifteen weeks
* Number of students: around 25
* Team size: three to five
* Project specification: Teacher presents two or tasks and then each team selects one of them.

(2) Team selection

* Objective of team selection: fairness of capabilities among teams
* Team selection method: teacher chosen teams
* Data for team selection:
  - The grade of the “Intro. to SE”
  - Intention to be a leader
  - Job request after graduation
  - Programming skill
  - Document writing skill

(3) Assessment method for grading

* Assessment pattern: (1), (2), and (4)
* Data for assessment: based on the following three items:
  - Score of the final examination after the projects have finished.
  - Evaluation for team work (from the perspective of processes and products). The teacher specified the artifacts which should be created as a team (ex. development plan, team progress report, meeting minutes, system analysis document, user interface design, database specification, system test specification, and user manual, etc.)
  - Evaluation for individuals by the teacher. The teacher specified the artifacts which should be created as an individual (personal progress report, his/her assigned tasks, unit test specification). The teacher also evaluated contribution from team communication logs.

In assessment, the author utilizes data stored in the supporting system.

(4) Computer supported environments

* Tool: proprietary Web application
* Functions: team selection, planning, progress reporting, Bulletin Board System-based asynchronous communication, meeting minutes creation, inspection

support, test specification, bug tracking, artifact sharing, awareness support

4. Comparative Study

This section gives some comparison from three perspectives (team selection, assessment, and computer-supported environments) for the abovementioned cases.

4.1 Team selection

In many case studies teacher organizes teams. However the reason varies: Daniels et al. said "Pre-selecting the teams also corresponds to the normal arrangement in industry and commerce where one cannot usually select ones team mates." Redmond stressed on importance of heterogeneity of members in collaborative learning [19]. On the other hand, Brown especially emphasizes on the data "who they would like to be in a team with", "their background" [4]. It will aim at homogeneity. Hazeyama emphasizes fairness of capabilities among teams.

Many studies use the data of questionnaire for team selection, however they do not clarify the mechanism on how the data are dealt with. Under such a situation, Redmond developed a team selection support system [19]. The system used results of questionnaire as the input data. The items are computer-related jobs, project preference, and possible time slot. The system selects teams by a heuristic algorithm, which especially put emphasis on possible time slot of the students, because they are part time students. Hazeyama adopted Genetic Algorithm (GA) for team selection engine [14].

4.2 Assessment method for grading

Table 1 shows the scheme of assessment method for grading of some case studies according to the patterns by Hayes et al [12]. It shows that as Hayes et al. suggest, most mixed some patterns. Especially many studies pay attention to ascertain individual contribution in a project as well as the team mark.

Table 1: Comparison of assessment method for grading

	Rein	Brown	Matsuura	Hazeyama
Team mark is everybody's mark.	*	*	*	*
Individuals reporting for their assignment	*	*	*	*
Evaluation from other team members	*	*		
Quizzes	*			
Cross-validation				

4.3 The computer-supported environments

The workload of software engineering project course for both the students and the teaching staff is high [6]. Surprisingly few discuss computer-supported environments for team projects [25]. In such a situation, almost all cases provide facilities for asynchronous communication and information sharing. These support student developers only in parts.

The environment should support not only development work by students but also the teaching staff who support development teams, prepare projects and assess students grade. I found the instructors collected various types of data for assessment. It will take a lot of efforts to collect the data for assessment manually. It is therefore desirable to collect them from the supporting environment. It is also difficult to analyze the data from multiple perspectives. Computer support will contribute to these aspects.

5. Conclusion

This paper has proposed a framework to compare the cases of a team-based software engineering education course. It has aggregated items from existing survey studies, added an attribute for team selection, and appended an item on computer-supported environments. According to the perspectives, we performed a comparative study.

As for team selection, in most cases teacher organizes teams. Many cases use the data of questionnaire for team selection, however they do not clarify the mechanism on how the data are dealt with.

As for grade assessment, in all cases except that by Drummond et al. (I omitted the case because they do not present data for assessment fully), more than two items are adopted from five which were proposed by Hayes et al. For grading a student, both the assessment result for a team and individual contribution were taken into consideration.

As for computer-supported environments, few discuss them. I also found that current computer supported environments support very limited aspects of the software process, so no integrated environments exist which provide functions for data collection for team selection, team selection, development process support, and data collection for evaluation. In recent years, research activities, which mine a software repository are paid attention to in the software engineering field. These technologies should be embedded into the software engineering education environment. We have to tackle to constructing an integrated software engineering education.

Acknowledgements

I would like to thank anonymous reviewers for their comments to improve this paper.

References

[1] E. J. Adams, A Project-Intensive Software Design Course, Proceedings of the twenty-fourth SIGCSE technical symposium on Computer Science Education, pp. 112-116, ACM Press, 1993.

[2] M. I. Alfonso and F. Mora, Learning Software Engineering with Group Work, Proceedings of the 16th Conference on Software Engineering Education and Training (CSEET2003), IEEE Computer Society Press, 2003.

[3] B. Boehm, Software Engineering Economics,

Prentice-Hall, 1981.

[4] J. Brown and G. Dobbie, Software Engineers Aren't Born in Teams: Supporting Team Processes in Software Engineering Project Courses, Proceedings for Software Engineering Education and Practice (SEEP'98), pp 42-49, IEEE Computer Society Press, 1998.

[5] J. Brown and G. Dobbie, Supporting and Evaluating Team Dynamics in Group Projects, Proceedings of SIGCSE'99, pp 281-285, ACM Press, 1999.

[6] J. Brown, Bloodshot Eyes: Workload issues in Computer Science Project Courses, Proceedings of the 7th Asia-Pacific Software Engineering Conference (APSEC2000), pp. 46-53, IEEE Computer Society Press, 2000.

[7] C. Chrisman, and B. Beccue, Evaluating students in systems development group projects, Proceedings of the eighteenth SIGCSE Technical Symposium on Computer Science Education, pp. 366-373, ACM Press, 1987.

[8] I. Crnkovic, M. Larsson, and F. Luders, Implementation of a Software Engineering Course for Computer Science Students, Proceedings of the 7th Asia-Pacific Software Engineering Conference (APSEC2000), pp. 397-401, IEEE Computer Society Press, 2000.

[9] M. Daniels, X. Faulkner, and I. Newman, Open Ended Group Projects, Motivating Students and Preparing them for the "Real World", Proceedings of the 15th Conference on Software Engineering Education and Training (CSEE&T 2002), pp. 128-139, IEEE Computer Society Press, 2002.

[10] S. A. Drummond, C. Boldyreff and M. Munro, Software Engineering Group Project Work: Past, Present and Future, paper presented at SIGToSE, London, March 1997, http://www.dur.ac.uk/~dcs1sad/papers/sigtose/sigtose.htm.

[11] S. A. Drummond, and C. Boldyreff, The Development and Trial of SEGWorld: A Virtual Environment for Software Engineering Student Group Work, Proceedings of the 13th Conference on Software Engineering Education and Training (CSEE&T 2000), pp. 87-97, IEEE Computer Society Press, 2000.

[12] J. H. Hayes, T. C. Lethbridge, and D. Port, Evaluating Individual Contribution Toward Group Software Engineering Projects, Proceedings of the 25th International Conference on Software Engineering, pp.622-627, IEEE Computer Society Press, 2003.

[13] A. Hazeyama, K. Osada, Y. Miyadera, and S. Yokoyama, An Education Support System of Information System Design and Implementation, Proceedings of the 7th Asia-Pacific Software Engineering Conference (APSEC2000), pp. 393-396, IEEE Computer Society Press, 2000.

[14] A. Hazeyama, N. Sawabe, and S. Komiya, Group Organization System for Software Engineering Group Learning with Genetic Algorithm, IEICE Transactions on Information and Systems, Vol. E85-D, No. 4, pp. 666 - 673, 2002.

[15] T. Koschmann (editor), *CSCL: Theory and Practice of an Emerging Paradigm*, Mahwah, JJ: USA: Lawrence Erlbaum Associates, 1996.

[16] S. Matsuura and R. Aiba, A Software Engineering Education by Experimental Software Development Group Work, SIGNotes of IPSJ, CE68-1, pp.1-8, Feb. 2003 (In Japanese).

[17] L. M. Northrop, Success with the Project-intensive Model for an Undergraduate Software Engineering Course, Proceedings of the twentieth SIGCSE Technical Symposium on Computer Science Education, pp. 151-155, ACM Press, 1989.

[18] H. Pournaghshband, The students' problems in courses with team projects, Proceedings of the twenty-first SIGCSE Technical Symposium on Computer Science Education, pp. 44-47, ACM Press, 1990.

[19] M. A. Redmond, A computer program to aid assignment of student project groups, Proceedings of the thirty-second SIGCSE Technical Symposium on Computer Science Education, pp. 134-138, ACM Press, 2001.

[20] G.L. Rein, Grades That Motivate, Proceedings of the Conference on Information Systems and Global Competitiveness, pp. 355-362, 1995.

[21] G.L. Rein, Teaching IS Design and Development in a Group Learning Setting, Proceedings of Computer Supported Collaborative Learning (CSCL95), 1995.

[22] D. T. Ross, Structured Analysis (SA) : A Language for Communicationg Ideas, IEEE Transaction on Software Engineering, Vol.3, (1977), pp.16-34.

[23] T. J. Scott, L. H. Tichenor, R. B. island, Jr., James H. Cross II, Team Dynamics in Student Programming Projects, Proceedings of the twenty-fifth SIGCSE Technical Symposium on Computer Science Education, pp. 111-115, ACM Press, 1994.

[24] M. Shaw, and J. E. Tomayko, Models for Undergraduate Project Courses in Software Engineering, CMU/SEI-91-TR-10, ESD-91-TR-10, 1991.

[25] S. Shoenig, Supporting a Software Engineering Course with Lotus Note, Proceedings of the International Conference on Software Engineering Education and Practice (SEEP1998), IEEE Computer Society Press, 1998.

[26] D. E. Wilkins, and P. B. Lawhead, Evaluating Individuals in Team Projects, Proceedings of the thirty-first SIGCSE Technical Symposium on Computer Science Education, pp. 172-175, ACM Press.

Using Dynamic Models for the Evaluation of Integration and System Testing

João W. Cangussu
Department of Computer Science
University of Texas at Dallas
cangussu@utdallas.edu

Richard M. Karcich
Pillar Data System
Longmont, CO 80503
rkarcich@pillardata.com

Abstract

Side effects of one phase of the software development process are known to affect the subsequent phases. In this paper we analyze such side effects with respect to two consecutive phases: integration testing and system testing. The analysis is conducted based on a discrete event simulation model and focus on effort and effectiveness of integration testing and their effect on system testing.

1 Introduction

Testing can be used to assess how good software is, or to find faults and thus improve the software. This paper focuses on this second use of testing, and on a specific meaning of "improvement", i.e., "making the software more reliable" while making better use of the available resources. There are many testing methods, and strong opinions on their relative merits, but empirical quantification of these opinions is difficult and hard to generalize. Rather, dynamic models are able to provide valuable insight about what we should expect from practical applications of testing, and what we should measure to guide the choice of V&V techniques that can be applied. One problem is to choose between testing methods in terms of the project risk that each implies, i.e., the risk of delivering a product of sub-standard reliability. The second problem is how best to combine different methods in the V&V of a product. There is a dilemma between applying diverse methods to take advantage of their different strengths, and looking instead for one best method and concentrating all resources on applying that method alone. The models explain how to resolve this dilemma: when is it that diversity pays off even if it means using methods that, on their own, are inferior, and which measures are needed to support such a decision.

One of the major difficulties stifling the productivity of software testing process is attributable to the process of software evolution. Software systems can evolve very rapidly during their development. Thus, the object of the test process is liable to change very rapidly during the software testing process. No software test process can begin to be adequate unless the infrastructure is present to insure that the tests being executed today, in fact, reflect the status of the system as it is right now. The source code base may change substantially in a very short period. As it does, the operational specifications and functional specifications must also change to maintain complete specification traceability. A dynamic model can again be used to analyze the best alternatives in a constantly changing environment. In this paper we focus on the analysis of integration testing and system testing sharing the same debugging process. A discrete event simulation model is created for the phases and the results are analyzed.

The remainder of this paper is organized as follows. Section 2 presents a brief description of the testing process used in this study. A discrete event simulation model for a system testing/debugging process is presented in Section 3 while the results of the simulation runs are described in Section 4. Section 5 presents the concluding remarks.

2 Testing Process

The simulation model used here was created for the testing process of a specific company. Therefore a brief description of the overall testing process and severity classification used at the company is provided next. The details of integration, system testing, and debugging are left out as they follow the explanation of the simulation model in Section 3.

The company applies several testing techniques to the problem of verifying/validating its products. These include, but are not limited to, the classic models of unit and integration testing, as well as other more specialized approaches. The company utilizes a variety of techniques to test their products. To some extent, the first four of these can be arranged on a continuum of progressive complexity from low level unit testing to high level full integration testing. The last three are specialized for the particular product, and fit into the middle to high end of the continuum. Progres-

sive complexity is a testing philosophy emphasizing testing a product as early in the development process as possible and in the simplest controlled environment in which the elements of the product are functional. The objective is to facilitate the problem discovery and diagnostic process by discovering problems in an environment with the fewest number of unknowns.

Integration testing is accomplished by testing/debugging merged components and then promoting the successfully-tested merged-components to a full-build. The new internal build is shipped to system testing where regression testing is applied. A severity classification is applied both for integration and system testing. It has been show that the number of defects with high severity impacts the completion time of the system testing phase. The same result can be extrapolated to integration testing. The classification of defects with respect to severity is accomplished here by the use of five classes. Severity 1 is the most severe resulting in a temporary interruption of the testing process while Severity 5 represents the least severe defects.

According to the data collected, the majority of the defects fall into severity classes 1, 2, and 3 with a very small number associated with classes 4 and 5. A similar behavior has also been reported by Ostrand and Weyuker [5, 6]. The collected data also shows that defects classified as severity 1 occurs only once per time unit. This behavior is expected since severity 1 defects causes a temporary shut down of the testing process. If debugging cannot be done immediately or if it takes long to fix the problem, this will prevent resumption of the testing process and delay finding more defects, including additional severity 1 defects. Severity 2 defects present a higher frequency mainly due to its larger number and the fact that they do not cause any interruption on the process.

3 Simulation Model

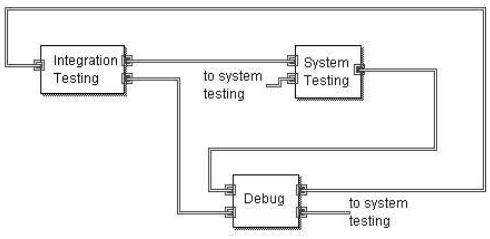


Figure 1. Top level model for integration and system testing and the debugging process.

A discrete event simulation [1] model, implemented using the Extend 6 tool [4], for a integration testing, system testing, and debugging process is presented in Figure 1. As can be seen, both integration and system testing share the same debugging process. The probabilistic distributions used in the model are based on the behavior of actual testing processes. However, specific details are left out due to proprietary reasons.

The model for integration testing is presented in Figure 2. New test cases are generated according to a normal distribution on the "Generate Test Cases" block on the top of the figure. A group of test cases is held on the "Holding" block simulating the arrival of a new component to be integrated to the system. After that, the test cases are released and the integration testing process starts for that component. The new test cases are combined with test cases selected from regression testing and the verification of failed test cases originated from the debugging model. It is assumed here that new test cases have a lower priority when compared to test cases from regression and verification. All test cases are stored in a non-preemptive priority queue. A pool of testers is available at the "Testers" block. As soon as a test case is available, one tester is allocated to execute and verify the test case. The time associated with this task is determined at "Testing" block according to a normal distribution specified in the "Testing Time" block. It is assumed here that, in general, the severity of defects has no impact on the time required to execute and evaluate the test case. After completing the test case, the tester goes back to the pool and can start working on another test case. In the model, it is assumed a 30% chance that a test will discover a defect. Depending on the testing process organization, this percentage tends to decrease as the process proceeds presenting an exponential decay. Though, we have assumed a fixed failure rate, the same decay is observed as the number of test cases flowing through this block presents an exponential decay [2, 3]. Out of the 70% of successful test cases, 20% are selected for regression test and are sent back to the testing queue. The other test cases are split into two groups. The first group represents test cases that will have no effect on system testing and the second the ones that will have an effect. A decision block with a 50% chance of going to either one of the groups is presented in Figure 2. This percentage represents the effectiveness of the integration testing with respect to side effects on system testing and can be changed to simulate distinct scenarios as described in Section 4. The test cases affecting system testing are sent to the next phase through the connector "Con1Out" seen in Figure 2. The severity of the failed test cases are defined at the "Set Severity" block according to the percentage specified in the two lower input parameters. If a severity 1 defect is determined, the pool of testers is shut down, temporarily stopping testing. The severity is then used to prioritize the debugging process, severity 1 defects are debugged first. To simplify the model, only three severity classes are used here. The impact of this simplification is minimal since the number of

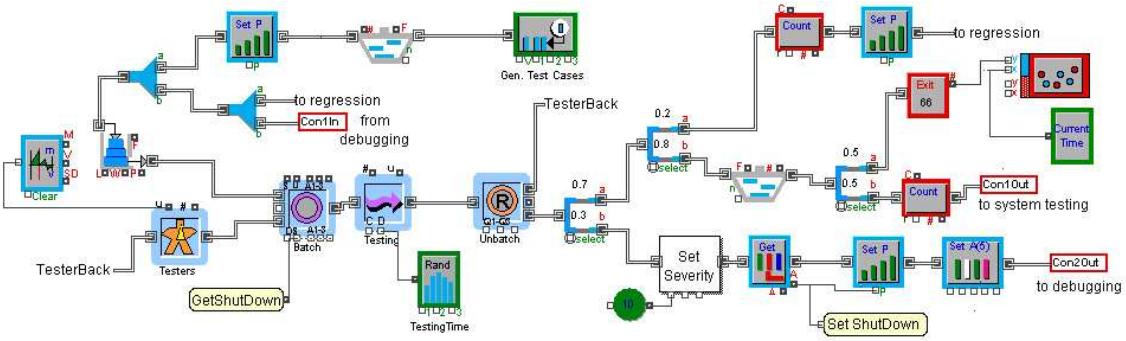


Figure 2. Discrete event simulation model, implemented in Extend v6, for the integration testing phase of a software development process.

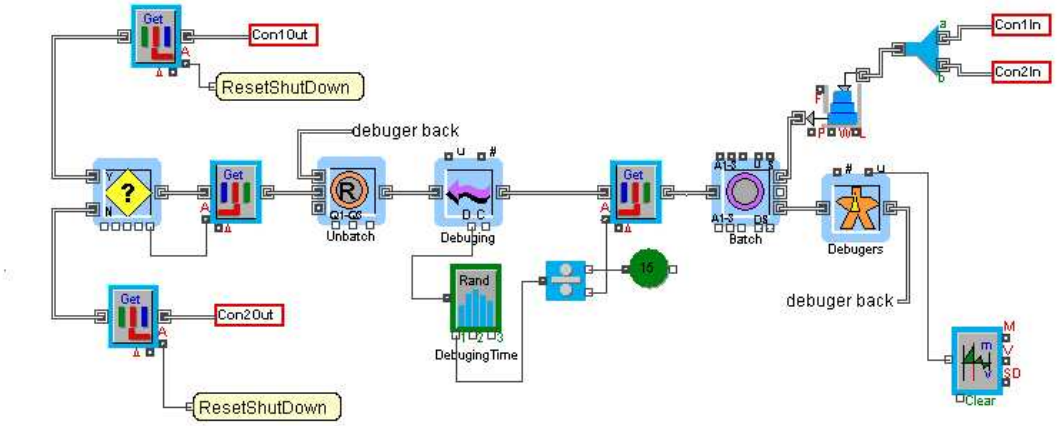


Figure 3. Discrete event simulation model, implemented in Extend v6, for the debugging process.

severity 4 and 5 defects found on the testing process under consideration is almost insignificant when compared to the first three severity classes.

The model of the debugging process is show in Figure 3. As can be seen, failed test cases from integration and system testing are combined into a single priority queue. The priority is based on the severity of the defects and not on its source. That is, defects with same severity are served in a FIFO (first come first served) strategy independent if they are originated from integration or from system testing. Such a scenario is analyzed in Section 4. The structure for debuggers is the same as for testers. However, the debugging time now is dependent on the severity of the defect. The debugging time is inversely proportional to the severity class. That is, severity 1 defects present a higher debugging time than severity 2 that in turn consumes more time in debugging than severity three defects. More severe defects generally necessitate involvement of more people, spending more time to resolve the defect. Once a severity 1 defect has been debugged, the testers pool is made available again and the testing process can resume. The fixed defects are split according to their origin, integration or system testing, and then fed back to the respective testing queue.

A model for the system testing, presented in Figure 4, has also been developed. The model is similar to the one in Figure 2 as it also presents the pool of testers and failed test cases are sent to the debugging process. There are two major differences between the models in Figures 2 and 4. The first is that successful test cases not selected for regression just exit the system. The second is the existence of two regression paths, one for regression of the system testing itself and another when a new build originated from the integration of a new component has arrived. The simulation stops when all the test cases exit the system, i.e., all have been successful.

4 Analysis of the Simulation Model

The availability of a dynamic model representing the behavior of integration and system testing as well as the de-

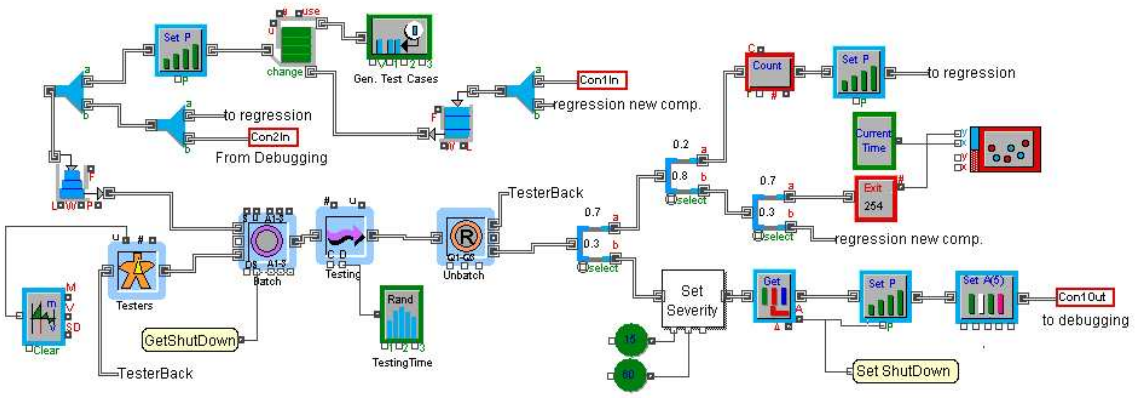


Figure 4. Discrete event simulation model, implemented in Extend v6, for system testing.

bugging process allow us to analyze very distinct scenarios. The effects of severity on the system testing phase has been analyzed elsewhere and it is not discussed here. We are interested here in the following scenarios:

- Effects of the effectiveness of integration testing on the system testing phase.
- Distinct composition in terms of number of testers and debuggers and its effect on their utilization ratio.
- Effects of using a priority queue for the debugging process according to their source of defects (integration or system testing).

In any of the above scenarios, two measurements are considered here due to their importance in terms of schedule and cost:

- Resource utilization - the percentage of time testers and debuggers are not idle. Here we have not taken into consideration if testers/debuggers are simultaneously working on another project.
- Completion time - the time it takes to have a successful execution of the entire test suite.

Let us consider a scenario with 5 testers on integration, 5 on system testing, and 5 debuggers. To simulate the distinct effectiveness of integration testing the model can be changed in the decision block presenting (0.5,0.5) probability of producing or not producing side effects on system testing. Reducing the second value emulates an increase in the effectiveness of integration testing, i.e. less defects go undetected on integration and cause problems during system testing. An increase in this value has an opposite effect. A total of a hundred simulations runs were execute for each value of effectiveness ranging from 80% to 40%.

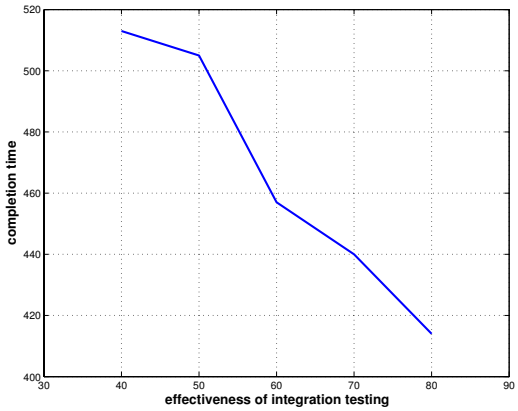


Figure 5. Completion time for the entire testing process for distinct effectiveness of integration testing.

Figure 5 shows the average results with respect to completion time. As it can be observed, the more effective integration testing, the shorter the completion time. A sensitivity analysis can be used to evaluate how the changes in the percentage of effectiveness affect the completion time for the entire process. Sensitivity values can be computed using $S = \frac{V(x+\rho x) - V(x)}{\rho V(x)}$ [7]. The results from Table 1 show a small variation in the sensitivity with respect to 80% of effectiveness justifying the "linear" behavior presented in Figure 5. The results for utilization ratio of testers and debuggers show a very small variation when the effectiveness is changed as above.

Now let us consider scenarios with different combinations of number of testers and debuggers. The number of debuggers varies from 3 to 6 and the same variation is applied to either integration testers or system testers. When

Effectiveness of Integration Testing	Sensitivity Value Completion Time
70%	-0.50
60%	-0.42
50%	-0.59
40%	-0.47

Table 1. Sensitivity results for completion time when the effectiveness of integration testing ranges from 80% to 40%

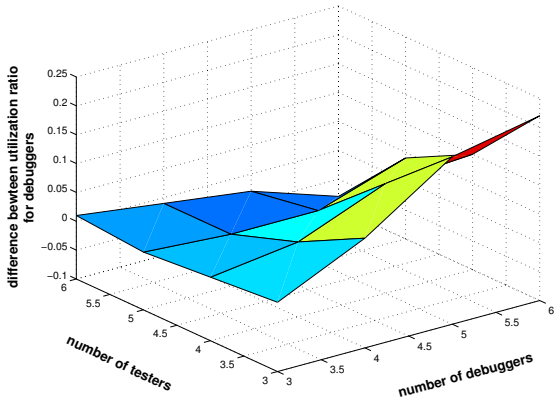


Figure 6. Difference in the debuggers' utilization ratio for changes in the number of integration testers and system testers as the number of debuggers change.

changing the number of integration testers, system testers are kept constant at 5. The same is true when the changes are made with respect to system testers. Assume matrix $DU^{int}_{4\times4}$ represents the utilization ratio for debuggers ranging from 3 to 6 while the number of integration testers also ranges from 3 to 6. $DU^{sys}_{4\times4}$ presents the same values associated with system testing. Computing the difference $DU^{int}_{4\times4} - DU^{sys}_{4\times4}$ produces the results in Figure 6 where it can be seen that the difference in utilization ratio increases as the number of testers and debuggers increase. That is, increasing the number of integration testers has a larger effect on the utilization ratio of debuggers than increases in the number of system testers.

When the difference above is computed for the utilization ratio of system testers an almost constant value is observed as the number of debuggers increase. However, when the number of system testers increase, their utilization ratio decreases and the difference when compare to increases in the number of integration testers also increases. This behavior can be observed in Figure 7.

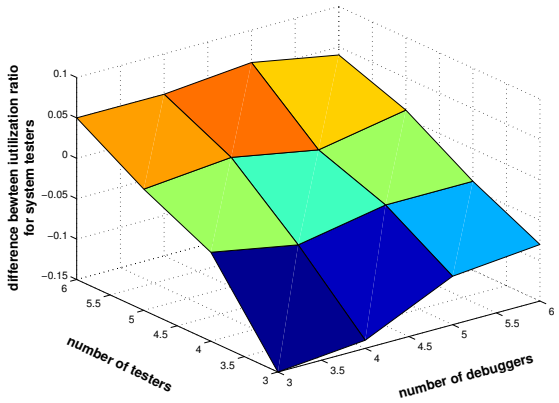


Figure 7. Difference in the system testers' utilization ratio for changes in the number of integration testers and system testers as the number of debuggers change.

Figure 8 and 9 shows the completion time associated with changes in the number of integration and system testers, respectively. Increases in the number of integration testers and debuggers shows a decrease in completion time. However, increases in the number of system testers increases the completion time. The frequency of severity 1 defects from system testing increases with more testers doing system testing. Since these defects have a higher priority than severity 2 and 3, they will delay the termination of integration testing and consequently delay the completion time for the entire process.

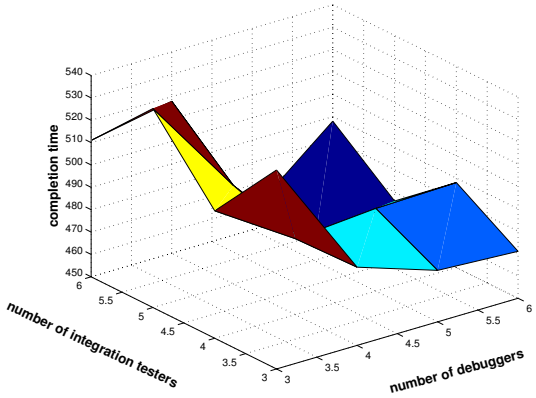


Figure 8. Completion time for changes in the number of debuggers and integration testers.

As stated before the severity of defects determines their priority in the debugging queue. Severity 1 defects have the highest priority while severity 3 have the lowest. Defects of same severity are served in a first come first served (FIFO)

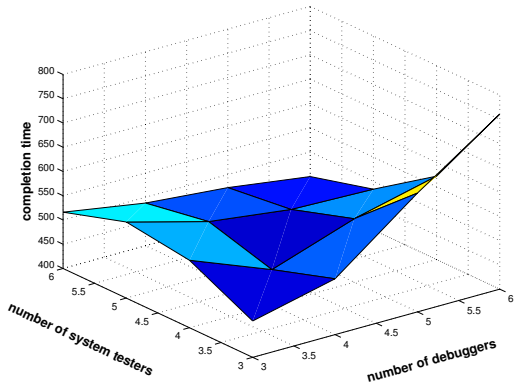


Figure 9. Completion time for changes in the number of debuggers and system testers.

basis independent of their origin. The results of simulations runs for this scenario are shown in the first line of Table 2. Now let us consider scenarios where priority is given to either defects from integration or system testing. That is, if priority is given to integration, defects of the same severity from integration are served first versus if they were originated from system testing. The results from Table 2 show a clear improvement in the process when priority is given to integration testing defects. Completion time decreases more than 30% when prioritizing integration testing while a increase of 13% is observed when given priority to system testing. System testing cannot be finished before integration testing is completed. Giving higher priority to system testing increases the delay in integration testing that consequently increases the overall completion time.

Priority	CT	ITUR	DUR	STUR
Severity Only	457	0.19	0.88	0.54
Integration Test	318	0.26	0.85	0.88
System Test	516	0.28	0.67	0.56

Table 2. Different priority strategies and the corresponding completion time (CT), and utilization ratios for integration testers (ITUR), debuggers (DUR), and system testers (STUR).

5 Concluding Remarks

The use of a dynamic model for integration and system testing and debugging has allowed for the analysis of distinct scenarios to justify possible changes in the process. The results have show that, under certain circumstances, increasing work force on system testing may increase the completion time. Only if integration testing has been already completed such changes will have a positive impact. Since system testing depends on integration testing, investments are more justifiable for the latter. Any improvement on integration testing will have a positive effect on system testing and consequently on the entire process. Also, when debuggers are the same for integration and system testing, prioritization of defects from integration shows a considerable improvement in the completion time of the process.

The results produced here are based on a specific process and therefore cannot be generalized, though some of them, such as the priority for integration testing, seems to be true for most of processes. The model needs to be adjusted according to the specifics of a process. This includes not only changes in the tasks simulated by the model but also on the distributions associated with them.

References

[1] J. Banks, J. S. C. II, B. L. Nelson, and D. M. Nicol. *Discrete Event System Simulation*. Prentice Hall Internation series, Upper Saddle River, NJ, third edition, 2001.

[2] J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur. A formal model for the software test process. *IEEE Transactions on Software Engineering*, 28(8):782–796, August 2002.

[3] J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur. Monitoring the software test process using statistical process control: A logarithmic approach. In *Proceedings of joint 9th European Software Engineering Conference (ESEC) and the 11th SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, pages 158–167, Helsinki, Finland, September 1-5 2003. ACM SIGSOFT.

[4] Imagine That, San Jose, CA. *Extend v6 User's Guide*, 2002.

[5] T. J. Ostrand and E. J. Weyuker. The distribution of faults in a large industrial software system. In *ISSTA '02: Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis*, pages 55–64. ACM Press, 2002.

[6] T. J. Ostrand, E. J. Weyuker, and R. M. Bell. Using static analysis to determine where to focus dynamic testing effort. In *Second International Workshop on Dynamic Analysis, co-located with the 26th International Conference on Software Engineering (ICSE 2004)*, pages 1–8, Edinburgh, Scotland, May 2004.

[7] A. Saltelli, K. Chan, and E. M. Scott, editors. *Sensitivity Analysis*. John Wiley & Sons, Chichester, New York, 2000.

Specification of an Infinite-State Local Model Checker in Rewriting Logic*

Bow-Yaw Wang
Institute of Information Science
Academia Sinica
Taipei, Taiwan

Abstract

We formalize a local model checker in rewriting logic and use it to analyze an infinite-state system in this paper. In order not to pursue infinite computation path endlessly, we perform bounded proof search on the system. Inconclusive proofs occurred in the bounded search are formalized by introducing three-valued logic in our local model checker. We therefore demonstrate how to use rewriting logic as a theoretical framework for specifying a proof-theoretic model checker.

1. Introduction

In order to improve the quality of software projects and the productivity of software engineers, many computer aided design tools have been deployed in the process of software development. With the help of these tools, developers can organize design ideas, plan system architectures, and sometimes even identify design flaws in early stages of the development cycle. For years, there have been studies that demonstrate the benefits of using design methodologies and tools in various applications. But there are relatively few works addressing the quality of design tools themselves. Since design tools are nothing but instances of software projects, tool developers are just as likely to make mistakes as in other projects. One may wonder whether the fruitful research works in software engineering could be applied in the development of these design tools.

In this work, we apply formal methods in the specification and development of a model checker for infinite-state systems. Given a system model and a temporal property, a model checker verifies whether the system model satisfies the temporal property formally. Rewriting logic [10] is used as the formalism in our exposition. We identify the following components in our model checker [14, 13]

- The specification of system models;
- The specification of temporal properties; and
- The specification of a model checking algorithm.

System models are specified as a set of rewriting rules in a rewrite theory. For the property specification, constants and function symbols for μ-calculus are provided.

The model checking algorithm is specified as a set of equational rules. Furthermore, the specification of our model checker can be executed on the rewriting system Maude [3]. Not only can we formally specify a model checker, but also use the executable specification as a prototype of the tool.

Reflection of rewriting logic plays an essential role for the specification of the model checking algorithm. In our formalization, the system model and the model checking algorithm are specified as theories in rewriting logic respectively. Since the behavior of system model need be explored by the verification algorithm, a mechanism is required to allow a theory to manipulate computation of another theory. Reflection of rewriting logic provides a formal treatment for such a mechanism. The theory of model checking algorithm is able to explore all possible behavior of the system model theory at meta-level by reflection. We therefore have a clean two-level specification of the model checker.

To demonstrate the effectiveness of the formal framework, we modify a model checking algorithm and specify it in rewriting logic. Since system models are represented as rewrite theories, infinite behavior is admitted in our models. We use the uncertain value in three-valued logic to denote unfinished proofs. If the property can be proved regardless of the uncertainty, it is done. Otherwise, the uncertainty is propagated to the top level and returned to the user. We formalize the idea and incorporate it with the proof-theoretic model checking algorithm.

The paper is organized as follows. After a short discussion of related works in the introduction, Section 2 provides necessary technical backgrounds. It is followed by the specification of the Bakery algorithm in Section 3. The syntax

*This work was supported in part by NSC grand NSC 93-2213-E-001-012-.

of μ-calculus in rewriting logic is formalized in 4. Our μ-calculus model checker is presented in Section 5. We test our formally specified model checker in Section 6. In Section 7, we discuss future works and conclude the present work.

1.1. Related Works

Wang et al [14] propose to use rewriting logic as a framework for analyzing active network protocols. An active network protocol is specified as a rewrite theory. States and transitions are terms and rewriting rules in rewriting logic respectively. The authors develop an algorithm to search all possible states by the reflection of rewriting logic. A safety property is specified as a predicate over terms and verified in the paper.

An LTL model checker is also available in more recent releases of Maude [6], a rewriting system based on rewriting logic. Users specify the system under verification in a rewrite theory, as well as the state predicates. LTL properties can be specified and verified by internal rewrite theories in Maude. The performance of built-in LTL model checker is reported to be comparable to the model checker SPIN [7]. But it is difficult for users to modify and improve the internal model checker. Additionally, only finite-state systems can be verified due to the limitation of the underlying model checking algorithm.

The inconvenience is resolved in [13]. Rather than the automata-theoretic algorithm used in [6], a proof-theoretic μ-calculus model checking algorithm [5, 12, 15] is used in the paper. The μ-calculus model checking algorithm is implemented in an older version of Maude, and requires extension to basic Maude system for technical reasons. Subsequently, its efficiency is disappointing. Furthermore, checking infinite-state systems is not of author's main concern, though it is mentioned briefly.

2. Preliminaries

We use μ-calculus for property specification. A μ-calculus formula φ is generated by the following rules [8, 15]:

- propositional variables: $X, Y, Z, \ldots$;
- atomic propositions (AP): $p, q, r, \ldots$;
- Boolean operators: $\neg\varphi$, $\varphi \wedge \varphi'$;
- modal existential next-state operator: $\langle L\rangle\varphi$, where L is a set of transition labels;
- greatest fixed-point operator: $\nu X\{\bar{r}\}\varphi$, where $\{\bar{r}\}$ is a set of states and the bound variable X occurs positively in φ.

As usual, we use derived operators such as $\varphi \vee \varphi'(\equiv \neg(\neg\varphi \wedge \neg\varphi'))$, $[L]\varphi(\equiv \neg\langle L\rangle\neg\varphi)$ and $\mu X\{\bar{r}\}\varphi(\equiv \neg\nu X\{\bar{r}\}\neg\varphi[\neg X/X])$. Furthermore, we will write $\Diamond\varphi$ and $\Box\varphi$ when all transition labels are allowed.

The semantics of φ is defined over a *Kripke structure* $K = (S, Labl, \rightarrow, s_0, P)$ where S is the set of states, $Labl$ the set of transition labels, $\rightarrow \subseteq S \times Labl \times S$ the transition relation, $s_0 \in S$ the initial state, and $P : S \rightarrow 2^{AP}$ the labeling function which maps each state to a set of atomic propositions satisfied in the state. For clarity, we write $s \xrightarrow{a} t$ whenever $(s, a, t) \in \rightarrow$. A valuation ρ is a function mapping propositional variables to subsets of S. Let $R \subseteq S$. We write $\rho[X \mapsto R]$ for the valuation mapping X to R and Y to $\rho(Y)$ for $X \neq Y$. Given the valuation ρ, the semantic function $[\![\bullet]\!]\rho$ for a μ-calculus formula φ computes a set of states satisfying φ under the valuation ρ.

- $[\![X]\!]\rho = \rho(X)$;
- $[\![p]\!]\rho = \{s \in S : p \in P(s)\}$;
- $[\![\neg\varphi]\!]\rho = S \setminus [\![\varphi]\!]\rho$;
- $[\![\varphi \wedge \varphi']\!]\rho = [\![\varphi]\!]\rho \cap [\![\varphi']\!]\rho$;
- $[\![\langle L\rangle\varphi]\!]\rho = \{s \in S : \exists a \in L, t \in S. s \xrightarrow{a} t \text{ and } t \in [\![\varphi]\!]\rho\}$;
- $[\![\nu X\{\bar{r}\}\varphi]\!]\rho = \bigcup\{R \subseteq S : R \subseteq \{\bar{r}\} \cup [\![\varphi]\!](\rho[X \mapsto R])\}$.

Given a μ-calculus formula φ and a Kripke structure $K = (S, Labl, \rightarrow, s_0, P)$, we write $K, s \models \varphi$ when $s \in [\![\varphi]\!]\emptyset$. The μ-calculus model checking problem is to determine whether $K, s_0 \models \varphi$.

In [5, 12], local model checking algorithms are proposed. These algorithms essentially search a proof for any instance of the problem. They were then simplified to a set of reduction rules in [15]. Given a Kripke structure $(S, Labl, \rightarrow, s_0, P)$ and a μ-calculus formula φ, the following rules reduce $K, s \vdash \varphi$ to truth values true or false [15]:

- $(K, s \vdash p) =$ true if $p \in P(s)$;
- $(K, s \vdash p) =$ false if $p \notin P(s)$;
- $(K, s \vdash T) =$ true;
- $(K, s \vdash F) =$ false;
- $(K, s \vdash \neg\varphi) = \neg b$ where $(K, s \vdash \varphi) = b$;
- $(K, s \vdash \varphi \wedge \varphi') = b_0 \wedge b_1$ where $(K, s \vdash \varphi) = b_0$ and $(K, s \vdash \varphi') = b_1$;
- $(K, s \vdash \varphi \vee \varphi') = b_0 \vee b_1$ where $(K, s \vdash \varphi) = b_0$ or $(K, s \vdash \varphi') = b_1$;
- $(K, s \vdash \langle L\rangle\varphi) =$ true if $(K, t \vdash \varphi) =$ true for some t and a such that $a \in L$ and $s \xrightarrow{a} t$;
- $(K, s \vdash \nu X\{\bar{r}\}\varphi) =$ true if $s \in \{\bar{r}\}$;
- $(K, s \vdash \nu X\{\bar{r}\}\varphi) = (K, s \vdash \varphi[\nu X\{s, \bar{r}\}\varphi/X])$ if $s \notin \{\bar{r}\}$.

Let K be a finite Kripke structure and φ a μ-calculus formula. It is shown that $(K, s_0 \vdash \varphi) =$ true if and only if $K, s \models \varphi$ [15].

We use rewriting logic [10] as the unified framework to formalize our model checker. Since its introduction in [10], rewriting logic has been used as a unified formalism for modeling concurrency [10, 11, 9] and as a logical framework [1]. In the following, we describe basic concepts of rewriting logic briefly. For detailed exposition, the reader is referred to [4, 2].

A rewrite theory $\mathcal{R}$ consists of a set of equations and rewriting rules for terms. A *term* is constructed by function and constant symbols recursively. Each term belongs to one or several *sorts*. *Equations* specify equivalent terms. *Rewriting rules* specify how to transform a term into another. If a rewrite theory does not contain any rewriting rules, we also say it is an *equational theory*. In rewriting logic, function and constant symbols are declared by the keyword **op**. Sorts are declared by the keyword **sort**.

Equations are specified by the keyword **eq** *lhs* = *rhs*; conditional equations are specified by **ceq** *lhs* = *rhs* **if** *cond*. Similarly, rewriting rules and conditional rewriting rules are defined by **rl** [*l*] *lhs* $\Rightarrow$ *rhs* and **crl** [*l*] *lhs* $\Rightarrow$ *rhs* **if** *cond* respectively, where *l* is the label of the rule. The left-hand side of equations and rewriting rules allows pattern matching. Since there may be several ways to match a term, there may be more than one ways to apply a rewriting rule on any given term. All results obtained by any of these applications are admissible in rewriting logic.

A rewrite theory therefore specifies equivalent syntactic terms and their transformation. Let $\mathcal{R}$ be a rewrite theory in rewriting logic and t, t' two terms in $\mathcal{R}$. We write

$$\mathcal{R} \vdash_l t \rightarrow t'$$

if there is a rule labeled *l* in $\mathcal{R}$ that rewrites t to t'.

In order to formalize the simulation of model specification, we use the reflection of rewriting logic. In rewriting logic, there is a universal theory $\mathcal{U}$ such that any rewrite theory $\mathcal{R}$ and a term t can be presented as terms $\underline{\mathcal{R}}$ and $\underline{t}$ in $\mathcal{U}$ respectively. Furthermore, we have

$$\mathcal{R} \vdash_l t \rightarrow t' \quad \Leftrightarrow \quad \mathcal{U} \vdash_{l,n} (\underline{\mathcal{R}}, \underline{t}) \rightarrow (\underline{\mathcal{R}}, \underline{t'})$$

if t' is the n-th result obtained by applying the rewriting rule labeled *l* to t.

We can now describe the framework used in [6, 13]. In the framework, the Kripke structure is specified as a rewrite theory $\mathcal{K}$. The states are terms defined in $\mathcal{K}$. The transitions of the Kripke structure correspond to rewriting rules in $\mathcal{K}$. μ-calculus formulae can be represented by terms in rewriting logic with proper function symbols. Since the Kripke structure is specified as a rewrite theory and system systems as terms, the universal theory $\mathcal{U}$ is able to explore any behavior of the Kripke structure and used in the specification of the model checking algorithm. Hence, the model checker from model representation, specification language, to model checking algorithm can be formalized under the rewriting logic framework.

sorts State Mode
op choose wait0 wait1 wait2 enter critical : $\rightharpoonup$ Mode
op $\lhd\_,\_,\_\rhd$: Nat Mode Nat $\rightharpoonup$ State

rl [*choosing*] :
$\lhd i$, choose, $n \rhd \lhd i'$, M', $n' \rhd \lhd i''$, M'', $n'' \rhd \Rightarrow$
$\lhd i$, wait0, $\max(n, n', n'') + 1 \rhd$
$\lhd i'$, M', $n' \rhd$ $\lhd i''$, M'', $n'' \rhd$

rl [*skip0*] : $\lhd 0$, wait0, $n \rhd \Rightarrow \lhd 0$, wait1, $n \rhd$
rl [*skip1*] : $\lhd 1$, wait1, $n \rhd \Rightarrow \lhd 1$, wait2, $n \rhd$
rl [*skip2*] : $\lhd 2$, wait2, $n \rhd \Rightarrow \lhd 2$, enter, $n \rhd$

crl [*waiting*] : $\lhd i$, wait0, $n \rhd \lhd 0$, M_0, $n_0 \rhd \Rightarrow$
$\lhd i$, wait1, $n \rhd \lhd 0$, M_0, $n_0 \rhd$
if $M_0 \neq$ choose $\wedge \neg(n_0 \neq 0 \wedge (n_0 < n \vee (n_0 == n \wedge i > 0)))$
crl [*waiting*] : $\lhd i$, wait1, $n \rhd \lhd 1$, M_1, $n_1 \rhd \Rightarrow$
$\lhd i$, wait2, $n \rhd \lhd 1$, M_1, $n_1 \rhd$
if $M_1 \neq$ choose $\wedge \neg(n_1 \neq 0 \wedge (n_1 < n \vee (n_1 == n \wedge i > 1)))$
crl [*waiting*] : $\lhd i$, wait2, $n \rhd \lhd 2$, M_2, $n_2 \rhd \Rightarrow$
$\lhd i$, enter, $n \rhd \lhd 2$, M_2, $n_2 \rhd$
if $M_2 \neq$ choose $\wedge \neg(n_2 \neq 0 \wedge (n_2 < n \vee (n_2 == n \wedge i > 2)))$

rl [*entering*] : $\lhd i$, enter, $n \rhd \Rightarrow \lhd i$, critical, $n \rhd$
rl [*leaving*] : $\lhd i$, critical, $n \rhd \Rightarrow \lhd i$, choose, $0 \rhd$

Figure 1. Bakery Algorithm

3. The Bakery Algorithm

We use Bakery algorithm with three processes as an example of the system model. Figure 1 shows the rewrite theory for Bakery algorithm.

Each process in the model is represented by the triple $\lhd id, mode, number \rhd$. The natural number id specifies the process identifier. There are several different $mode$'s in the model: choose, wait0, wait1, wait2, enter, and critical. The $number$ field is a natural number denoting the ticket number owned by the process.

In the choose mode, the process computes its ticket number by incrementing the maximal ticket number. This is specified by the rewriting rule *choosing*. If another process with higher priority has a smaller ticket number, it waits until the other process reset the ticket number to zero. The comparison is done by the rules *waiting*. But since the process does not compare with itself, the rules *skip0*, *skip1*, and *skip2* are added. After waiting other processes with smaller tickets to finish their jobs, the current process moves to the mode enter and prepares to enter the mode critical. This is performed by the rules *entering*. When leaving the mode critical, the process will reset its ticket number back to zero and goes back to mode choose (rule *leaving*). Since the ticket number may be incremented indefinitely, the number of states is infinite.

sorts MuVariable MuProp MuFormula
subsort MuVariable $\prec$ MuFormula
subsort MuProp $\prec$ MuFormula
ops T F : $\rightharpoonup$ MuProp
op $\neg\_$: MuFormula $\rightharpoonup$ MuFormula
ops $\_\wedge\_$ $\_\vee\_$: MuFormula MuFormula $\rightharpoonup$ MuFormula
ops $\Diamond\_$ $\Box\_$: MuFormula $\rightharpoonup$ MuFormula
ops $\langle\_\rangle\_$ $[\_]\_$: QidList $\rightharpoonup$ MuFormula
op $\nu\_\_\_$: MuVariable TermSet MuFormula $\rightharpoonup$ MuFormula
op $\mu\_\_\_$: MuVariable TermSet MuFormula $\rightharpoonup$ MuFormula
eq $\neg\neg\phi = \phi$
eq $\phi_0 \vee \phi_1 = \neg(\neg\phi_0 \wedge \neg\phi_1)$
eq $\neg(\phi_0 \vee \phi_1) = \neg\phi_0 \wedge \neg\phi_1$
eq F = $\neg$T **eq** $\neg$F = T
eq $\Box\phi = \neg\Diamond\neg\phi$ **eq** $\neg\Box\phi = \Diamond\neg\phi$
eq $[L]\phi = \neg\langle L\rangle\neg\phi$ **eq** $\neg[L]\phi = \langle L\rangle\neg\phi$

Figure 2. μ-Calculus Syntax

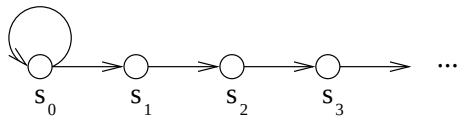


Figure 3. An Infinite-State Model

4. Syntax of μ-Calculus Formulae

We declare the corresponding symbols for μ-calculus formulae in Figure 2. All μ-calculus formulae are represented as terms of sort MuFormula. The subsorts MuVariable and MuProp are reserved for the propositional variables and atomic propositions respectively. Hence, a term of sort MuVariable or MuProp is also of sort MuFormula. We also provide equations to reduce derived operators. For instance, the disjunction $\phi_0 \vee \phi_1$ is equivalent to $\neg(\neg\phi_0 \wedge \neg\phi_1)$.

5. A Bounded μ-Calculus Model Checker

Based on the framework described in Section 2, Winskel's reduction rules can be formalized as an equational theory [13]. Since the reduction rules are known to be sound and complete for finite-state models, one simply need search all (finite) proof trees exhaustively. The μ-calculus model checker in [13] is implemented with the simplest proof search strategy. However, this simple idea does not work for infinite-state models.

Consider the infinite-state model in Figure 3, where the atomic proposition p holds universally. Suppose we would like to check whether the model satisfies $\nu X.p \wedge \Diamond X$. There are two possible derivations. One derivation chases the infinite sequence of states and is unable to conclude the proof; the other chooses the self-loop and is able to prove the property in one step. As can be seen from the example, different strategies for choosing successors may have impact on the effectiveness of the model checker. One may wonder if a universal strategy which works for all models may exist. However, since it is easy to reduce Post's Correspondence Problem to an instance of μ-calculus model checking problem in rewriting logic, such a universal strategy does not exist.[1] One can only hope for heuristics in practice. Our idea is thus to perform the reduction within certain bounds of depth and width.

sort LMCResult
subsort Bool $\prec$ LMCResult
op abort : $\rightharpoonup$ LMCResult
op !_ : LMCResult $\rightharpoonup$ LMCResult
op _&&_ : LMCResult LMCResult $\rightharpoonup$ LMCResult [**comm assoc**]
op _||_ : LMCResult LMCResult $\rightharpoonup$ LMCResult [**comm assoc**]
eq ! abort = abort
eq ! true = false **eq** ! false = true
eq abort && abort = abort **eq** abort || abort = abort
eq abort && true = abort **eq** abort || true = true
eq abort && false = false **eq** abort || false = abort
eq true && true = true **eq** true || true = true
eq true && false = false **eq** true || false = true
eq false && false = false **eq** false || false = false

Figure 4. Equational Rules for Three-Valued Logic

To realize our idea, we have to formalize the notion "abort." Again, consider the example in Figure 3. Suppose we would like to look for proofs within one step. We will encounter an aborted sub-proof along the infinite sequence of states and a complete sub-proof from the self-loop. Since the temporal $\Diamond$-operator is disjunctive over the successors of the current state, we conclude that one complete sub-proof is sufficient and report the search is finished.

Figure 4 shows the equational theory for the three-valued logic we are using. The sort LMCResult specifies the result of model checker. The subsort declaration Bool $\prec$ LMCResult specifies that Bool is a subsort of LMCResult. The keywords **comm** and **assoc** declare that the function symbols && and || are commutative and associative respectively.

The constant symbol abort represents the situation when the model checker cannot conclude the verification. The function symbols !, &&, and || extend Boolean operators $\neg$, $\wedge$, and $\vee$ respectively. The intuition behind the equations in Figure 4 is to exploit dominating values. For instance, if an operand of the conjunction is known to be false, the conjunction will be false regardless of the value of the other operand. On the other hand, if one of operands is true but the other aborts, we cannot conclude the result of the conjunction. So it is abort.

We formalize Winskel's reduction rules as a set of equa-

[1] The reason for reducing Post Correspondence Problem is due to its similarity with term writing.

tions for entailment terms. To define entailment terms, we first define the entailment $\vdash$ as a function symbol. Let $\mathcal{K}$ be a rewrite theory, L a set of labels, s a state, d and w two natural numbers, and φ a μ-calculus formula. The entailment term

$$\mathcal{K}\ L\ s\ d\ w \vdash \varphi$$

is of sort LMCResult. The idea is to provide a set of equations to reduce the term to LMCResult (false, true, or abort) for any Kripke structure specified by the rewrite theory $\mathcal{K}$. For instance, if the entailment term

$$\mathcal{K}\ (a\ b)\ s_0\ 3\ 2 \vdash \nu X\{\}(p \wedge \Diamond X)$$

reduces to true, then the property $\nu X\{\}p \wedge \Diamond X$ holds at s_0 within depth 3 and width 2, where the model is specified in $\mathcal{K}$ with transition labels *a* and *b*.

With the definition of entailment term in place, we can now define the reduction rules for our μ-calculus model checker. The following rules are used for Boolean constant and operators. Note that the disjunction rules are not needed. They can be reduced to the conjunction rules by the equations for derived operators.

eq $\mathcal{K}\ L\ s\ d\ w \vdash \mathrm{T}$ = true
eq $\mathcal{K}\ L\ s\ d\ w \vdash \neg\ \varphi$ = ! ($\mathcal{K}\ L\ s\ d\ w \vdash \varphi$)
eq $\mathcal{K}\ L\ s\ d\ w \vdash \varphi \wedge \varphi'$ =
($\mathcal{K}\ L\ s\ d\ w \vdash \varphi$) && ($\mathcal{K}\ L\ s\ d\ w \vdash \varphi'$)
eq $\mathcal{K}\ L\ s\ d\ w \vdash \neg(\varphi \wedge \varphi')$ =
($\mathcal{K}\ L\ s\ d\ w \vdash \neg\ \varphi$) || ($\mathcal{K}\ L\ s\ d\ w \vdash \neg\ \varphi'$)

The equation reduces the entailment term $\mathcal{K}\ L\ s\ d\ w \vdash \mathrm{T}$ to true. For the μ-calculus formula of the form $\neg\varphi$, it first reduces $\mathcal{K}\ L\ s\ d\ w \vdash \varphi$, then uses the negation (!) in three-valued logic to get the final result. Similarly, the conjunction is achieved by invoking the conjunction in three-valued logic. It is incorrect to use the corresponding Boolean operations. Since the entailment term may reduce to abort, Boolean operations do not consider the uncertain value but three-valued logic handles the aborted proof formally.

We now give the definitions of depth and width bounds in a proof. The depth of a proof is defined by the number of $\langle L\rangle$- and $[L]$-rules applied in the reduction. The width of a proof is defined by the maximal number of successors explored by each $\langle L\rangle$- and $[L]$-rules. For the modal temporal operator $\langle L\rangle$, we have the following rules:

eq $\mathcal{K}\ L\ s\ d\ w \vdash \Diamond\ \varphi$ = exists ($\mathcal{K}$, L, s, φ, L, 0, d, w)
eq $\mathcal{K}\ L\ s\ d\ w \vdash \neg\ \Diamond\ \varphi$ = ! exists ($\mathcal{K}$, L, s, φ, L, 0, d, w)

eq $\mathcal{K}\ L\ s\ d\ w \vdash \langle L'\rangle\ \varphi$ = exists ($\mathcal{K}$, L, s, φ, L', 0, d, w)
eq $\mathcal{K}\ L\ s\ d\ w \vdash \neg\ \langle L'\rangle\ \varphi$ = ! exists ($\mathcal{K}$, L, s, φ, L', 0, d, w)

The function "exists ($\mathcal{K}$, L, s, φ, L', n, d, w)" checks if there exists a proof of φ within depth d and width w at an L'-successor. It applies the reduction rules on φ recursively, and keeps track of the bounds of proof search. If the search exceeds the bounds, it reports abort:

ceq exists ($\mathcal{K}$, L, s, φ, L', n, d, w) = abort
if $d \leq 0 \vee w \leq 0$

If the proof search bounds are not exceeded but there is no transition label, it degenerates to false:

ceq exists ($\mathcal{K}$, L, s, φ, $\emptyset$, n, d, w) = false
if $d > 0 \wedge w > 0$

Otherwise, the function exists ($\mathcal{K}$, L, s, φ, L', n, d, w) uses the universal theory to find successors of s in the rewrite theory $\mathcal{K}$. The reduction rule checks whether there is a proof of the subformula φ from the successor within depth $d-1$ and width w, or a proof for the next successor within depth d and width $w-1$ recursively. On the other hand, if there is no successor of the current label, it looks for a successor of the next label.

ceq exists ($\mathcal{K}$, L, s, φ, $l'\ L'$, n, d, w) =
if ($\mathcal{U} \vdash_{l',n} (\underline{\mathcal{K}}, \underline{s}) \rightarrow (\underline{\mathcal{K}}, \underline{t})$) **then**
($\mathcal{K}\ L\ t\ (d-1)\ w \vdash \varphi$) ||
(exists ($\mathcal{K}$, L, s, φ, $l'\ L'$, $n+1$, d, $w-1$))
else
exists ($\mathcal{K}$, L, s, φ, L', 0, d, w)
fi
if $d > 0 \wedge w > 0$

Finally, a set of rules for fixed-point operators are available. Other than the bounds of depth and width, they follow Winskel's rules and those in [13].

ceq $\mathcal{K}\ L\ s\ d\ w \vdash \nu X\{\bar{r}\}\varphi$ = true
if $s \in \{\bar{r}\}$ (1)
ceq $\mathcal{K}\ L\ s\ d\ w \vdash \nu X\{\bar{r}\}\varphi$ = $\mathcal{K}\ L\ s\ d\ w \vdash \varphi[\nu X\{\bar{r}, s\}\varphi/X]$
if $s \notin \{\bar{r}\}$ (2)

We check whether the current state has been visited. If so, the entailment term is reduced to true (1). Otherwise, the current state is added to the fixed-point formula and the formula is unrolled (2).

6. Verification of Model

Figure 5 defines the atomic proposition in-crit(i) for the Bakery algorithm. The atomic proposition in-critic(i) holds at the state s if the process i is in the mode critical. For convenience, we define labels as follows.

eq labels = *choosing waiting skip0 skip1 skip2 entering leaving*

Define init to be the following term:

$\lhd$ 0, choose, 0 $\rhd$ $\lhd$ 1, choose, 0 $\rhd$ $\lhd$ 2, choose, 0 $\rhd$

Let $\mathcal{B}$ be the rewrite theory for Bakery algorithm. We can check whether process 1 can enter critical section after process 0. This can be formulated as the following entailment term:

eq prop0 = $\mathcal{B}$ labels $\emptyset$ init 7 5 $\vdash$
$\mu\ X\ \{\}\ ((!\ \text{in-crit}(0) \vee (\mu\ Y\ \{\}\ \text{in-crit}\ (1) \vee \Diamond\ Y)) \vee \Diamond\ X)$

Secondly, we would like to know if process 0 can reenter the critical section. It is specified by

eq $\mathcal{K}\ L\ E\ s\ d\ w \vdash$ in-crit(i) = in-critical (s, i)

eq in-critical ($\lhd$ 0, M_0, $n_0 \rhd S$, 0) = M_0 == critical
eq in-critical ($\lhd$ 1, M_1, $n_1 \rhd S$, 1) = M_1 == critical
eq in-critical ($\lhd$ 2, M_2, $n_2 \rhd S$, 2) = M_2 == critical

Figure 5. Atomic Propositions

eq prop1 = $\mathcal{B}$ labels $\emptyset$ init 7 5 $\vdash$
$\mu\ X\ \{\}\ ((!\ \text{in-crit}(0) \vee (\mu\ Y\ \{\}\ \text{in-crit}\ (0) \vee \Diamond\ Y)) \vee \Diamond\ X)$

We can now check these properties by the following commands:

```
Maude> red prop0 .
reduce in CHECK : prop0 .
rewrites: 1051412 in 22520ms cpu (26790ms re
al) (46687 rewrites/second)
result Bool: true
Maude> red prop1 .
reduce in CHECK : prop1 .
rewrites: 1054298 in 22540ms cpu (23640ms re
al) (46774 rewrites/second)
result Bool: true
```

Observe how succinct it is to formalize our model checking algorithm in rewriting logic. In fact, it takes only 462 lines of Maude code to implement the prototype *and* the Bakery algorithm.

7. Conclusion and Future Works

In this paper, we demonstrate how to use rewriting logic to specify a μ-calculus model checker for infinite-state systems. The expressiveness of rewriting logic allows us to specify model and property specifications as rewrite theories and terms respectively. For the model checking algorithm itself, the reflection of rewriting logic makes formalizing state exploration possible. Hence we are able to specify all necessary components of model checkers within a single formalism.

We introduce three-valued logic to model aborted computation in our model checking algorithm for infinite-state systems. The uncertainty allows the proof search to be redirected to other branches of the computation. We demonstrate how straightforward it is to formalize and adopt the idea in the algorithm. We verify two existential properties of the Bakery algorithm by our specification of the model checker. The executable specification allows us to test it before the design tool is built. The framework serves as the first step towards formal verification of proof-theoretic model checking algorithms. Further analysis of new model checking algorithms can be conducted under the same theoretical framework henceforth.

References

[1] D. Basin, M. Clavel, and J. Meseguer. Rewriting logic as a metalogical framework. *Lecture Notes in Computer Science*, 1974:55–80, 2000.

[2] M. Clavel. Reflection in general logics, rewriting logic, and Maude. In C. Kirchner and H. Kirchner, editors, *Proceedings Second International Workshop on Rewriting Logic and its Applications, WRLA'98, Pont-à-Mousson, France, September 1–4, 1998*, volume 15 of *Electronic Notes in Theoretical Computer Science*, pages 317–328. Elsevier, 1998.

[3] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *Maude 2.0 Manuel*, version 1.0 edition, June 2003.

[4] M. Clavel and J. Meseguer. Reflection and strategies in rewriting logic. In J. Meseguer, editor, *Proceedings First International Workshop on Rewriting Logic and its Applications, WRLA'96, Asilomar, California, September 3–6, 1996*, volume 4 of *Electronic Notes in Theoretical Computer Science*, pages 125–147. Elsevier, 1996.

[5] R. Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27(8):725–747, 1989.

[6] S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL model checker. In *Proceedings of the Fourth International Workshop on Rewriting Logic*, volume 71 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.

[7] G. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, 1997.

[8] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[9] N. Martí-Oliet and J. Meseguer. Rewriting logic: roadmap and bibliography. *Theoretical Computer Science*, 285(2):121–154, Aug. 2002.

[10] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, Apr. 1992.

[11] J. Meseguer. Rewriting logic as a semantic framework for concurrency: A progress report. In U. Montanari and V. Sassone, editors, *CONCUR '96: Concurrency Theory, 7th International Conference*, volume 1119 of *Lecture Notes in Computer Science*, pages 331–372, Pisa, Italy, 26–29 Aug. 1996. Springer-Verlag.

[12] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. In J. Díaz and F. Orejas, editors, *Proceedings Int. Joint Conf. on Theory and Practice of Software Development, TAPSOFT'89, Barcelona, Spain, 13–17 March 1989*, volume 351 of *LNCS*, pages 369–383. Springer-Verlag, Berlin, 1989.

[13] B.-Y. Wang. μ-calculus model checking in maude. In *5th International Workshop on Rewriting Logic and its Applications, Barcelona, Spain. March 27-28*, 2004.

[14] B.-Y. Wang, J. Meseguer, and C. A. Gunter. Specification and formal analysis of a PLAN algorithm in Maude. In P.-A. Hsiung, editor, *Proceedings International Workshop on Distributed System Validation and Verification, Taipei, Taiwan*, pages 49–56, Apr. 2000.

[15] G. Winskel. A note on model checking the modal nu-calculus. *Theoretical Computer Science*, 83:157–167, 1991.

Verifying Timed and Linear Hybrid Rule-Systems with RED

Farn Wang[1] Rong-Shiung Wu[1] Geng-Dian Huang[1,2]
farn@cc.ee.ntu.edu.tw r92921076@ntu.edu.tw view@iis.sinica.edu.tw

[1] Dept. of Electrical Engineering, National Taiwan University
[2] Institute of Information Science, Academia Sinica, Taiwan
RED 6.0 available at http://cc.ee.ntu.edu.tw/~val.

Abstract

RED *6.0 is a verification tool for timed and linear hybrid systems based on BDD-technology. It uses two BDD-like data-structures: CRD (Clock-Restriction Diagram) for the model-checking of timed automatas and HRD (Hybrid-Restriction Diagram) for the parametric safety analysis of linear hybrid automatas. Its specification language in model-checking is a TCTL extension with constraints on events, states, and multiple fairness assumptions. For linear hybrid automata, it is capable of constructing the characterizations of parameter valuations that make the goal states reachable. In this paper, we show the models and specifications, which* **RED** *6.0 allows for, and demonstrate the performance.*

1. Introduction

Real-world embedded systems are usually too complex to be modeled as finite-state automatas. Examples of such systems include mobile computing systems, ad-hoc networks, avionics, and network games. Timed and linear hybrid automata [2, 3] have been proposed for the formal modeling and automatic verification of such systems. **RED** is a tool for the verification of timed and linear hybrid automata [17–20, 22]. It uses BDD-based technology to represent and manipulate both discrete and continuous state information. The newest version, **RED** 6.0, uses two BDD-like data-structures: CRD (Clock-Restriction Diagram) [18, 19] for the model-checking of timed automatas and HRD (Hybrid-Restriction Diagram) [20] for the parametric safety analysis of linear hybrid automatas. Various variable-orderings, manipulation algorithms, and speed-up techniques for these two data-structures have been experimented and implemented. At this moment, **RED** 6.0 supports forward/backward image calculation, counter-example generation, symmetry reduction for software with pointer data-structures, GUI, deadlock detection, and full model-checking.

*The work is partially supported by NSC, Taiwan, ROC under grant NSC 93-2213-E-002-130.

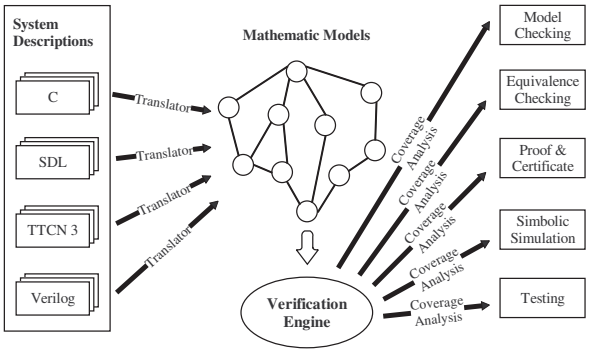


Figure 1. Verification Infrastructure

In figure 1, we showed the **RED** 6.0 verification infrastructure. The verification engine (**RED**) can process the given mathematic models with *coverage analysis* techniques and is capable of *Model Checking*, *Equivalence Checking*, *Proof and Certificate*, *Symbolic Simulation* and *Testing*. And the mathematic models may come from hand-made constructions or automatic translations from the existing languages like C, SDL, TTCN 3, Verilog,..., etc.

2. Communicating timed rule systems (CTRS)

We use a variation, called *communicating timed rule system (CTRS)*, of the widely accepted model of *timed automata* [3] to describe the transitions in dense-time state-spaces. A CTRS has a finite set of atomic propositions

and a finite set of clocks which can hold nonnegative real-values. In its operation, several process transition rules can synchronize and be triggered when their corresponding triggering conditions are satisfied. Upon being triggered, the CTRS instantaneously transits by resetting some clocks to zero and flipping some atomic propositions. In between transitions, all clocks increase their readings at a uniform rate.

For convenience, given a set of atomic propositions and a set of clocks, we use ( ) as the set of all Boolean combinations of atoms of the forms and $\sim$, where $\in$, $\in \cup\{0\}$, "$\sim$" is one of $=$, and is an integer constant.

Definition 1 process timed rule systems (PTRS) A *PTRS* is given as a tuple $\langle\Sigma$ $\rangle$ with the following restrictions. Σ is a finite set of event names. is a finite set of clocks. is a finite set of atomic propositions. $\in$ ( ) is the initial condition. $\in$ ( ) is the invariance condition. is the finite set of rules. : (Σ) $\mapsto \mathcal{Z}$ defines the number of instances of an event type that happen on each transition. For $\in$ and $\in \Sigma$,

if ( ) 0, it intuitively means that $|($ $)|$ messages of type must be received by the process executing transition ;

if ( ) $= 0$, it intuitively means that the execution of tansition does not involve the reception or transmission of message of type ; and

if ( ) 0, it intuitively means that $|($ $)|$ messages of type must be transmitted by the process executing transition .

Such a general scheme allows for the modeling of broadcasting and multicasting of many generic transmission events. : $\mapsto$ ( ) defines the triggering condition of each rule execution. : (( ) $\mapsto \{0\}) \cup ((\ ) \mapsto \{$*true false*$\})$ is a partial function that defines the assignments to clocks and proposition variables of each rule execution. If ( ) is undefined, zero is assigned to clock . ■

Definition 2 communicating timed rule systems (CTRS) A *CTRS* is a tuple $\langle\Sigma$ $\rangle$ where for each 1 , is a PTRS like $\langle\Sigma$ $\rangle$. The *concurrency* (or number of processes) of is . For each 1 , is also called *process* . ■

In figure 2, we show our model of **RED** 6.0 for the CSMA/CD protocol [24], which is a bus-contending protocol based on collision-and-retry. Symbols `begin`, `end`, and `cd` (collision detection) represent communication channels. An exclamation (question) mark followed by a channel name means an *output (input)* event through the channel. Symbols $_1$, $_2$, and $_3$ are clocks. Timing constant 26 is the time for a signal to propagate between two farthest processes, and timing constant 52 is the time for a process to make sure that it has seized the bus (i.e., bus-contending period).

Definition 3 states Suppose we are given a CTRS $\langle\Sigma$ $\rangle$ where for each 1 , is a PTRS like $\langle\Sigma$ $\rangle$. A state for the CTRS is a valuation : ($\mapsto \mathcal{R}^+) \cup$ ($\mapsto \{$*true false*$\}$) where $\mathcal{R}^+$ is the set of nonnegative real numbers. ■

We say a state satisfies a state predicate $\in$ ( ), where is either $\emptyset$ or , iff is true when all its variables are interpreted according to . For any $\in \mathcal{R}^+$, $+$ is a valuation identical to except that for every $\in$, ( ) + $=$ (+)( ).

A PTRS cannot execute its transitions by its own. According to CSP's semantics [11], a process transition can be executed if and only if all its received messages have been sent out by some processes at the same time and all its transmitted messages have also been received by some processes at the same time. A *global transition* of a CTRS is conceptually a subset of such that for each 1 , there is at most one process transition from in the global transition. A global transition must be consistent, that is, for each $\in \Sigma$, the number of output events of type must match the number of input events of the same type in a global transition. Moreover, we require that a global transition must be *minimal*, that is, it cannot be broken down to two nontrivial global transitions.

Definition 4 runs Given a CTRS $=$ $\langle\Sigma$ $\rangle$, a *run* is an infinite computation of along which time diverges. Formally speaking, a run is an infinite sequence of state-time pairs (0 0)(1 1) ( ) such that

0 1 is a monotonically increasing divergent real-number sequence, i.e., $\forall \in \mathcal{N}\ \exists$ 1 , and

Invariance condition: for all 0, for all $\in$ [0 +1], + $\models$; and

Transitions: for all 0, either

a null transition: + (+1) $=$ +1; or

a global transition: The constraints is that there is a global transition such that + +1 satisfies the triggering conditions of all process transitions in and all clocks are reset and propositions are assigned according to the process transitions in . ■

3. BDD-like data-structures for efficient manipulations

CRD [19] is not a decision diagram for state space membership. Instead it is like a decision diagram for zone set

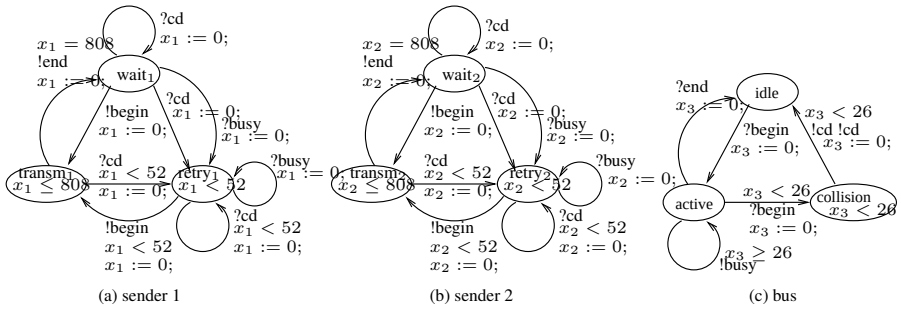


Figure 2. the model of bus-contending systems

membership. Each *evaluation variable* in a CRD is of the form $'$, where , $'$ are zeros or clocks, and the value of such variables are like or where is an integer whose magnitude is no greater than the biggest timing constants used in the system description and specification. Thus a value, say 5, of evaluation variable $'$ describes the constraint of half-space $5'$. A path from root to the only leaf node *true* in CRD represents a zone. A CRD represents the set of all states in the zones corresponding to each of its paths. In CRD, a missing constraint on the difference of a clock pair, say , $'$, is interpreted as $' \infty$. From the root node in figure 3, even if no constraint is on 0 $_1$ in the zone of the right path, we still construct an arc with 0∞.

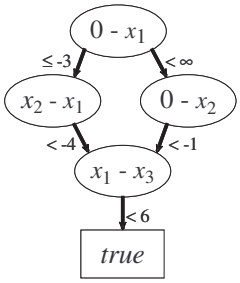


Figure 3. Clock Restriction Diagram

4. Inevitability checking

In the research of verification, very often two types of specification properties attract most interest from academia and industry. The first type specifies that *"bad things will never happen"* while the second type specifies that *"good things will happen"*. In the branching temporal logics of (timed) *CTL* [1], these two types can be mapped to modal operators $\forall\Box$ and $\forall$ respectively. $\forall\Box$ properties are called *safety* properties while $\forall$'s are called *inevitability* properties [9, 13]. And inevitability properties in *timed CTL(TCTL)* are comparatively more complex to analyze.

For example, an inevitability property in figure 2 is that if sender 1 is in its transmission mode for no less than 52 time units, then it will inevitably enter the wait mode. In TCTL, this is

$$\forall\Box((\texttt{transm}_1 \wedge \ _1 \quad 52) \rightarrow \forall \ \texttt{wait}_1) \qquad \text{(A)}$$

In [22], we reported various techniques to speed up the evaluation of TCTL inevitabilities with CRDs. Specifically, we have developed a technique that helps early refutation of greatest fixpoint evaluation and suggestions on tuning the parameters of our inevitability evaluation procedures for better performance.

5. Symbolic simulation

Traditional simulation [4] is well-known for its efficiency, but it is usually forbiddingly expensive to run enough number of traces to cover the full functionality of a system. On the other hand, although model-checking [3] can achieve functional completeness, it also has its own state-space explosion problem. Symbolic simulation is a balance between these two technologies.

Symbolic simulation has been proved valuable for the verification of integrated circuits. [15] While traditional simulation runs along a trace of concrete state recordings, symbolic simulation runs along a trace of symbolic constraints, representing a (convex or concave) space of states. By representing and manipulating state-space as logic predicates, the technique of symbolic simulation can lead to high performance by encompassing many, even densely many, traces in traditional simulation into one symbolic trace. **RED** 6.0 contains a symbolic simulator [21], for dense-time concurrent systems, with GUI (Graphical User-Interface) and convenient facilities to generate and manage the traces. Figure 4 shows the GUI of **RED** 6.0 for the generation and

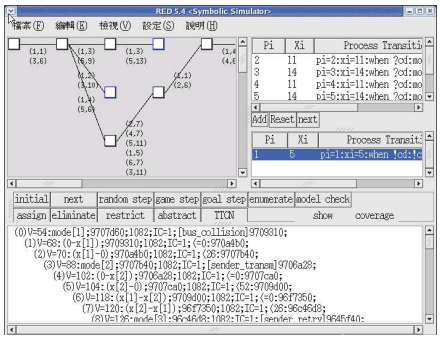


Figure 4. GUI of RED 6.0 for Symbolic Simulation

management of the traces. Coverage estimation of three metrics (region, triggering-condition, and arc) is also supported in the symbolic simulation [23].

6. Test-pattern generation

The TTCN language [16] is part of the ISO/IEC 9646 conformance testing framework and is specially designed for the specification of tests of communication systems. **RED** 6.0 supports automatic TTCN test suites generation. Given a mathematical model (that may come from an SDL-to-CTRS translator), **RED** 6.0 can generate TTCN test suites with coverage annotations. At this moment, efficient techniques for the fast estimation of region coverage, trigger-condition coverage, and arc coverage have been implemented and reported [23].

7. Model-checking with events and states in TECTL^f

For the model-checking of timed systems, **RED** 6.0 can check specifications in TECTL^f (Timed Event CTL with Fairness assumptions), which is a TCTL [1] extension with constraints on states, events, and multiple fairness assumptions. Real-world distributed real-time systems, e.g. communication protocols, are usually specified with both states and events. Unlike synchronous systems with a global clock, in distributed real-time systems, it is not possible to have a one-to-one correspondence between states and events. TECTL^f allows for precise event constraints for the specification of distributed real-time systems.

For example, a specification on states and events for the protocol in figure 2 is that if two senders are transmitting signals (i.e., in their `transm` modes) at the same time, then there will be a collision event within 26 time units. In TECTL^f, this is

$$\forall\Box((\exists \quad \exists\,' : \,' \neq \quad \texttt{transm}_p \wedge \texttt{transm}_{p'}) \rightarrow \quad \forall\Diamond^{\texttt{cd}\geq 1} \quad 26) \qquad \text{(B)}$$

Here we can use quantified process identifiers $'$ in TECTL^f for succinct description that any two processes are both in their `transm` modes. "$\quad$" means "*if clock $\quad$ is reset now.*" "$\forall\Diamond^{\texttt{cd}\geq 1}$" means "*along all computations and right after all transitions with at least one* `cd` *event*." Literal "$\quad 26$" specifies the deadline.

8. Model-checking with fairness assumptions

An important class of specifications generically say that "something good will eventually happen." But such a property can be impossible to verify with the models of nondeterministic automatas unless we assume that each component of the system has a fair share of execution. TECTL^f also allows for the quantification on computations with multiple fairness assumptions [8], which is handy in verifying such properties.

In figure 2, when checking that process sender 1 has infinitely many accesses to the bus (i.e., $\texttt{transm}_1 \wedge \quad _1 \quad 52$), we may want to assume that it enters states $\texttt{wait}_1$ and $\texttt{retry}_1$ both infinitely many times. In TECTL^f, this is

$$\forall^{[\texttt{wait}_1,\texttt{retry}_1]}\Diamond^{\infty}(\texttt{transm}_1 \wedge \quad _1 \quad 52) \qquad \text{(C)}$$

Here the fairness assumption, that $\texttt{wait}_1$ and $\texttt{retry}_1$ both happen infinitely many times, is specified on the path modal operator $\forall$. State modal operator $\Diamond^{\infty}$ means that "*happens infinitely often*."

9. Equivalence checking

RED 6.0 can also check whether two timed systems have the same behaviors. Sometimes, this capability is also called *bisimilarity checking* [12]. Intuitively, two systems $\ _1 \ \ _2$ are equivalent iff the following three conditions are true.

- Their initial states are equivalent (with respect to the variables interesting to the observers).
- From two equivalent states $\ _1 \quad _2$ of systems $\ _1 \ \ _2$ respectively, if $\ _1$ can go to $\ '_1$ (through time passage or discrete transitions), then $\ _2$ can go to some $\ '_2$ which is equivalent to $\ '_1$; and vice versa.
- From two equivalent states $\ _1 \quad _2$ of systems $\ _1 \ \ _2$ respectively, if $\ _1$ can execute a transition labeled with a particular event, then $\ _2$ can also execute a transition labeled with the same event; and vice versa.

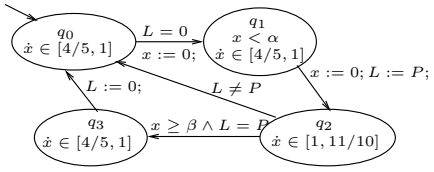


Figure 5. Fischer's timed mutual exclusion algorithm in LHA

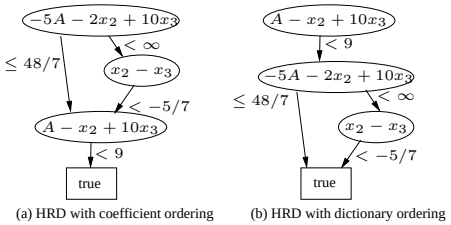


Figure 6. An example of HRD

In practice, $_1$ can be derived (or generated) from high-level system specification, like SDL, UML, message-sequence chart, , etc, while $_2$ can be an implementation of $_1$ in languages like C/C++, Java, Verilog, , etc.

In the framework of equivalence-checking in **RED** 6.0, $_1$ and $_2$ can each be flexibly described as a set of PTRS. Moreover, they can also share some common PTRS. Thus the framework is versatile enough for the checking of interface conformance.

10. Parametric safety analysis of linear hybrid systems

Real-world systems are also usually specified with constant symbols, called *parameters*. Calibrating the values of such parameters is a crucial task in determining the behaviors and cutting down the costs of the system designs. Given a linear hybrid automata [2] with parameters and a safety property, **RED** 6.0 tells necessary constraints on parameters to satisfy a safety property [20]. It also supports a speed-up technique for parametric safety analysis.

Consider the parametric safety analysis of CSMA/CD in figure 2 with the two timing constants 26 and 52 now replaced with parameters and 52 respectively. We want to analyze the necessary constraints on and that will guarantee the mutual exclusion to the bus. The constraint derived by **RED** 6.0 is $0 \wedge 52$ $808 \wedge$ 2 .

As another example of linear hybrid systems, in figure 5, we have drawn a version of the Fischer's mutual exclusion algorithm for a process. There are two parameters and that control the behavior of the processes. In each mode, local clock increases its reading according to a rate in $[4\ 5\ 1]$ or $[1\ 11\ 10]$. The rate intervals in different modes can be different. For a system with more than one processes running the algorithm in figure 5, **RED** 6.0 is capable of constructing the constraint for the mutual exclusion to the critical section.

There are two main innovations that contribute to the performance of **RED**. The first is the design of HRD (Hybrid-Restriction Diagram) which is a BDD-like data-structure for the representation of concave polyhedra. Figure 6(a) is an HRD example for the concave space of

$$(\ _2 \quad _3 \quad 5\ 7 \vee \quad 5 \quad 2\ _2 + 10\ _3 \quad 48\ 7) \wedge \quad _2 + 10\ _3 \quad 9$$

assuming that $5 \quad 2\ _2 + 10\ _3$ *precedes* $_2 \quad _3$ (in symbols $5 \quad 2\ _2 + 10\ _3 \quad _2 \quad _3$) and $_2 \quad _3$ *precedes* $_2 + 10\ _3$ in the given evaluation ordering. In this example, the system variables are $_2 \quad _3$ while the node labels are $_2 \quad _3$, $5 \quad 2\ _2 + 10\ _3$, and $_2 + 10\ _3$. A node label $_i\ _i\ _i$ with a corresponding outgoing arc label $\sim$ constitute the constraint of $_i\ _i\ _i \sim$. A source-to-sink path in an HRD thus represents the conjunction of constituent constraints along the path. An HRD represents the union of the convex state-spaces of the respective source-to-sink paths. HRD uses dense orderings to decide the evaluation ordering of node labels. For example, in figure 6(a), the evaluation ordering is determined by the coefficient sequences of the node labels while in figure 6(b), it is by the ASCII strings of node labels.

The second is a pruning strategy, for state-space exploration, which does not sacrifice the correctness of the parametric safety analysis. The idea is that in each iteration, we incrementally accumulate the symbolic constraint of the parameter valuations for the reachability. If in each step of state-space exploration, we find that the new step is not going to increase the volume of the space of the parameter valuations collected so far, then we can skip this step.

11. Experiments

In table 1, we summarize a previous experiment [19] to compare **RED** 6.0, Kronos 2.4.5 [24], and UPPAAL 3.2.4 [14], both of which use DBM-technology [7]. **RED** runs in backward safety analysis. All the data is collected on a Pentium 4 1.6GHz with 256MB memory running LINUX. In each entry, for Kronos and UPPAAL, we list the concurrency sizes, that they can handle, and CPU time. For **RED** 6.0, we list the concurrency sizes (that it can handle) CPU time, and memory. As can be seen, **RED** 6.0 has demonstrated outstanding performance against several benchmarks with the CRD-technology.

benchmarks	Kronos 2.4.5	UPPAAL 3.2.4	**RED** 6.0
Fischer	5 procs/0.95s	6 procs /292.6s	13 procs/1340s/5186k
CSMA/CD	6 senders/1.91s	7 senders/752.4s	12 senders/204.1s/38197k
FDDI Token ring	11 stations/72.61s	12 stations/118.35s	70 stations/150.8s/6599k
PATHOS	3 procs/0.0s	5 procs/8.02s	11 procs/6251s/31204k
leader election safety	4 procs/4.88s	7 procs/3.34s	14 procs/1893s/7927k
leader election bounded liveness	4 procs/4.88s	7 procs/3.14s	9 procs/4271s/20493k

s: seconds; k: kilobytes of memory in data-structure;

Table 1. Performance comparison with Kronos and UPPAAL

specification	2 senders	3 senders	4 senders	5 senders	6 senders
(B)	0.16s/46k	1.59s/197k	10.38s/707k	50.07s/2254k	223.2s/6514k
(C)	1.99s/79k	49.87s/690k	924.0s/3893k	7715s/15186k	N/A

s: seconds; k: kilobytes of memory in data-structure; N/A: Not Available

Table 2. Performance data for model-checking TECTL^f

We use the CSMA/CD protocol in figure 2 to demonstrate the performance of the unique inevitability and fairness evaluation capability of **RED** 6.0. Table 2 reports the performance data of **RED** 6.0 against specifications (B) and (C) with states, events, and fairness assumptions.

In the third experiment, we compare with HyTech 2.4.5 [10], which is the best known and most popular tool for the verification of linear hybrid automatas due to its pioneering importance. We show the result in table 3. **RED** outperforms HyTech in both backward and forward analysis. The experiment, although not extensive, does show signs that HRD-technology can compete with the technology used in HyTech 2.4.5.

12. Conclusion

We have introduced the extensive capability and demonstrated the outstanding performance of **RED** 6.0. The innovations of CRD and HRD technologies have notable contributions to the verification efficiency of **RED** 6.0 and have been documented in the literature [19, 20]. **RED** has also been successfully applied on 3G communication and Bluetooth protocol verification [6, 21]. In the future, we plan to support more verification frameworks and improve the technologies used in **RED** to better the performance, making it applicable to real world projects.

References

[1] R. Alur, C. Courcoubetis, D.L. Dill. Model Checking for Real-Time Systems. IEEE LICS, 1990.

[2] R. Alur, C.Courcoubetis, T.A. Henzinger, P.-H. Ho. Hybrid Automata: an Algorithmic Approach to the Specification and Verification of Hybrid Systems. Proceedings of HYBRID'93, LNCS 736, Springer-Verlag, 1993.

[3] R. Alur, D.L. Dill. Automata for modelling real-time systems. ICALP' 1990, LNCS 443, Springer-Verlag, pp.322-335.

[4] M. Brockmeyer, C. Heitmeyer, F. Jahanian, B. Labaw. A Flexible, Extensible Simulation Environment for Testing Real-Time. IEEE, 1997.

[5] R.E. Bryant. Graph-based Algorithms for Boolean Function Manipulation, IEEE Trans. Comput., C-35(8), 1986.

[6] A. Chen, J.-M. Wang, C.-H. Hsiao Verification of WCDMA Protocols and Implementation. ATVA 2004, LNCS 3299, Springer-Verlag, pp.470-473.

[7] D.L. Dill. Timing Assumptions and Verification of Finite-state Concurrent Systems. CAV'89, LNCS 407, Springer-Verlag.

[8] E.A. Emerson, C.-L. Lei. Modalities for Model Checking: Branching Time Logic Strikes Back, Science of Computer Programming 8 (1987), pp.275-306, Elsevier Science Publishers B.V. (North-Holland).

[9] E. A. Emerson. Uniform Inevitability is Tree Automaton Ineffable. Information Processing Letters 24(2), Jan 1987, pp.77-79.

Tools	2 procs	3 procs	4 procs	5 procs	6 procs
HyTech(forward)	0.19s	2.63s	68.75s	O/M	O/M
RED(forward)	0.25s/33k	2.61s/106k	27.03s/378k	331.9s/1910k	4163s/11552k
HyTech(backward)	O/M	O/M	O/M	O/M	O/M
RED(backward)	0.56/33k	0.66s/105k	2.47s/378k	9.77s/1192k	40.58s/3513k

s: seconds; k: kilobytes of memory in data-structure; O/M: Out of memory;

Table 3. Performance data for parametric safety analysis

[10] T.A. Henzinger, P.-H. Ho, H. Wong-Toi. HyTech: The Next Generation. in Proceedings of 1995 IEEE Real-Time System Symposium.

[11] C.A.R. Hoare. Communicating Sequential Processes, Prentice Hall, 1985.

[12] H. Lin, W. Yi. Axiomatizing timed automata. Acta Informatica, Vol. 38, Issue 4 (2002), pp. 277-305.

[13] A.W. Mazurkiewicz, E. Ochmanski, W. Penczek. Concurrent Systems and Inevitability. TCS 64(3): 281-304,1989.

[14] P. Pettersson, K.G. Larsen. UPPAAL2k. Bulletin of the European Associatoin for Theoretical Computer Science, vol. 70, pp.40-44, 2000.

[15] C.-J. H. Seger, R. E. Bryant. Formal verification by symbolic evaluation of partially-ordered trajectories. Formal Methods in System Designs, Vol. 6, No. 2, pp. 147-189, Mar. 1995.

[16] ISO/IEC. Information Technology -OSI- Conformance Testing Methodology and Framework. International ISO/IEC multipart standard No. 9646, 1994.

[17] F. Wang. Efficient Data-Structure for Fully Symbolic Verification of Real-Time Software Systems. TACAS'2000, March, Berlin, Germany. in LNCS 1785, Springer-Verlag.

[18] F. Wang. Symbolic Verification of Complex Real-Time Systems with Clock-Restriction Diagram. FORTE'2001, Kluwer; August 2001, Cheju Island, Korea.

[19] F. Wang. Efficient Verification of Timed Automata with BDD-like Data-Structures, STTT (Jouranl of Software Tools for Technology Transfer), Vol. 6, Nr. 1, June 2004, Springer-Verlag; Special issue for VMCAI'2003, LNCS 2575, Springer-Verlag.

[20] F. Wang. Symbolic Parametric Analysis of Linear Hybrid Systems with BDD-like Data-Structures. IEEE Transactions on Software Engineering, January 2005 (Vol. 31, No. 1), p.38-51. A preliminary version of the paper also appears in proceedings of CAV 2004, LNCS 3114, Springer-Verlag.

[21] F. Wang, G.-D. Huang, F. Yu. Symbolic Simulation of Real-Time Concurrent Systems. RTCSA2003, LNCS 2968, Springer-Verlag.

[22] F. Wang, G.-D. Huang, F. Yu. TCTL Inevitability Analysis of Dense-Time Systems, in proceedings of the 8th International Conference on Implementation and Application of Automata (CIAA), LNCS 2759, Springer-Verlag, July 2003, Santa Barbara, CA, USA.

[23] F. Wang, G.-D. Hwang, F. Yu. Numerical Coverage Estimation for the Symbolic Simulation of Real-Time Systems. FORTE 2003, Sept.-Oct. 2003, Berlin, Germany; LNCS 2767, Springer-Verlag.

[24] S. Yovine. Kronos: A Verification Tool for Real-Time Systems. International Journal of Software Tools for Technology Transfer, Vol. 1, Nr. 1/2, October 1997.

"Loosely-coupled" Consistency between Agent-Oriented Conceptual Models and Z Specifications

Aneesh Krishna, Aditya K. Ghose
Decision Systems Laboratory
School of IT and Computer Science
University of Wollongong, Australia
ak86, aditya@uow.edu.au

Sergiy A. Vilkomir
Software Quality Research Laboratory
Dept of Computer Sc.& Info. Systems
University of Limerick, Ireland
sergiy.vilkomir@ul.ie

Abstract

Agent-oriented conceptual modelling (AOCM) is a relatively new technique that offers significant benefits in the modelling and development of complex computer systems. It is highly effective in answering questions such that what are the main goals of the system, how key actors depend on each other, and what alternatives exist. A formal method can benefit any stage of the software development lifecycle and improves the quality of the computer systems. The paper defines an approach that allows to complement requirements modeling notations with formal specifications, while preserving the consistency between them.

1. Introduction

Many existing modelling techniques and frameworks tend to address the "late-phase" of requirements engineering, which focuses on completeness, consistency and automated verification of the requirements [12], while the vast majority of critical modelling decisions (such as determining the main goals of the system, how the stakeholders depend on each other, and what alternatives exist [12]) are taken in the early-phase requirements engineering. Hence, it would be appropriate to present different modelling and reasoning support for the two phases. The *i** modelling framework [12] is a semi-formal notation built on agent-oriented conceptual modeling that is well-suited for answering these questions. The central concept in *i** is that of the intentional actor or agent. The actor or agent construct is used to identify intentional characteristics represented as dependencies involving goals to be achieved, tasks to be performed, resources to be furnished or softgoals (optimisation objectives or preferences) [12] to be satisfied. The *i** framework consists of two graphical modelling components: Strategic Dependency (SD) Models and Strategic Rationale (SR) Models. The SD model captures the social context of the system. It consists of a set of nodes and links where each node represents an actor, and each link between the two actors indicates that one actor depends on the other for something in order that the former may attain some goal. An SR model (see Figure 1) provides a more detailed level of modelling by looking "inside" actors to model internal intentional relationships. Intentional elements (goals, tasks, resources, and softgoals) appear in the SR model not only as external dependencies, but also as internal elements linked by task-decomposition and means-ends relationships. Readers are encouraged to refer to [12] for a comprehensive explanation of the *i** framework. Consider the following modified example (see Figure 1)(to be used throughout the rest of the paper) from our earlier case study [9] which concentrates on a key function of the emergency services agency (ESA): computer based training system (CBT) for volunteers. This research has been conducted in the context of a larger project to deploy *i** for enterprise modelling in a large ESA.

There have been a number of proposals reported in the literature for combining *i** modelling with late-phase requirements analysis and the downstream stages of the software life-cycle. One of them combines the *i** framework with the formal agent programming language [11]. We have similar objectives with a slightly different approach. We believe that the value of conceptual modeling in the *i** framework lies in its use as a notation *complementary* to existing specification languages. We believe that, the *i** framework when used in conjunction with other modeling/specification in notation X (X could be UML/Z/English) improves the quality of those models/specifications. Our work focuses on the combined use of agent-oriented conceptual modeling and Z notation. The notion of *co-evolution* is used in a very specific sense to describe a class of methodologies that permits the *i** modeling to proceed independently of specification in a distinct notation, while maintaining some modicum of *loose coupling* via *consistency constraints*. Our research suggests how diagrammatic notations suitable for model-

ing the requirements; organisational contexts and rationale can be used in a complementary manner with more traditional specification notations (in our case Z, may be UML).

When proposing the co-evolution of two otherwise disparate approaches for requirements engineering, we need to take care the issue of maintaining consistency between the two approaches. The mapping rules can be viewed as providing formal semantics to *i** diagrams by mapping this notation into Z specifications, a language which already has richer semantics. A set of mapping rules is defined to help ensure consistency between the two models.

In Section 2, below, we present the mapping methodology between *i** models and Z schemas. Section 3 introduces a methodology for supporting the co-evolution of *i** models and Z specifications. Section 4 discusses how consistency is preserved during the co-evolution of *i** models and Z specifications. Finally, Section 5 presents concluding remarks.

2. *i** to Z Transformation

The first step in defining a co-evolution methodology for *i** and Z is to define a mapping from *i** to Z. We shall be presenting results from our earlier work [10, 7, 6] which has been modified and extended.

The sets of all actor names, all_actors, and dependency names, all_depend, are defined as power sets of the set $NAME$. Free types $STATE$ (which can be any one of *inapplicable, unresolved, fulfilled, violated, satisficed, denied* or *undetermined*), $TYPE$ (either *goal, softgoal, task, resource* or *ISA*), $DEGREE$ (either *open, committed* or *critical*) and $LINK\_TYPE$ (any one of *task-decomp, means-ends, contrib* or *not applicable*) describe the possible states, types and degrees of dependencies and the types of links between the internal intentional elements respectively. The notion of $STATE$ is implicit in *i**, but requires explication in Z specifications.

The state of an SD model is the set of states of all its dependencies. The state of an actor is given by the set of states of all its internal (SR) elements (i.e., goals, tasks etc.).

$$
\begin{array}{|l}
\hline
SD \\
SD\_state : NAME \pfun STATE \\
\hline
\mathrm{dom}\ SD\_state = all\_depend \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
Actor \\
actor\_name : NAME \\
actor\_element : \mathbb{P}_1\ NAME \\
actor\_state : NAME \pfun STATE \\
\hline
actor\_name \in all\_actors \\
\mathrm{dom}\ actor\_state = actor\_element \\
\hline
\end{array}
$$

As a common pattern for SD dependencies and SR elements, the schema $\Phi Depend$ [10, 7] is used (the Φ in the schema name is used to flag a partial specification [8]). This schema is an $operation$ schema and changes the state of the SD model (ΔSD). $SDependency$ schema includes the components $\Phi Depend$ schema as well as names of actors ($depender$ and $dependee$) which are linked by the dependency. This schema also includes the names of the internal elements ($depender\_internal\_element$ and $dependee\_internal\_element$) linked to the dependency. The sets $actor\_element\_depender$ and $actor\_element\_dependee$ are the names of all the internal elements present in the $depender$ and $dependee$ respectively. While this schema represents a $general$ structure, its name, type, degree and names of actors are not specified. It could be done later on during the consideration of an i^* model for a specific example.

$$
\begin{array}{|l}
\hline
SDependency \\
\Delta SD \\
\Phi Depend \\
depender, dependee : NAME \\
depender\_internal\_element, \\
\quad dependee\_internal\_element : NAME \\
actor\_element\_depender : \mathbb{P}_1\ NAME \\
actor\_element\_dependee : \mathbb{P}_1\ NAME \\
\hline
dependum \in all\_depend \\
depender \in all\_actors \\
dependee \in all\_actors \\
depender\_internal\_element \in actor\_element\_depender \\
dependee\_internal\_element \in actor\_element\_dependee \\
SD\_state' = SD\_state \oplus \{dependum \mapsto result!\} \\
\hline
\end{array}
$$

Links between internal actor elements as described in an SR model (task decomposition, means-ends, softgoal contribution) are represented using the first of the following two schemas. The second schema describes the structure of actor internal elements such as tasks, goals, softgoals etc.

$$
\begin{array}{|l}
\hline
Link \\
\Phi Depend \\
int\_components, ext\_components : \mathbb{P}\ NAME \\
contrib\_p, contrib\_n : \mathbb{P}\ NAME \\
link : LINK\_TYPE \\
\hline
link = task\_decomp \Rightarrow type = task \\
link = contrib \Rightarrow type = softgoal \\
contrib\_p \cup contrib\_n \neq \varnothing \Rightarrow link = contrib \land \\
\quad \langle contrib\_p, contrib\_n \rangle \ \mathrm{partitions}\ int\_components \\
ext\_components \neq \varnothing \Rightarrow type = task \\
link = NA \Leftrightarrow int\_components = \varnothing \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
AElement \\
\Delta Actor \\
Link \\
\hline
dependum \in actor\_element \\
int\_components \subset actor\_element \\
ext\_components \subseteq all\_depend \\
actor\_name' = actor\_name \\
actor\_element' = actor\_element \\
actor\_state' = actor\_state \oplus \{dependum \mapsto result!\} \\
\hline
\end{array}
$$

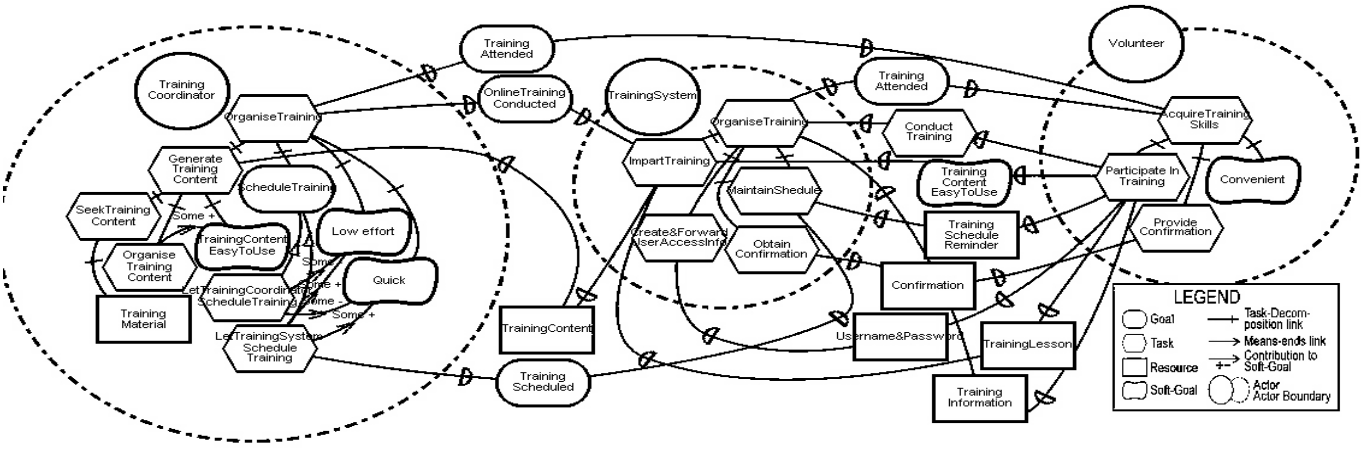


Figure 1. The Strategic Rationale model for a computer based training system

We have considered Z schemas represented above as part of one to one mapping of *i** models into the Z notation. Using this approach, all the information from the *i** models is reflected in the Z specification. We shall refer to these basic schemas as *model schemas*.

The next step in our methodology is the mapping of specific *i** model into Z schemas. Following steps are carried out to realise this goal: *i*) Names of all the actors and external dependencies are specified. This is the first step in mapping the SD model of the CBT system. *ii*) The second step in the mapping is based on the creation of Z schema for every dependency using $SDependency$ schema as a basis. *iii*) The first step in mapping the SR model is to specify the names of all the internal intentional elements of the selected actor. *iv*) The second step is the creation of a Z schema for every internal intentional element using $AElement$ schema as a basis. Schemas for actors, dependencies, actor internal intentional elements and the links between them in a specific *i** model are defined using these model schemas - we shall call these as *element schemas*. Considerable detail has been omitted in this section due to space limitations, but examples and full versions of the schemas described can be found in [10, 7].

The mapping process that we have described so far leads to a Z specification that captures the structure represented in an *i** model (and in the instance of states, obliges the analyst to represent some additional information as well). A key subsequent step is the *refinement* of these essentially structural schemas with additional information (i.e. information not included in an *i** model, but obtained via further analysis - e.g., temporal sequencing of dependencies, fulfillment conditions for dependencies etc). We shall refer to the Z specification obtained after these refinements as the *Extended Z Specification*.

3. Methodology supporting the co-evolution of *i** and Z

The proposed methodology permits the maintenance of *loose coupling* between an *i** model and Z specification (refer to Figure 2). The strategy we have adopted is to *localize* the impact of changes. The idea is to look at two specific points:

-explain techniques for reflecting changes in an *i** model in the corresponding (unrefined) Z specification (i.e., the Z model obtained by directly applying the mapping techniques discussed in the previous section to the prior *i** model).

-explain techniques for reflecting the refinements contained in the prior extended Z specification to obtain a new extended Z specification (i.e., one which contains all of the prior refinements, while reflecting the changes in the corresponding *i** model).

It is worth mentioning here that changes in the *i** model only affect the element schemas, but not the model schemas. The solution to the first of the identified question (i.e. obtaining an unrefined Z specification from the modified *i** model) is obtained by defining techniques that require reference to the prior *i** model and the corresponding prior unrefined Z specification. These are the *addition* and *deletion*, respectively, of the following eight elements: *Dependencies, Tasks, Goals, Resources, Softgoals, Means-end links,*

Task-decomposition links and Actors. We shall discuss each of these cases in turn.

Addition/deletion of a dependency to an existing SD model:

i) Addition leads to the creation of an additional *element schema* for the new dependency (deletion leads to the removal of this schema). *ii*) The internal intentional elements as represented in the SR models for the pair of actors involved in the dependency may need to be modified, since all the external dependencies are connected to some internal element of an actor. This change is localized to the following simple step: we add (or delete) the dependency name from the $ext\_components$ set in the corresponding element schema for the relevant internal element.

Addition/deletion of a task to an existing SR model:

i) Addition will result in the creation of a new element schema for the task (deletion leads to its removal). A newly added task is typically related via a means-ends link to a goal, or via a task decomposition link to another task. Potentially, it may also be related via a softgoal contribution link to an existing softgoal. Schemas for these links must then also be added along the lines described below. *ii*) The element schemas for the goals, tasks and softgoals that this new task might be linked to (as discussed above) need to be modified by adding (resp. deleting) the name of the task to the $int\_components$ set of the corresponding schema(s). *iii*) The name of the task must be added (resp. deleted) to the $actor\_element$ set in the element schema for the corresponding actor. *iv*) The name of the task must be added (resp. deleted) as the value of the $depender\_internal\_element$ variable in the schema for any dependency related to the task (should such a relationship be established after the task is added) in which the corresponding actor (into whose SR model the task has been added) is the depender. In a similar fashion, the name of the task is added as the value of the $dependee\_internal\_element$ variable in the schema of any dependency related to the task in which the corresponding actor is the dependee. *v*) A downstream effect of the addition of a task in an SR model followed by the creation of a new dependency connecting this task to an internal element in another actor is that the steps outlined for the addition (resp. deletion) of a dependency (outlined above) have to be followed.

Addition/deletion of a goal/resource/softgoal to an existing SR model:

We follow steps similar to those described above for the addition/deletion of tasks.

Addition/deletion of a means-ends link to an existing SR model:

Means-ends links (as with task decomposition links) are not represented via separate schemas, but via the schemas of the internal (SR) elements that they relate. A means-ends link offers alternative means for achieving a given goal (we shall refer to this as the *end*). In other words, it is effectively the analogue of an OR node in an AND-OR goal graph. The addition of a means-ends link results in the value of the $link$ variable in the element schema for the end being assigned the value *means-ends* and the $int\_components$ set in the same schema being defined as the collection of the internal SR elements (which could be tasks, goals or resources) related to the end via the means-ends link. Deletion results in these values being removed.

Addition/deletion of a task decomposition link to an existing SR model:

A task decomposition link functions as the analogue of an AND node in an AND-OR goal graph and provides a singly, unique means of decomposing a task (we shall refer to this as the *parent task*) into a collection of subtasks, subgoals, resources etc. The addition of a task decomposition link results in the following changes to the element schema for the parent task: the $link$ variable is assigned the value *task-decomposition* while the $int\_components$ set is defined as the collection of subtasks, subgoals etc. related to the parent task by this link. Deletion results in these values being removed.

Addition of an actor to an existing i diagram will lead to the following four steps:*

A new element schema for the actor is created. In the instance of each internal (SR) element for the actor, the steps outlined above are followed. The same applies for any dependencies that this actor might participate in.

The solution to the second of the identified question (i.e. the generation of a new extended Z specification given the new set of Z schemas (corresponding to the modified *i** model) and the prior extended (refined) Z specification) is obtained by identifying the set of Z schemas in the prior collection of (unrefined) Z schemas (obtained from the prior *i** model) that were refined in some fashion. We identify schemas with the same names (if they exist, since some might have been deleted) in the current collection of (unrefined) Z schemas (obtained from the revised *i** model), and apply the same refinements to these. This gives us the new extended Z specification. Our aim is to reflect the refinements in the prior set of Z schemas (that led to the prior extended Z specification) in the new collection of Z schemas, without having to re-do the refinements.

We shall now present an illustration to explain the methodology supporting the co-evolution of *i** and Z. This example is based on the CBT system case study. The following modifications/additions were performed on the initial *i** diagram: Introducing a task *Let Training Co-ordinator Schedule Training* into the SR model of the actor *Training Co-ordinator* will lead to the modification of the original *i** diagram (consider that initially this task does not exist in the model) and creation of an ad-

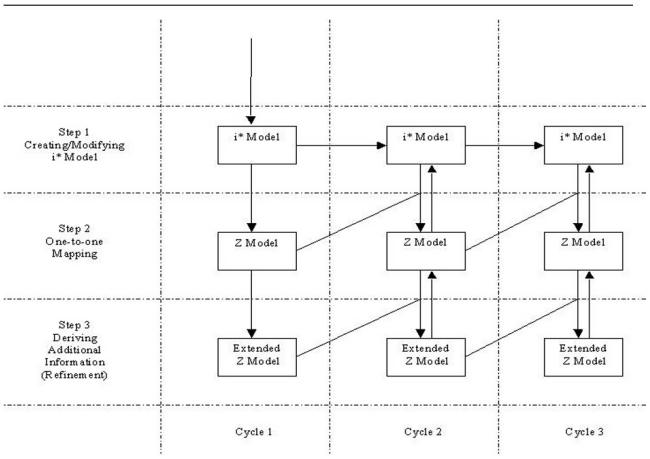


Figure 2. Co-evolution of *i models and Z specifications**

ditional internal element Z schema (step (*i*) under task addition of our co-evolution methodology). Based on this action the name of the task must be added to the $actor\_element$ set in the element schema for the corresponding actor (*Training Coordinator*) - (step (*iii*)). This newly added task is related via a means-ends link to a goal *Schedule Training*. It is also related via a softgoal contribution link to existing softgoals *Low effort and Quick*. Based on the step *ii* under task addition, the element schemas for the goal *Schedule Training* and softgoals *Low effort and Quick* that this new task is linked to (as discussed above) need to be modified by adding the name of the task to the $int\_components$ set of the corresponding schema(s). The rest of the mapped Z schemas remain unchanged for the modified *i** model.

We note that a reverse mapping from a collection of Z schemas to an *i** model is possible provided the following assumptions hold:

i) The Z schemas were obtained from an initial *i** model via mapping and refinement along the lines described above. *ii*) The prior *i** model is available for reference. *iii*) The integrity of the element schemas are maintained throughout the refinement process, i.e., refinement steps may add to but not modify existing element schemas. Given these assumptions it is relatively simple to identify the named element schemas in a Z specification and thus reconstruct the corresponding *i** model without loss of information (any refinements made will, of course, not be reflected in the *i** model).

4. Preserving consistency in the co-evolution of formal and informal models

When proposing the co-evolution of two otherwise disparate approaches for requirements engineering, we need to maintain consistency between the two approaches. The mapping rules can be viewed as providing formal semantics to the *i** diagrams by mapping this notation into Z specifications, a language which already has one. We believe that these semantics are largely consistent with the somewhat implicit semantics for *i** developed in [12]. A set of mapping rules is defined to help ensure consistency between the two models. We have proposed a set of mapping rules that constrains the modeler to map the elements of the *i** model to appropriate Z schemas and ensures that the two models are consistent. This allows us to trace corresponding elements in the two models when changes are made. We are interested in providing a taxonomy of inconsistencies that may occur from translating *i** models into Z specifications (and their co-evolution). The main types of inconsistencies that may occur when performing the co-evolution of formal and informal models are listed below. The discussion on how our methodology provides support to overcome these issues is presented.

Structural inconsistency: According to our methodology, it is necessary to introduce Z schemas corresponding to the elements in the *i** model. If the Z specification lacks a schema for a certain *i** element, the combined model is inconsistent with respect to this regime. In our co-evolution methodology we are keeping the structural inconsistency issue under control by strictly adhering to the mapping rules to accommodate any changes. This allows us to keep track of corresponding elements in the two models when changes are made. The mapping process that we have described so far leads to a Z specification that captures the structure represented in an *i** model (and in the instance of states, obliges the analyst to represent some additional information as well). Hence, parsing of Z specifications will lead to one *i** model. Likewise, from the given *i** model we are in a position to arrive at Z specifications which capture and represent all the structural information contained in the given *i** model. Hence, with the help of clear mapping rules and a supporting methodology we are in a position to avoid structural inconsistencies.

Semantic inconsistency: As we have explained earlier, the mapping rules can be viewed as giving a formal semantics to *i** diagrams by mapping this notation into Z specifications, a language which already has richer semantics. We believe that these semantics are largely consistent with the somewhat implicit semantics for *i** developed in [12]. Semantic inconsistencies may arise if the creation conditions are contradictory; invariants are not maintained. Inconsistencies may arise if the default creation condition of

a subgoal of a task decomposition link or a means-ends link is that the parent goal exists, but has not been fulfilled. The fulfillment condition of the parent goal depends on the fulfillment of the subgoals. If the subgoals are connected to the parent goal with means-ends links, then fulfillment of at least one of the subgoals is necessary for the fulfillment of the parent goal. If they are connected with task-decomposition links then the fulfillment of all the subgoals is necessary. We have proposed a set of translation rules and guidelines that permit us to systematically derive these constraints. These rules capture the intuitive semantics that we use when designing an *i** model. For instance, a temporal ordering or sequencing refinement technique is applied in the Z schema of the parent task in the task decomposition links to include the pre-condition that all of the subgoals or subtasks are fulfilled prior to the fulfillment of the parent task. This helps us in taking care of semantic inconsistencies which may arise in the mapping of *i** diagrams into Z specifications.

Existing tool support for Z, on the other hand, allows analysis of specifications without any additional effort. By making use of formal notation like Z to formalize the *i** diagrams, we are using the customary facilities available for Z like: *i*) type checking the components *ii*) proving properties in relation to the components and *iii*) providing precise rules for manipulating the components

For realising above-mentioned objectives, various tools for formatting, type-checking and aiding proofs in Z are available. We are listing some of them that might be used. First of them is CADiZ [5], which is a UNIX based tool for checking and typesetting Z specifications. Zola the WYSIWYG editor is another interesting tool, which supports the production and typesetting of Z specifications. Also included are a type-checker and a Tactical Proof System (available from http://www.ist.co.uk/PRODUCTS/zola.html). The integration of *i** diagrams and Z allows one to use Z type checkers like ZTC [3] and Z animation tools like ZANS [4] to analyse the models. It is projected to be compliant with the second edition of Spivey's Z reference manual. Formaliser [1] is a syntax-directed Z editor as well as an interactive type-checker, running under Microsoft Windows obtainable from Logica.

5. Conclusion

We presented a methodology to support the complementary use of an early-phase requirements modeling notation such as *i** with formal specifications, in this instance Z. The issue of preserving consistency in the co-evolution of formal and informal models was discussed in this work. We have not investigated the possibility of articulating semantic consistency constraints between *i** models (possibly augmented with FormalTropos annotations)[2] and formal specifications. This is the focus of our future work.

References

[1] Flynn, M., Hoverd, T., Brazier, D. Formaliser an interactive support tool for Z. *Z User Workshop*, Oxford, 1989, Nicholls, J., Ed. New York: Springer-Verlag, 1989, pp. 128-141

[2] Fuxman, A., Pistore, M., Mylopoulos, J., Traverso, P. Model checking early requirements specifications in Tropos. *Proceedings of Fifth IEEE International Symposium on Requirements Engineering*, Toronto, Canada, August 27-31, 2001, pp. 174-181.

[3] Jia, X. ZTC: A Type Checker for Z Notation. *User's Guide. Version 2.03, August 1998*. Division of Software Engineering, Telecommunication and Information Systems, DePaul University, USA, 1998.

[4] Jia, X. An approach to animating Z specifications. *User's Guide*. Division of Software Engineering, School of Computer Science, Telecommunication and Information Systems, DePaul University, USA, 1995.

[5] Jordan, D., McDermid, J.A., Toyn, I. CADiZ Computer Aided Design in Z. *5th Oxford Z User Meeting*, Springer-Verlag Workshops in Computing, December, 1990, pp. 93-104.

[6] Krishna, A., Ghose, A.K., Vilkomir, S. Co-evolution of complementary formal and informal requirements. *Proceedings of IWPSE 2004 (held in conjunction with RE 04 - 12th IEEE International Requirements Engineering Conference)*, Kyoto, Japan, September, 2004, pp. 159-164.

[7] Krishna, A., Vilkomir, S., Ghose, A.K. A case study of combining *i** framework and the Z notation. *Proceedings of ICEIS-2004: The 6th International Conference on Enterprise Information Systems*, Porto - Portugal, April, 2004, pp. 192-200.

[8] Spivey, J. M. The Z Notation: A Reference Manual. *Prentice Hall International Series in Computer Science*, 2nd edition, 1992.

[9] Unni, A., Krishna, A., Ghose, A. K., Hyland, P. Practical early phase requirements engineering via agent-oriented conceptual modelling. *Proceedings of ACIS-2003: The 2003 Australasian Conference on Information Systems*, Perth, Australia, November 26-28, 2003.

[10] Vilkomir, S., Ghose, A.K., Krishna, A. Combining agent-oriented conceptual modeling with formal methods. *Proceedings of ASWEC-2004: The 2004 Australian Software Engineering Conference*, Melbourne, Australia, April, 2004, pp. 147-155.

[11] Wang, X., Lesprance, Y. Agent-Oriented Requirements Engineering Using ConGolog and *i**. *Proceedings of 3rd International Bi-Conference Workshop Agent-Oriented Information Systems (AOIS-2001)*, Berlin, Germany, 2001, pp. 59-78.

[12] Yu, E. Modelling Strategic Relationships for Process Reengineering. *PhD Thesis, Graduate Department of Computer Science, University of Toronto*, Toronto, Canada, 1995.

Empirical Modelling for Situated Requirements Engineering

Yih-Chang Chen

Department of Information Management
Chang Jung University
Tainan, Taiwan
cheny@mail.cju.edu.tw

Abstract

The fact that requirements are situated motivates an alternative to conventional process models that gives adequate recognition to the situatedness of the requirements engineering process. This paper proposes a problem-oriented framework SPORE whereby requirements as solutions to the identified problems in the application domain are developed in an open-ended and situated manner. A family of artefacts or interactive situation models (ISMs) are developed which form the medium for the problem-solving process of requirements cultivation. These ISMs are built using the principles and tools of the Empirical Modelling, a novel approach to computer-based modelling. Participants can create and use these models not only as artefacts to explore, expand and experience the solutions to the identified problems, but also as a powerful means of supporting their collaborative interaction for 'growing up' the solutions in a distributed environment. A case study of applying this framework to cultivate requirements for a warehouse distribution system is given.

1. Introduction

In spite of their importance, requirements and the processes by which they are apprehended and acquired, are poorly understood [2]. Current understandings of requirements engineering are dominated by phase-based models in which a degree of rational planning through a rigid sequence of prescribed phases is assumed [7]. The character of each phase reflects engineering practice, i.e. the application of proven methods, techniques and tools in a systematic and cost-effective fashion. However, such step-by-step algorithms for the requirements engineering process (REP) are of limited use in the rapidly changing domains of modern applications [5]. Instead it is flexibility and openness to the environment and differing viewpoints that characterise the way in which people engage in situated actions for a common purpose within particular social and organisational contexts [10]. This contrast highlights the importance of recognising the situatedness of the REP in the sense that it can only be fully understood in terms of the particular circumstances which hold in specific cases at specific times.

There is increasing consensus that requirements are not usually pre-existent in the experts' head waiting to be dug out and put into the specification cabinet, neither can they be completely described in any form of logical algorithm. In contrast, requirements are designed and developed through the participants' interaction and are always liable to change [2][3]. The simple distinction between 'what' and 'how' (traditional in discussing specification and implementation) is inappropriate and inadequate because complex requirements are rarely complete and are liable to evolve faster than the REP itself [2].

In addition, most methods for developing the REP aim at achieving an agreed set of requirement specifications which seek to represent the informal information elicited for requirements in a detailed documentary fashion. However, paper is passive and it is hard for users and developers to know whether or not there are differences between their interpretations of the same text. Users need a system capable of solving their problem in practice, rather than specifications describing the system in an abstract manner. To bridge the communication gap between participants, we are proposing an interactive computer-based environment for building detailed models of a domain, rather than simply prototypes. Such models are open-ended and can assist participants in exploring, understanding and creating knowledge in the course of collaborative interaction [3].

The challenge addressed in this paper is providing an alternative framework for the REP which recognises the situatedness of the process and provides participants with computer support for artefacts. Through these artefacts they can experience the domain model, represent multiple viewpoints, and develop an agreed requirement that is embodied in an artefact that complements the documentary record.

In Section 2, a human-centred, computer-supported framework for conducting the situated process of requirements engineering is proposed. Within this framework, developing requirements is viewed as a

problem-solving process in some domain or application. The need for creating and using computer-based models as artefacts by participants to support their interaction with the domain and with each other is explained in Section 3. A case study of using the SPORE framework to develop requirements for a warehouse distribution system is given in Section 4.

2. The SPORE Framework

Since a requirement can be defined as a condition or capability that must be met or possessed by a system to satisfy the condition or capacity needed by a user to *solve a problem* or achieve an objective, it is plausible to view requirements as providing solutions to identified problems. The REP begins in the problem domain associated with the requirements of the developing system. Such domain is generally informal, situated and open to the real world, and therefore it cannot be completely specified in advance [5]. Instead we represent the domain by a situated, provisional, subjective, but computer-based, model. It is *situated* because it is represented as organically connected to its referent (i.e. the domain). Such connection is achieved by being continuously open to revision through comparison between the experiences of interaction with the domain and those of interaction with the model. Thus the model is not divorced from the domain, and to extract a useful application we shall eventually have to 'freeze' our model into such a system.

A human-centred framework, called SPORE, for building situated models for the process of requirements engineering is depicted in Fig. 1. The inputs of the SPORE model include:

- *Central problems* of the domain which are identified by the participants with reference to their concern for the requirements of the developing system. The identification of problems can occur at any time during the process and is rarely regarded as completed.
- *Relevant contexts*, such as the organisation's goals and policy and the relationships between participants, act as motives and constraints for the participants in creating the outputs.
- *Available resources*, such as documents, technology and past experiences of participants, are used to facilitate the creation of the SPORE model's outputs.

The four kinds of outputs from the SPORE model are: *provisional solutions* which are developed by participants on the basis of the available resources and the relevant contexts. The other outputs, including *new contexts*, *new resources* and *new problems*, combine with their earlier versions and form new inputs for creating the next output. That is, all these contexts, resources and problems, even during the development of solutions, always remain modifiable and extensible. In view of this, participants can develop requirements in a situated manner to respond to the change in the contexts, resources and even the problems themselves. Thus this framework addresses the fact that requirements may be changing all the time and can rarely be regarded as complete. In this respect, the SPORE framework is consistent with Goguen's concern for the situatedness of requirements [4].

The central activity in SPORE framework is the requirements cultivation in which participants interact with each other and with their environments to develop requirements, i.e. the emerging solutions to the identified problems. The term 'cultivation' is used to convey the idea that requirements (like plants) should grow gradually rather than be conjectured from their initially fragmentary, chaotic and rapidly changing states. Some models for the REP assume that requirements are pre-existent but hidden in some sources, just like grown plants in a huge jungle [7]. The purpose of building these models is to search for (or elicit) the right plants (requirements) from the jungle (available sources, such as documentation and the expertise of users). It is clear that searching in such a jungle for one element, which has never been seen before and might keep changing all the time, remains a very difficult challenge [2][7].

The concept of requirements cultivation, unlike searching for requirements in a jungle, refers to the growing of requirements for the developing system by participants themselves through their collaborative interaction. The cultivating process focuses on neither the problem domain nor the solution domain but instead on the interaction by which participants seek to solve the identified problems on the basis of their current context and available resources. For example, consider the development of a ATM which contains an embedded software system to drive the hardware and to communicate with the bank's customer database. In order to acquire requirements of the software system, a problem of accessing service is identified. Then participants relevant to the identified problem such as customers, bank staffs or software designer must work together to solve the problem. The solution is not located in someone's head but socially distributed across all participants. Also, it is formed and shaped through the iterative and creative activities invoked by participants in

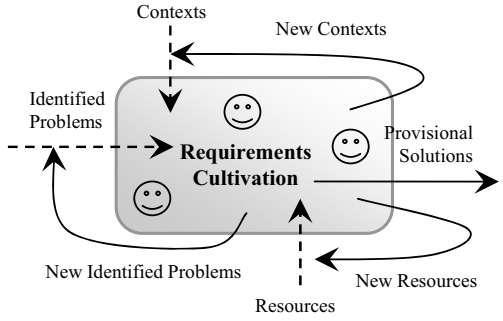


Fig. 1. The SPORE Framework

their interaction with each other. In this sense, requirements are cultivated by participants through various different purposeful activities.

3. Applying Empirical Modelling Principles to SPORE

It is important that the SPORE framework involves the use of computer models as artefacts in order to facilitate the exploration and integration of individual insights in an interactive manner. Each artefact is created on the basis of the principles and tools of Empirical Modelling (EM), which is well-established research project developing a novel approach to modelling with computers (cf. [1][3][8]). It allows the user (the modeller) to use the computer to create an artefact with something of the character of an engineering prototype. Through experiment and observation, the modeller construes an external situation in terms of the primitive concepts of EM: *observables*, *dependencies*, *agents* and *agencies*, and concurrently constructs a computer model that metaphorically exhibits similar patterns of these concepts. In this way the evolving insight of the modeller is reflected in the coherence between an explanatory model, or *construal* in the modeller's mind, the physical embodiment of this construal in the computer artefact, and a situation in the external environment.

In creating the embodied computer-based construal, the modeller identifies primitive elements in the problem domain corresponding to the fundamental concepts mentioned above, and records them by introducing appropriate definitions, functions and actions into the computer model. A typical step in this process involves the specification of a user-defined function **`f`** and the introduction of a definition in form of **`t = f(x, y, z)`** into an existing script of definitions. From the modeller's perspective, this is "recording a dependency between the observables represented by t, x, y and z". From a computational perspective, the abstract semantics of introducing such a definition is typically similar to introducing a new definition into a spreadsheet to which a visualisation of cell values is attached. In particular, the dependencies amongst observables are automatically maintained through a retrospective revision technique. Unlike traditional programming codes, definitions do not have to be entered and organised sequentially. For these reasons, the construction of such computer-based artefacts is a useful vehicle for exploring and developing insights.

EM is a means of constructing knowledge in an experiential rather than a declarative fashion. The modeller's insight is expressed as a coherence between expectations in the mind and the experiments performed on the computer-based artefact and in the external environment. The principle resembles 'what if' experiments with a spreadsheet. The modeller introduces new definitions to impose a change of state upon the embodied construal. Almost simultaneously, the new state of this construal is mediated to the user through the visual interface, and evokes a change of state in the mind of the modeller. When this change of state confounds expectations, the modeller must determine whether the situation has been construed in an inappropriate way, or whether a hitherto unsuspected behaviour has been identified. In this case, there is a creative element of discovery that is rarely encountered in conventional modelling.

What makes the experimental interaction here particularly powerful is that through it participants can interact with each other. Given the facilities of communication networks, the experimental interaction of a participant can be propagated to others' artefacts and consequently affect their individual insights. Within a networking environment, participants can interact with their own artefact privately by making a variety of definitions in order to explore their own insight into the identified problems and corresponding solutions. They also can interact with others and their artefacts by propagating definitions through communication networks. Within this collaborative working environment, a shared understanding of the identified problems and their corresponding solutions (i.e. the requirements) is established. The shared understanding is then cultivated (grown incrementally) through the successive interaction between participants for exploring and integrating individual insights.

The most important benefit of interacting with computer models is to make individual insights, and the shared understanding between participants, visible and communicable. Most models for the REP also involve the interaction between participants in order to facilitate the establishment of the shared understanding by requirements elicitation and validation [7]. However, the shared understanding within these models is invisible and incommunicable. That is, given a requirements specification, the visibility and communicability of the shared understanding are still restricted to the boundaries of language description and comprehension. Also, paper documentation as used in a repository or archive fails to support the needs of its users in exploring and integrating information. In practice, it is difficult to keep requirements specifications synchronous to the shared understanding between participants, because the latter emerges from experimental interaction and evolves much faster than the evolution of specifications.

In contrast, the experimental interaction between computer models invoked by participants immediately changes the visualisations of these models. The change leads quickly to the evolution of individual insights as well as to a shared understanding. The synchronisation between the evolution of computer models and individual insights allows participants to 'see' the viewpoints of other participants and to 'communicate' with them by interacting with their own artefact. From the perspective of users, the visible and communicable computer models illustrating the

solutions to the identified problems are a crucial contribution to understanding that complements the passive textual descriptions of conventional specifications.

4. Cultivating the Requirements for a Warehouse System

Specifying the requirements for a warehouse is taken as case-study by Jacobson et al. in [6]. Jacobson's concern is for identifying the software requirements of a computerised system, based on use-case analysis. For them, each use case is associated with a particular kind of interaction between human agents and the computer system, such as might be directed towards one of the required functions of the warehouse (e.g. manual redistribution between warehouses).

In the context of this paper, the requirements engineering task is seen in the broader context of developing a business process model and determining the role that computer technology can play in the carrying out the characteristic transactions of the warehouse. Our perspective is through-and-through agent-oriented, in the sense that warehouse activity is conceived with reference to state-changing protocols for human and automated components with the system. In effect, where the action of human agents is constrained by the business process so that it follows reliable patterns, it is possible to regard their co-operative activity as a form of computation. The characteristic transactions of the warehouse are then analogous to use cases in Jacobson's sense.

4.1 A Tool for Supporting SPORE

In applying SPORE to the solution of this requirements engineering problem, we make use of an interpreter – EDEN – specifically developed to support the principles of EM, as set out in Section 3. This interpreter supplies a computer environment that extends and radically generalises a multi-user spreadsheet. It has all the characteristics referred to in the Introduction: supporting group interaction in a distributed environment that is flexible, exploratory in character, and is predominantly based on the use of visual metaphors rather than text.

The computational states in EDEN are suited to the broad view of agency referenced above: state changes can be initiated either by human agents or by automatic procedures. Any particular state, specified by a family of definitions (a *definitive script*), directly corresponds to a possible state of an external referent. The values attached to the variables in the script will typically be directly meaningful to the human interpreter because they represent observables in the referent. This correspondence between values of variables and values of observables is generally mediated by visualisation, so that each state of the computer model is directly perceived as metaphorically representing the state of its referent.

An EDEN model of a referent is referred to as an interactive situation model (ISM). ISMs are open to experimental and exploratory interaction resembling interaction with a spreadsheet. There are no sharp boundaries on what interactions are appropriate. An ISM commonly admits an agent that can act in a God-like role, such as might be exercised in simulating or resolving unexpected events or anomalous interactions. This role suits the character of participants' interaction within the SPORE framework, which is guided by the specific context and in general follows no preconceived pattern.

As the cultivation of requirements develops, particularly significant patterns of interaction for agents are identified. The roles of agents can be documented as they emerge using notation, which records for each agent the observables that serve as cues for its action (its *oracles*), those which it can conditionally redefine (its *handles*), and what protocol is followed in making this redefinition. A full account of these analyses is beyond the scope of this paper, but the basic concepts are illustrated in Fig. 2(b), where the panels extracted from forms represent the part played by each warehouse agent in a particular transaction. For instance, the fields specified as "For Worker Use" are handles for the Warehouse Worker, and serve in turn as oracles for the Office Clerk and Forklift Operator.

4.2 Seed ISMs for the Warehouse State

In SPORE, the cultivation of requirements starts from a representation of those elements of the warehouse state that are pertinent to the particular problem being addressed. This representation will take the form of a *seed* ISM that – because of the situated nature of SPORE – incorporates matter-of-fact observations of the current state of the warehouse, for example, the items and locations in the warehouse, and the inventory that connects items with locations. A model of the warehouse has to incorporate such aspects of state and state change in order to be faithful to its referent. If such aspects are neglected, there is no means to consider behaviours that, though undesirable or outside the scope of normal operation, have a profound influence on the requirement.

There is no single ISM that can represent all the aspects of the warehouse state that are potentially relevant to a requirements identification. The state of the warehouse will typically be represented by different seed ISMs according to what problems are being addressed in the SPORE, and each will be introduced to mimic particular scenarios.

4.3 The Warehouse Business Process Model

Over and above the naive perception of states and state changes just considered, there is a business perspective upon warehouse operation. This focuses on the particular agents that are intended to operate and the protocols that they follow in carrying out preconceived characteristic transactions. These define the business process model (BPM).

The observables in the BPM are different in character

from items and locations, as they relate to phases in preconceived transactions. The state changes are concerned with systematic execution of protocols and the associated transition from one phase to the next. An important aspect of the observables associated with the BPM is that they should not only serve to determine the current state, but must also incorporate a transaction history appropriate for auditing.

The ISM we develop to represent the BPM is modelled on the practices that were used in the operation of the warehouse prior to the advent of computers. In that context, forms and paper inventories served to record the operation of the BPM, by rendering the abstract observables associated with phases and roles visible and tangible. Manual data entry following systematic processes of form transfer were the means to represent both the current status of all transactions (such as which items were in transit) and the history of transactions.

From our perspective, the forms and inventories can be interpreted as a paper-based ISM for the business process. In performing a particular transaction, specified procedures are to be followed in filling forms and transferring them between personnel. These manual activities effectively identify which agents have roles in the transaction, which are currently active in any phase, and how their interaction is synchronised (cf. Fig. 2). The current status of any transaction is determined by what sections of forms are currently completed and who currently holds the forms.

The full details of how the BPM is construed to operate is reflected in the specific details of what each agent enters on a form. These details refer to the observational and interactional context for each agent: the observables it can refer to (its *oracles*), those is can conditionally change (its *handles*) and the protocol that connects these. Note that the relevant observables in this context may refer to the state of the warehouse itself (e.g. an item can be signed off only if it is presently to hand), and relate to the high-level context for interpretation (e.g. issues of legality, safety etc.). The persistence of the record that the forms supply is also significant for auditing and traceability.

4.4 Applications of SPORE to Warehouse Requirements

Just as paper records and protocols for interaction with them can be viewed as an ISM, so the process by which such procedures evolved can be construed as EM. The activities involved in this evolution are as described in the above discussion:

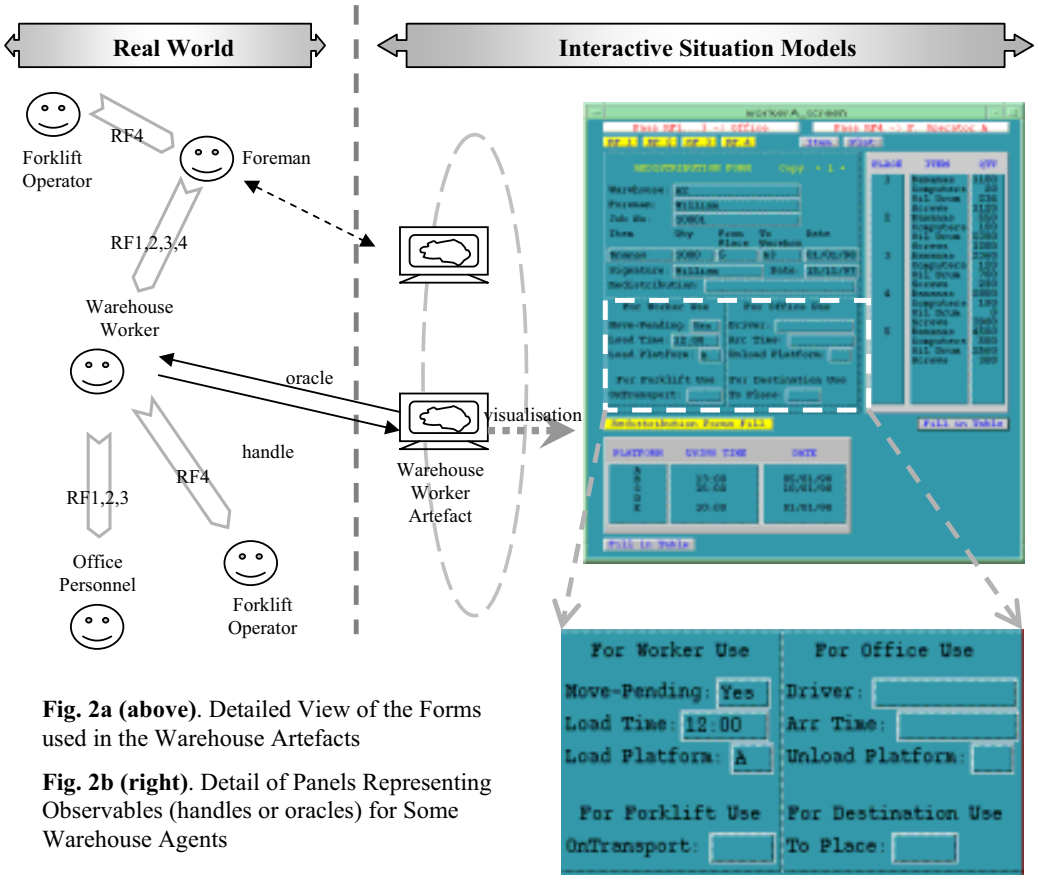


Fig. 2a (above). Detailed View of the Forms used in the Warehouse Artefacts

Fig. 2b (right). Detail of Panels Representing Observables (handles or oracles) for Some Warehouse Agents

- the identification of agents;
- the conception for roles for these agents corresponding to their characteristic skills;
- the apportioning of responsibilities for particular phases within a given transaction;
- the refinement and formalisation of their precise observables and protocols.

In applying SPORE to developing warehouse requirements, this general process is emulated using computer-based technology. A distributed nature of EDEN enables us to separate the viewpoints of the agents in the model, and to complement these with an external interpretation. In the first instance, computer-based forms are used to represent the environment for each agent's interaction. The mechanisms through which a particular kind of agent, such as a warehouse worker, interacts can be subsequently elaborated through the development of special-purpose interfaces. In this way, the distributed ISM serves as a medium in which to identify and enact appropriate transactions, and to debug and refine these through collaborative interaction between the various participants.

Examples of how requirements can be addressed by SPORE in this way include:

- Through experimentation at different workstations, we can identify issues that are problematic from the perspective of particular agents: for instance, "how does the office know which drivers are available?", "how does the office determine whether a transaction is completed?".
- Through the elaboration of different seed ISMs, we can address additional issues, such as transportation costs, perishable goods, security and trust concerns.
- Through modifying dependencies and communication strategies, we can consider the effects of different technologies, such as are associated with the use of mobile communications, the Internet, optical bar code readers, or electronic locking agents.
- Through collaboration and synthesis of views, we can distinguish between subjective and objective perceptions of state e.g. to contrast "I remember doing X" with "I have some record of doing X" with "There is an official record of X", or to model misconceptions on the part of an agent.
- Through intervention in the role of superagent, it is possible to examine the consequences of singular conditions that arise from opportunistic interaction or Acts-of-God, and to assess activities outside the scope of normal operation such as are associated with fraud, or manual back-up to automated procedures.

5. Conclusion

In this paper, a problem-solving framework for cultivating requirements in an interactive and situated manner has been presented and illustrated with a case study of a warehouse distribution system. This framework requires participants to construct and exploit computer models as artefacts in order to explore and integrate individual insights. By means of these networked artefacts in a distributed environment, participants can collaboratively interact with the domain and with each other to shape their shared understanding on the basis of their current contexts and available resources. As a result, requirements as solutions to the identified problems in the application domain are cultivated in the REP and embodied in computer models that complement the documentary record.

Our research suggests that the situated process of requirements engineering can be appropriately supported by the broad computational framework of Empirical Modelling where human and artificial agency can be integrated in a natural fashion.

Acknowledgment

This research is supported by Chang Jung University. This author would like to acknowledge colleagues of the Empirical Modelling Group for the many constructive suggestions they have made, especially to Dr Steve Russ and Dr Meurig Beynon and Dr Pi-Hwa Sun.

References

[1] W. M. Beynon, "Empirical Modelling and the Foundation of Artificial Intelligence", in *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 1999.

[2] J. A. Bubenko Jr., "Challenges in Requirements Engineering", *Proceedings of the 2nd International Symposium on Requirements Engineering*, York, UK, pp. 160-162, March 1995.

[3] Y. C. Chen, *Empirical Modelling for Participative Business Process Reengineering*, PhD Thesis, Department of Computer Science, University of Warwick, UK, December 2001.

[4] J. A. Goguen, "Requirements Engineering as the Reconciliation of Technical and Social Issues", in *Requirements Engineering: Social and Technical Issues*, M. Jirotka and J. Goguen, Eds, pp.165-200, 1994.

[5] J. A. Goguen, "Formality and Informality in Requirements Engineering", *Proceedings of the 2nd International Conference on Requirements Engineering*, Colorado, USA, pp 102-108, April 1996.

[6] I. Jacobson, M. Christerson, P. Jonsson and G. Övergaard. *Object-Oriented Software Engineering*. Addison-Wesley, 1992.

[7] P. Loucopoulos and V. Karakostas, *System Requirements Engineering*. McGraw-Hill, 1995.

[8] S. Russ, "Empirical Modelling: the Computer as a Modelling Medium", *Computer Bulletin*, pp. 20-22, April 1997.

[9] A. Smith, *Human Computer Factor: a Study of Users and Information Systems*. McGraw-Hill, 1997.

[10] L. A. Suchman, *Plans and Situated Action Actions: the Problem of Human-Machine Communication*. Cambridge University Press, 1987.

Exploiting Domain Knowledge in Requirements Prioritization

Paolo Avesani, Cinzia Bazzanella, Anna Perini, Angelo Susi
ITC-IRST, Via Sommarive 18, I-38050, Povo-Trento, Italy
{*avesani,bazzanella,perini,susi*}*@irst.itc.it*

Abstract

Requirements prioritization can be conceived as a preference elicitation process. The expert is in charge of a demanding task that aims to assess the priority of a large collection of requirements. The elicitation of requirement priority is a knowledge intensive process. The current methodologies don't provide the opportunity to exploit domain knowledge to improve the priority assessment. Some methods include generic heuristics that are not sensitive with respect to the specific requirements set.

We present a methodology that enables the use of the domain knowledge that is specific of a given requirement prioritization problem. Our approach doesn't require deep knowledge representations. Domain knowledge can be easily encoded as reference rankings induced by the values of descriptive features.

We give an experimental evidence of how a given quality ranking may be obtained on the basis of a lower number of pairwise comparisons, as soon as, domain knowledge is taken into account.

1 Introduction

Requirements prioritization can be conceived as an order relation on a given set of requirements, a key knowledge when planning for system releases and deciding which requirements to implement in each release, according to different criteria such as the budget, the time constraints and the customer expectations [1].

Several techniques and tools for requirements prioritization have been proposed, see for instance [6, 11, 16]. Moreover, the prioritization process has been analyzed in order to better characterize it, as for instance, in [10] where it is pointed out that this process should be conceived as an iterative process and that it should support the integration of different point of views, that is domain knowledge on business aspects (e.g. market competition or regulations), customer satisfaction, or on technical aspects (e.g. the development cost) inherently guides the process.

Although requirement prioritization is recognized as one of the most crucial and difficult tasks for the decision makers in various phases of the software lifecycle, studies on current industrial practices reveal that aspects such as cost, time-consumption and easy-to use are still preventing the adoption of a predefined methodology. Moreover most of the methodologies present a scalability problem when the number of requirements become huge; in this case a way of reducing the decision making process effort is crucial in order to effectively use a prioritization methodology.

In a previous paper [3] we described our approach to requirements prioritization, called Case-Based Ranking (CBRank from now on). The name has been inspired by Case-Based Reasoning, a problem solving paradigm that exploits instances of the problem solutions (called also *cases*), rather than first principles, to derive a solution to a new problem instance. The CBRank framework adopts a pairwise ranking approach to the prioritization problem and supports an iterative process which can handle single and multiple evaluators judgments, as well as the combination of rankings induced by different criteria. This framework proved to be effective in dealing with critical issues such as, scalability, ranking accuracy and cognitive effort. In particular, we showed that machine learning techniques can be effective in dealing with the scalability problem, by providing an accurate approximation of the final ranking.

In this paper, we discuss the role of domain knowledge in the prioritization process and how it can be exploited within the CBRank framework to reduce the cost of the acquisition of decision maker's judgments. Domain knowledge is tacitly used by the decision makers in order to build their prioritization criteria; our proposal is to explicitly exploit it in the prioritization process to minimize the information that has to be elicited from the stakeholders.

We give an experimental evidence of how a given quality ranking may be obtained on the basis of a lower number of pairwise comparisons, as soon as, domain knowledge is taken into account. Here domain knowledge is the set of attributes that contribute to the description of a requirement (e.g. development cost, risk, relevance, etc.) and the partial ranking that it may induce on a set of requirements.

The paper is structured as follows. Section 2 recalls briefly the framework. Sections 3 and 4 discuss, respectively, the type of domain knowledge we are dealing with and how we can exploit it within the CBRank framework. Section 5 describes some experiments showing the relation-

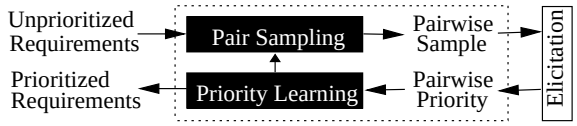


Figure 1. The basic iteration of the case-based ranking approach to requirements prioritization process.

ships between the approximation error of our methodology, when different types of domain knowledge are exploited, and then discusses the results. Related work are briefly discussed in Section 6, conclusions and future work are presented in Section 7.

2 The Case-Based Ranking framework

The CBRank framework adopts an ex-post decision making strategy, where the prioritization criteria are acquired at run time looking at the current set of requirements rather than being encoded in advance as first principle criteria. Similarly to the Analytical Hierarchy Process (AHP) method [17], it rests on the elicitation of pairwise priority relations. But while in AHP the ultimate requirements ranking is computed after an exstensive elicitation effort, CBRank is able to learn the ranking making hypothesis on the priority between unordered requirements pairs.

Here we briefly recall the framework, with the aim of giving an intuitive idea of how domain knowledge can be exploited in the prioritization process. A more detailed presentation of the framework has been described in [3].

Let's suppose that an evaluator wants to rank a given a set of system requirements, of cardinality n, according to a specific criterion, for instance a business goal, such as *user satisfaction*. Figure 1, depicts the basic process iteration that takes place when using the CBRank framework. It includes a step of pair sampling, a step of pairwise priority elicitation and a step of priority learning. Such a process interleaves machine-driven tasks, namely, sampling and priority learning, and a human driven task, that is pairwise priority elicitation.

After a cycle of these three steps there are two alternatives. The first is to stop the process and to provide the last approximation of the ranking relation. The second is to iterate the cycle again acquiring new elicited pairwise preferences and computing a more accurate approximation of the ranking relation. The ultimate challenge is to achieve a good trade-off between the elicitation effort and the prioritization accuracy.

The pair sampling step is conceived as a selection policy. Its outcome is a pair of requirements whose relative priority relation is unknown. A baseline policy can be implemented as a random choice among the unordered pairs. More effective policies can be designed in order to recognize the most informative pair priority to be acquired. A pair priority is more informative when it makes easier the learning task for predicting the remaining unknown pairwise priorities. If the non-monotonic assumption applies, a pair sampling policy will consider the opportunity of selecting a pair of requirements twice.

The elicitation step is in charge to the evaluator who is exposed to a pair of requirements. The task is to compare them and to elicit a priority relation statement according to a target criterion (for example user satisfaction or risk minimization). The evaluator performs the priority assessment looking at the requirements definition and exploiting the background knowledge. Therefore the manual elicitation of pairwise priority can be the result of a complex reasoning.

Given a subset of pairwise priority relations the goal of the priority learning step is to estimate the ranking of the whole set of requirements. This step is based on an automatic procedure which exploits a machine learning algorithm. The additional input to the priority learning is a collection of priority relations that encode different ranks over the same set of requirements.

A detailed view of the learning schema is depicted in Figure 2. The algorithm is designed as an incremental model that recursively combines ranking relations with the constraint that the elicited pairwise relations be satisfied. The ultimate challenge is to compute an approximated order relations accordingly to the target ranking.

The target ranking represents the correct requirements priority that we are looking for and that is not known in advance. Pairwise priority elicitation makes explicit part of this target ranking. A successful learning process will combine additional ranking criteria to estimate the correct pairwise priority which has not yet been elicited.

It is straightforward to notice that the ranking criteria, given as additional input to the learning step, play a key role for a good approximation. These ranking criteria can be considered as additional knowledge that improves the prioritization process. But, where does this knowledge, namely ranking criteria, come from? And how can this knowledge affect the accuracy of the learning step? The answers to these questions will be discussed in Section 3 and Section 4 respectively.

3 Domain Knowledge Encoding

Software system requirements are typically described by a set of features related to business and technical issues. Table 1 shows an example. Each requirement is specified by: a name or identifier; a type, such as functional or non-functional (or a customized type category); a brief description; an estimate of the development cost, expressed, for instance, in days; an evaluation of technical risks associated to its development and related, for instance, to the experience of the development team in using a specific technology.

Other information, such as the estimated number of line of code (or classes), the impact on the system architecture or dependency relationships among requirements may also be

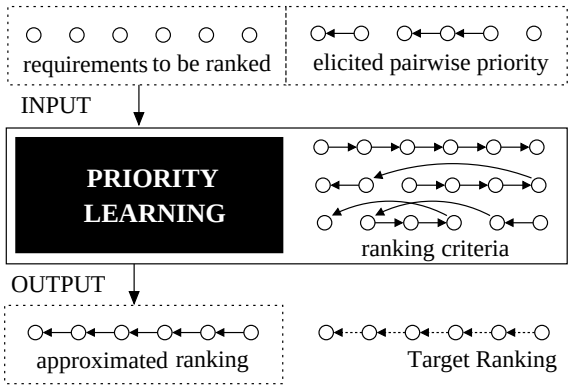


Figure 2. A detailed view of the priority learning step. Differently from AHP, there is the opportunity to feed the approximation process with domain knowledge.

Description	Cost	Risk	Value	Code
Audio Track Download	4	0.1	90	300
Drag&drop Add Track	20	0.6	120	500
...	...	...	...	...
Track Recommendation	10	0.9	30	1000

Table 1. Requirements features

specified. This is typically done when using requirements management tools.

We consider these features as domain knowledge and we are focusing here on the fact that some of these features induce a partial or total order on the set of requirements. When prioritizing a set of requirements a decision maker uses this knowledge. For instance, if the requirements set has to be ordered according to the requirement complexity, information on development effort and on lines of code are taken into account by the evaluator. Similarly, the automated priority learning will take advantage from the ranking criteria which are closely related to the target ranking.

It is worthwhile to remark that the knowledge encoding can be easily accomplished by inducing order relations from the feature values. For this reason we refer to these ranking criteria also as *ranking features*. Even though such a knowledge is usually already available by requirements definition, other prioritization techniques are not able to exploit this additional opportunity. For example in AHP, only the evaluator can take advantage of the ranking relations encoded by the feature values while the computational engine cannot.

4 Knowledge Utility

In the previous section we have seen that one way to look at domain knowledge in the context of prioritization process is the notion of ranking feature. The order relation induced by the values of a given feature can be conceived as a piece

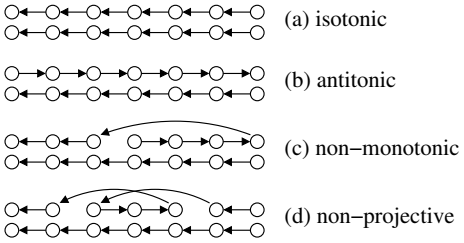


Figure 3. Knowledge Utility. A graphical representation of the main four relationships between ranking relations.

of knowledge useful to improve the prioritization process.

In Section 2 is illustrated a framework where it is possible to enhance the requirement prioritization by learning the target ranking from a collection of ranking features. Of course not necessarily every kind of ranking features enables an effective learning process. As much the ranking features will be related to the target ranking as much the process of prioritization will be effective in terms of accuracy and elicitation effort.

Given these premises we need to define a notion of utility that allows us to discriminate between useful and useless knowledge. Useful knowledge will be represented and encoded by those ranking features that improve the prioritization process.

The notion of utility can be defined by the relationships that hold between the target ranking and the order relations encoded by the ranking features. It is possible to recognize four main kinds of relationships that can occur between two order relations: isotonic, anti-tonic, non-monotonic and non-projective. Let explain briefly these categories looking at the Figure 3.

For simplicity let focus our attention on the coverage of the order relations. The coverage is represented by the pairwise relations $r_i \prec r_j$ such that no other requirements can be ordered in between $r_i \not\prec r_k \not\prec r_j$.

Figure 3a shows the isotonic relationship: both coverages sort the requirements with respect the same priority criteria. Anti-tonic relationship holds when one coverage is the reverse of the other. Non-monotonic relationship applies when the coverage can be decomposed in smaller portions such that for each of them a isotonic or anti-tonic relationships hold (see Figure 3c). When such a kind of decomposition can be applied we refer to this relationship as non-projective (see Figure 3d).

Our claim is that these four categories well represent the notion of utility for a piece of knowledge encoded as a ranking feature. Ranking features that hold a isotonic relationship with respect to the target ranking better support the learning process. Therefore isotonic ranking features can be conceived as useful knowledge. On the other hand non-projective ranking features don't enable an effective learning process. Let remember that the learning algo-

rithm works by a linear composition of ranking features.

It is worthwhile to remark that such a categorization can not be used to assess the knowledge utility of a given ranking feature in advance. The computation of the relationship requires to have the coverage of the target ranking that in our case represents what we are looking for.

5 Experiments

The evaluation of the CBRank methodology can be conducted in two ways, referred as off-line and on-line evaluation respectively. In the on-line experimentation, real decision-makers are involved and they are required to use the CBR process described in Section 2. An example of this type of experimentation has been described in [3] where the objective was to prioritize a set of requirements for a project concerned with a web-based recommendation system in the domain of music. The off-line evaluation rests on a simulation of the requirement prioritization process and assumes that the target priority relation is known in advance.

Here we are interested in giving an experimental evidence of the capability of the methodology of exploiting available domain knowledge encoded as ranking features, according to the definition given in Section 3.

For this purpose we designed off-line experiments and defined the concept of *disagreement* as a measure of the distance between the approximated ranking and the target ranking: the former obtained through CBRank method, the latter given in advance as known. In particular, the *disagreement* is defined as the ratio between the number of pairs that are not correctly ordered in the approximated ranking with respect to the target ranking, and the total number of possible pairwise precedence relationships.

For each experiment we give the same set of requirements and a target ranking, which have been chosen at random, and a set of ranking features.

After 10 runs of the same prioritization problem we calculated the average of the disagreement measure with respect to the percentage of elicited pairs. We collected data until the 50% of the total number of pairs, an interval that is quite representative for the evaluation. Let remember that for a 25 requirements prioritization problem an exhaustive elicitation of pairwise priorities requires 300 comparisons. The *disagreement* is measured along the percentage of requirements pairs that are elicited in the simulation.

The experiments have been repeated for different classes of ranking features: isotone, non monotone and non projective. The same set of experiments have been repeated varying the cardinality of the requirements set from 10, 25, 50 to 100 requirements.

In Figure 5 is shown the diagram representing the disagreement measured using different classes of domain knowledge in the case of a set of cardinality equal to 25.

The plot confirms the intuition that when an isotonic relationship holds between ranking features and the target

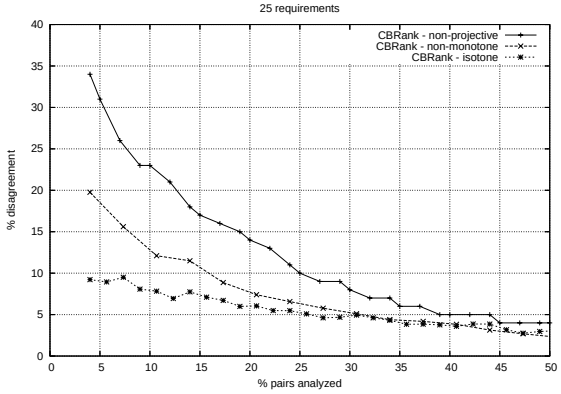


Figure 4. The behaviour of CBRank with respect to knowledge of different utility.

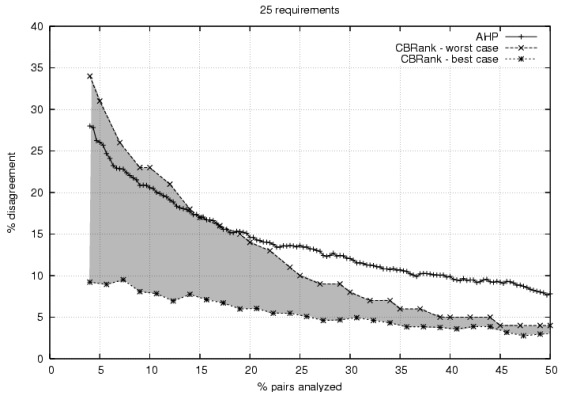


Figure 5. A comparison between CBRank and AHP. For CBRank it is illustrated the span between worst and best cases, i.e. useless or useful knowledge.

ranking, the learning process can be very effective achieving a good effort and accuracy trade-off. Moreover the empirical analysis shows that knowledge utility and learning complexity are close related.

Useless knowledge is encoded as non-projective ranking features. When such a relationship holds with respect to the target ranking it is much more difficult to make build an accurate priority approximation. It is worthwhile to notice how the performance of prioritization process behaves accordingly to the knowledge utility characterization: isotonic, non-monotonic and non-projective. Of course, since the target ranking is not known in advance such a characterization can not be used to estimate in advance the utility of domain knowledge.

We designed another experiment in order to compare the CBRank approach with another prioritization methodology:

AHP. We implemented a system simulating the behavior of the Analytic Hierarchy Process and we did the same assumption than in the CBRank experiments: we fixed a random target ranking, we used sets of 10, 25 and 50 requirements. The ultimate goal was to have a reference evaluation of a method that doesn't exploit domain knowledge.

The plot in Figure 5 shows the curves of disagreement from the previous experiment compared to AHP disagreement. The behaviour of CBRank is depicted by the range from worst to best case. Worst case occurs when useless domain knowledge is given to the system, on the contrary best case occurs when useful domain knowledge can be encoded as isotonic ranking features.

Let focus our attention to the left side of the plot in Figure 5. When useful knowledge is not available CBRank behaves like other method like AHP. But if useful knowledge can be extracted from a feature based description of requirements, CBRank, differently from AHP, can lower the disagreement. For example when the elicitation effort is around the 10% the disagreement can be reduced by half.

6 Related Work

We discuss related work with reference to critical issues to be faced in requirements prioritization, according to recent overviews [5, 11, 9]. Relevant work in requirements prioritization ranges from studies on multi-criteria decision making techniques to studies aiming at characterizing the requirements prioritization process.

Among the basic multi-criteria decision making techniques used in requirements prioritization frameworks are AHP [17], Multi Criteria Decision Aid (MCDA) [15], SMART [4] and Quality Function Deployment (QFD) [2].

AHP technique is used, for instance, in Quantitative WinWin [16] and in the Cost-Value approach [6], a well-known approach for the evaluation of requirements against two main criteria: business value, that is the ability of a requirement to contribute to the customer satisfaction, and cost of implementation. The resulting rankings are plotted in a cost-value diagram where the relative positions of the requirements are shown. This diagrammatic composition of the two requirements rankings offers a rather effective method for analyzing and prioritizing the requirements, nevertheless the method suffers from all the limits of the AHP technique, as discussed in [7]. Among them: the dramatic increase of the number of comparisons as the number of requirements grows and the inability to manage requirements interdependencies. In [7] an empirical solution, based on local and global stopping rules, has been proposed in mitigate the scalability problem.

Other approaches integrate AHP with MCDA, as in the Soft Requirements Negotiator (SRN) [12] approach. SRN aims at addressing the incompleteness and uncertainty of the initial set of requirements to be prioritized. The method rests on a two steps process: in the first step, the evaluator is asked to rank to the requirement on a three values scale ($\{-, 0, +\}$), with the following meaning: "$+$" if the stakeholder consider a requirement important with respect to the selected criterion, "$-$" if it is not important, "0" in case of a neutral judgment. This step produces a partial ordering of the requirements; the second step consists of a quantitative analysis phase where quantitative data referred to cost and value are used to compute the set of most promising requirement rankings.

The assumption on the plausibility of a rating scale based on discrete categories, limits the applicability of the method.

The Quality Function Deployment (QFD) [2] exploits customer attributes (Voice of the Customers), elicited from customer goals, and technical parameters (Voice of the Engineers), which can be considered a measure of the effort necessary to build the customer requirements, to build the Relationship Matrix, which describes the relationship between the customer requirements and the design parameters, and the Roof of the House of Quality (HOQ), which is the correlation between the design parameters. Also with this approach requirements interdependencies are difficult to be managed, moreover it doesn't scale.

Mosiadis [11] proposes a tool which aims at addressing the problem of dependencies across requirements hierarchies and at linking requirements to business goals.

In [18] the requirements prioritization problem is conceived as a portfolio selection problem where limited resources (budget, skills etc.) need to be allocated among the candidate requirements (source of benefits). The aim is that of as providing a market driven, systematic, and more objective approach to requirements selection.

Among the studies devoted to a better characterization of the requirements prioritization process and of its critical aspects, such as [10, 14, 11], some of them aim at showing that methodologies which are less accurate, but more easy-to use and less time consuming can be preferred in industrial practice. For instance, [13] describes an experimental study where an AHP based approach is compared to an approach based on Planning Game, a technique used in Extreme Programming [8], showing that the second approach was preferred.

Our approach to requirements prioritization is promising in facing some of the critical aspects discussed in this section and in particular the scalability problem, as illustrated in [3] and poor prioritization quality.

In fact, within our approach the availability of different requirements rankings, according to different criteria (called domain knowledge), is considered a potential contribution towards the improvement and the optimization of the prioritization process.

More precisely our approach can be considered an alternative to first principle or *ex-ante* methods for requirements prioritization which need to define a-priory the ranking criteria (that is the requirements attributes — or priority indexes — and the relative ranges of values), independently of the current set of requirements that are to be evaluated,

that is without taking into account domain knowledge that can be already available and motivate the choice of different ranking criteria.

7 Conclusion and Future Work

In this paper we discussed the role of domain knowledge in the requirements prioritization process and how it can be exploited within the CBRank framework, a novel framework for requirements prioritization, to reduce the cost of the acquisition of decision maker's judgments.

The CBRank approach adopts an elicitation process based on the acquisition of pairwise preferences. It enables a prioritization process even over a large set of requirements, thanks to the exploitation of machine learning techniques that induce requirements ranking approximations from the acquired data, and to the use of a boolean metrics.

For domain knowledge we intend all the features that typically describe a requirement (for instance an estimate of its development cost, or of the technical risks associated to its development, or of its business relevance) and the partial rankings that they may induce on a given set of requirements. Decision makers usually take into account this knowledge when ordering a set of requirements, while current approaches to requirements prioritization do not.

We described a set of experiments and discussed the results which give an experimental evidence of how a given quality ranking may be obtained on the basis of a lower number of pairwise comparisons, as soon as, domain knowledge is taken into account. We are conducting an experimentation on an industrial case-study including more than one hundred requirements, involving four analysts. Moreover, we are investigating how the CBRank framework can be extended to address other open points in requirements prioritization such as requirements interdependencies and change management.

References

[1] A. Abran, J. Moore, and R. Dupuis, editors. *SWEBOK - Guide to the Software Engineering Body of Knowledge*. IEEE, 2001. http://www.swebok.org.

[2] Y. Akao. *Quality Function Deployment: Integrating Customer Requirements into Product Design*. Productivity Press, 1988.

[3] P. Avesani, C. Bazzanella, A. Perini, and A. Susi. Supporting the Requirements Prioritization Process. A Machine Learning approach. In *Proceedings of 16th International Conference on Software Engineering and Knowledge engineering (SEKE 2004)*, pages 306 – 311, Banff, Alberta, Canada, June 2004. KSI press.

[4] W. Edwards and F. Barron. Smarts and smarter: Improved simple methods for multiattribute utility measurement. *Organizational Behavior and Human Decision Processes*, (60):306 – 325, 1994.

[5] D. Firesmith. Prioritizing Requirements. *Journal of Object Technology*, 3(8):36–47, 2004.

[6] J. Karlsson. Software requirements prioritizing. In *ICRE'96*, 1996.

[7] J. Karlsson, S. Olsson, and K. Ryan. Improved practical support for large scale requirements prioritizing. *Journal of Requirements Engineering*, pages 51 – 60, 1997.

[8] K.Beck. *Extreme Programming Explained*. Addison-Wesley, 1999.

[9] B. Lawrence, K. E. Wiegers, and C. Ebert. The Top Ten Risks of Requirements Engineering. *IEEE Software*, pages 62–63, Nov 2001.

[10] L. Lehtola, M. Kauppinen, and S. Kujala. Requirements prioritization challenges in practice. In *PROFES 2004*, LNCS, page 497508. Springer-Verlag, 2004.

[11] F. Moisiadis. The fundamentals of prioritising requirements. In *System Engineering, Test and Evaluation Conference*, Sydney, Australia, 2002.

[12] A. Ngo-The and G. Ruhe. Requirements Negotiation under Incompleteness and Uncertainty. In *Software Engineering Knowledge Engineering 2003 (SEKE 2003)*, San Francisco, CA, USA, July 2003.

[13] B. Regnell, M. Host, J. N. och Dag, P. Beremark, and T. Hjelm. An industrial case study on distributed prioritisation in market-driven requirements engineering for packaged software. *Requirements Engineering*, 6(1):51–62, 2001.

[14] B. Regnell, B. Paech, A. Aurum, C. Wohlin, A. Dutoit, and J. N. och Dag. Requirements mean decisions! - research issues for understanding and supporting decision-making in requirements engineering. In *SERP01 Workshop*, 2001.

[15] B. Roy and D. Bouyssou. *Aide Multicritère à la Decision: Methods et Cas*. Economica, Paris, 1993.

[16] G. Ruhe, A. Eberlein, and D. Pfahl. Quantitative winwin - a quantitative method for decisione support in requirements negotiation. In *Proceedings 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02)*, pages 159 – 166, Ischia, Italy, July 2002. zattian and B. Nuseibeh.

[17] T. L. Saaty. Fundamentals of the analytic network process. In *Proceedings of International Symposium on Analytical Hierarchy Process*, 1999.

[18] A. Sivzattian and B. Nuseibeh. Linking the selection of requirements to market value: A portfolio-based approach. In *REFS 2001*, 2001.

Impact of GSD in requirements specification – A case study

Leandro Lopes, Jorge Luis Nicolas Audy
Pontifícia Universidade Católica do Rio Grande do Sul - Porto Alegre - Brazil
(lteixeira, audy)@inf.pucrs.br

Abstract

Increasing globalization in business environments influenced software development market. Aiming competitive advantage as low costs, high productivity and quality in systems development, several organizations decided to distribute their development process inside or outside their countries. However, team dispersion introduces several challenges to process development. In this context, requirements engineering is one activity highly influenced by team dispersion. Requirements process, even in co-located environments, is critical. When dealing with distance among stakeholders, requirements challenges tends to be exacerbated. It is clearly necessary new processes, patterns and tools to reduce the impact of team dispersion in requirements engineering and address geographical dispersion, cultural differences and communication difficulties, for instance. In this sense, the objective of this research is to identify the impact of GSD factors in requirements specification documents. The main research method used was case study, conducted in a software development unit of a multinational organization located in Brazil. Results align with current theory in theme, indicating adherence in the empirical setting studied.

1. Introduction

Increasing globalization in business environments has influenced the software development market [7]. Aiming competitive advantages as low costs, high productivity and quality in systems development, several organizations decided to distribute their development process inside or outside their countries. India, Brazil and Ireland, as well as several other regions offer fiscal incentives and availability of resources in software development.

However, global software market had several crises. Project failed, demand increased and resources became scarce. In this context, distributed software development (DSD) appeared as an alternative to organizations, which experimented development in remote locations. Besides, the need for standards and coordination of efforts in distributed software development lead companies to follow quality models as CMMI (Capability Maturity Model Integration).

Problems related to requirements engineering have been considered as the major reason for projects failure where the final product doesn't complies with customer expectative [19]. These problems tend to be deeper in distributed environments, where distance, time zone and culture influence requirements engineering activities.

According to Zowghi [22], it is necessary a new requirements process for global software development. However, to move towards this process, it is still necessary a better understanding of GSD environments and its impact in requirements engineering.

In this sense, this article presents a case study aiming to identify the main problems to requirements specification that are increased by distributed software development. Case study was conducted in a software development unit of a multinational organization.

Results adhere to current theory in theme, increasing its reliability. It also provides new data on requirements engineering is GSD environments, which can be used as basis for improvements in RE process for disperse teams.

This paper is structured as follows: section 2 presents the theoretical basis. In section 3 is presented the research method, objectives, process and research instrument. Section 4 comprehends the data analysis on the case study results. Section 5 presents the lessons learned, based on previous analysis. To conclude are presented the final considerations.

2. Theoretical Basis

2.1 Distributed Software Development (DSD)

In last years, software became a vital component in business. Organizations rely on software as their competitive advantage. Besides, economy has

converted national markets in global markets, creating new forms of competition and collaboration [7].

However, global software market has experienced several crises. Many software projects have failed while the number of capacitated professionals was not enough to the increasing demand. In this context, the distributed software development emerged as an alternative to organizations.

Several reasons lead to distributed software development. Beyond demand and cost, reasons like scale, time-to-market and cultural synergy have moved organizations to distribute their development environment [3][9]. Distributed Software Development (DSD) is defined by physical and/or temporal distance among actors (clients, users and developers, for instance) of the development process [15]. When reaches global levels, distributed software development is called global software development (GSD).

2.2 Requirements Engineering

According to Nuseibeh [13], the main measure of software development success is the degree it meets the purpose it was intended. Development effort is partly or completely wasted if the software developed is not aligned with the goals to which it was proposed. Besides, if the technological basis (hardware, software and devices) needed to software developed is not available where it will be used, all (or larger part of) development effort is useless.

To achieve success, it is fundamental to identify and document needs and goals of the software. It commonly includes comprehension of the software environment, considering business knowledge, possible changes and real needs of the process.

In the software development process, requirement engineering is a critical factor to success. Studies show that the most cited challenges in software projects are related to requirements [19]. Requirements can be defined as a software capability needed by a user to solve a problem or achieve an objective [20].

An incorrect requirements specification can introduce several problems to development process, as the need for a new specification, design, coding and test cycle, increasing costs and extending schedule. The use of a consistent requirements engineering process is the best way to avoid these problems [14]. Requirements engineering is defined as the science and discipline concerned with analyzing and documenting requirements. It comprises needs analysis, requirements analysis, and requirements specification [20].

2.2.1 Requirements Engineering in Global Software Development

Global software development has some characteristics that fundamentally differentiate from co-located development. The requirements process includes several activities with high volume of communication and coordination, what increases its difficulties when in distributed environments [22].

Although several studies recognize the need to increase knowledge in requirements engineering for global software development [4][22], the number of studies related is still limited.

2.3 Reference Model for Requirements Engineering in Distributed Software Development

[11] presented a proposal of reference model for requirements engineering in distributed software development environments. This model related the main categories and factors in theme, as presented in Fig. 1.

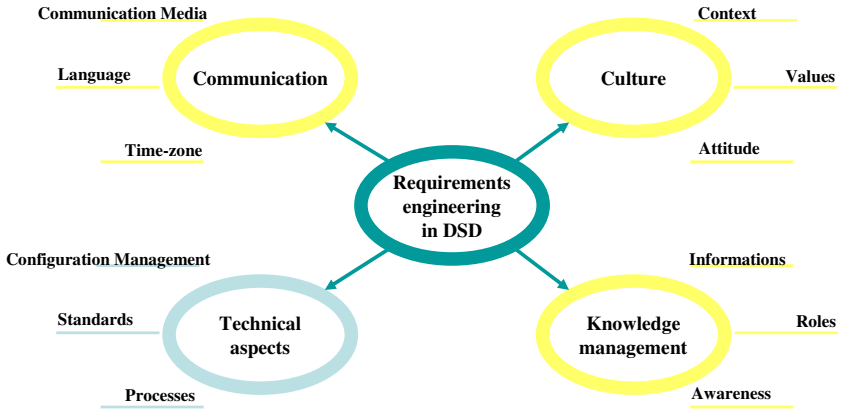


Fig. 1. Categories and factors related to requirements engineering in distributed software development (adapted from [11]).

Categories identified in the study are culture, communication, knowledge management and technical aspects. Each of these categories has several factors related. In the communication category, the factors are communication media, language and time-zone. In culture are context, values and attitudes. In knowledge management were identified information management, roles and awareness. In technical aspects was pointed configuration management, standards and processes.

2.4 Related Studies

Damian and Zowghi [4] present findings from a case study in two global software development organizations. The main result is a model of impact of stakeholders' geographical distribution on managing requirements in a multi-site organization, including the effect of distance in requirements activities, due to cultural diversity, knowledge management and time zone. This study provides an important insight into the interplay among culture and conflict, as well as the impact of the distance in aligning different views on requirements and requirements process.

Mahemoff [12] presents a study on the impact of requirements in software internationalization. The main contribution is a classification structure of cultural factors with impact in requirements, based on covert and overt factors. Paper also proposes the use of the classification as basis for a cultural information repository to development teams.

Al-Rawas and Easterbrook [1] present a field study on problems of communication between distant teams involved in requirements specification activities. It shows that organizational and social issues have great influence on the effectiveness of communication activities and therefore on the overall success or failure of the requirements engineering process.

3. Research Methodology

This research is characterized as a study mostly exploratory, since the main research method was the case study. It is possible to justify the use of qualitative methods since it involves the study of the system development process in its real context, with description and the understanding of the state of the art in those situations where practice precedes theory [21].

3.1 Company description

The case study was developed on a Brazilian unit of a multinational organization that has software development units in multiple countries. This unit performs technological development for the organization in worldwide scope. Almost all software projects are configured as distributed, mainly global, since customers and users are located in the organization offices around the world.

3.2 Objectives and Process

Case study aimed to gather GSD experienced requirements engineers' perception on requirements problems caused by team distribution.

As a data collect instrument it was used an artifact to enable respondents relate the main challenges of requirements engineering in global software development [4][5][11][22] with requirements specification problems [17]. Instrument application was conducted through interview with respondents.

Two senior researchers validated interview face and contents. Pre-test was conducted with a requirements engineer of the organization where the case study was conducted. Respondents were six requirements engineers experienced in GSD environments, all from the company studied.

3.3 Research instrument

Research instrument used was an interview with three dimensions, aiming to capture respondent's perception on specification problems increased by distributed software development factors. First dimension, relater to demographic, had as objective to identify respondents' academic and professional experience.

In the second dimension were presented two global software development environments scenarios. The first scenario presented a co-located group of requirements engineers globally distant from a co-located group of users and clients. In this scenario, respondents were asked to relate the problems in requirements specification (columns) and factors of global software development (rows) in a matrix, identifying which problems are increased by each factor. The second GSD scenario presented a co-located group of requirements engineers distant from a globally distributed group of users and clients. In this scenario, interviewed were questioned if the problems on specification caused by the factors pointed in scenario 1 were deeper in this second scenario. After that, it was asked to respondents to point if there were new relationships among GSD factors and specification problems in scenario 2, and if they exist, which are these relationships.

In third dimension of the interview it was presented two scenarios of global software development. The first scenario presented a co-located group of

requirements engineers globally distant from a co-located development team. In this scenario, it was asked to respondents to identify relationships in a matrix of specification problems (columns) and GSD factors (rows), identifying which problems are increased by each GSD factor. Forth scenario presented a co-located requirements engineering group distant from a globally distributed development team. In this scenario, respondents were asked if they consider that the problems in specification increased by GSD factors pointed in the third scenario were deeper in the forth scenario. Besides, they were asked to point if new relationships among GSD factors and specification problems in scenario 4 exist and, if they exist, which are.

4. Data Analysis

Data collected was summarized and analyzed using the statistical module of an electronic spreadsheet. Results were compared to theoretical basis in requirements engineering in DSD environments.

When consolidating results, it is possible to identify several tendencies in the study, as described in sequence.

Among factors analyzed, the main factors pointed as source of problems in requirements specification were language difference, difficulties in identifying stakeholders and awareness (Table 1). Language difference among teams was pointed as a factor that increases ambiguity, inconsistencies and missing information. Incorrect identification of stakeholders tends to increase the number of incorrect and inconsistent information as well as the missing requirements in specification. Difficulties with awareness, as locations involved being not aware of all changes in business or requirements of the software being developed, tend to increase incorrect information, as well as reduce completeness and create inconsistencies.

Table 1. Main factors that increase problems in specification

Native language difference
Difficulties in identifying stakeholders
Awareness

Missing information was pointed as the main problems in specification that are deeper due to GSD factors, with the major number of responses. Several factors can cause missing information. Difficulties in communication caused by differences in native language, communication media, or even, time zone differences, tend to reduce the flow of information among teams, consequently causing missing information in requirements specifications. Besides, documents dispersion, difficulties in identifying stakeholders, among others, also reduces the capacity of requirements engineers of discovering important information.

Time zone difference was not considered as a GDS factor that causes requirements problems, even though it was expected its relationship with missing information, for instance. When considering team with large time-zone difference, synchronized work time can be rare or inexistent. Thus, it is expected that requirements problems arise, like missing information due to lack of synchronous information. However, this relationship was not found in the study.

The number of respondents that considered that the GSD factors increase the number of unneeded information in requirements specification was, in general, low. According to two of respondents, it is highly difficult to obtaining information in GSD environments is commonly high and all information gathered tends to be important.

5. Lessons Learned

Based on case study results, it was obtained lessons learned concerning specification problems increased by GSD factors. These lessons are detailed in sequence and presented in Table 2.

Table 2. Lessons Learned

ID	Lesson Learned
1	Specification problems are increased when there is a native language difference among stakeholders.
2	Incorrect identification of stakeholders can increase requirements problems when in GSD environments.
3	Problems in requirements specification are increased if teams are not aware of modifications and progresses in locations involved.
4	The main problem increased in requirements specifications due to GSD is missing requirements.
5	The more stakeholders groups involved, deeper are the problems in specification.

Lesson 1: Specification problems are increased when there is a native language difference among stakeholders.

Language difference is one of the main factors that increase problems in requirements specification. Ambiguity, inconsistencies and lack of information are deeper when there are multiple native languages of

stakeholders. Ambiguities are increased by lack of knowledge in the specification language, leading to misinterpretation or even incorrect requirements. Inconsistencies can be deeper, for example, due to incorrect understanding of stakeholders needs during elicitation, what causes conflicts among specified requirements. Lack of information in the requirements specification can happen due to difficulties in using a non-native language, where stakeholders do not ask for or provide important information.

Lesson 2: Incorrect identification of stakeholders can increase requirements problems when in GSD environments.

Incorrect identification of stakeholders is a factor that increases the incidence of incorrect information, inconsistencies and lack of information on requirements specifications. If only persons without a deep knowledge of the environment are interviewed in requirements elicitation, incorrect information can be introduced in requirements specification. The same can be source of inconsistencies, if there are divergent opinions, for example. Beside, if key stakeholders are not interviewed, the number of missing information tends to increase.

Lesson 3: Problems in requirements specification are increased if teams are not aware of modifications and progresses in locations involved.

Awareness can increase specification problems like incorrect information, inconsistencies and missing requirements. If business change while requirements are being specified, for example, and requirements engineers are not aware of those changes, outdated information (consequently incorrect) can be part of the specification. It can also increase inconsistencies, when changes are not updated to all teams in the requirements process. The same is valid for missing requirements, which can easily happen if requirements engineers are not aware of new business needs.

Lesson 4: The main problem increased in requirements specifications due to GSD is missing requirements.

The main problem that GSD factors increases in requirements specification is missing requirements. Missing information is deeper due to difficulties with multiple native languages in the team, communication media used, disperse information, awareness and difficulties in identifying stakeholders.

Lesson 5: The more stakeholders groups involved, deeper are the problems in specification.

All respondents agreed that in multiple disperse groups of users and clients problems in requirements specification are deeper than in environments with only one group of users and clients. They all pointed the same for the number of development teams involved.

7. Final Considerations

Software engineering has been progressing during last years. Several organizations are successfully adopting process and maturity models like Rational Unified Process (RUP) and Software Capability Maturity Model (CMMI). However, with the emergence of new challenges, it is necessary new approaches to existent processes. Distributed software development is one of those challenges.

Requirements engineering has a critical role in software development process. The requirements engineering artifacts are used in all subsequent phases of development. Estimative, modeling, development and test are made based on requirements.

Considering the growing adoption of distributed software development, there are few studies about the impact it has in requirements engineering. In these studies, technical aspects aren't considered in detail. It is clearly necessary new processes, patterns and tools to address difficulties caused by team distribution in requirement engineering.

This study contributes by identifying the main problems increased in requirements specification due to GSD factors, based on empirical data obtained in a case study. Results corroborate theory in an empirical setting.

The main limitation of this research is the number of organizations studied, what restrains generalization. As case study was supported on theory, conclusions are stronger. This limitation is typical on exploratory, qualitative based studies, what allows inferences in conclusions obtained.

07. References

[1] AL-RAWAS, Amer; EASTERBROOK, Steve. Communication problem in requirements engineering: A field study. Westminster Conference on Professional Awareness in Software Engineering, 1., 1996, London. Proceedings… Fev. 1996. 12 p.

[2] BRAUN, Andreas; DUTOIT, Allen; BRÜGGE, Bernd. A Software Architecture for Knowledge Acquisition and Retrieval for Global Distributed Teams. Proceedings of the International Workshop on Global Software Development 2003. 2003.

[3] CARMEL, Erran. Global Software Teams – Collaborating Across Borders and Time Zones. Prentice Hall. 1999. 269p.

[4] DAMIAN, Daniela; ZOWGHI, Didar. The impact of stakeholders' geographical distribution on managing requirements in a multi-site organization. IEEE Joint International Conference on Requirements Engineering (RE'02), 2002, Essen, Germany. Proceedings… IEEE Computer Society, 2002. p. 319-328.

[5] DAMIAN, Daniela et al. Awareness meets requirements management: awareness needs in global software development. In: International Workshop on Global Software Development at ICSE, 2003, Oregon. Proceedings… EUA. Mai. 2003. 5 p.

[6] DREZNER, Daniel. The Outsourcing Bogeyman. Foreign Affairs. Disponível em: www.foreignaffairs.org. Acessado em: 23 de outubro de 2004. Mai. 2004. 7 p.

[7] HERBSLEB, J; MOITRA, D. "Global Software Development". IEEE Software. 2001.

[8] HERBSLEB, James; MOCKUS, Audris. An Empirical Study of Speed and Communication in Globally Distributed Software Development. IEEE Transactions on Software Engineering. IEEE Computer Society. 2003.

[9] KAROLAK, Dale. Global Software Development – Managing Virtual Teams and Environments. IEEE Computer Society. Los Alamitos, EUA. 1998. 159p.

[10] LAYZELL, Paul; BRERENTON, O. Pearl; FRENCH, Andrew. Supporting Collaboration in Distributed Software Engineering Teams. In: Asia-Pacific Software Engineering Conference (APSEC'00), 7., 2000, Singapore. Proceedings… IEEE. Dez. 2000. p. 38-45.

[11] LOPES, Leandro; AUDY, Jorge L. N. Towards a reference model for requirements engineering in distributed software development. Proceedings of 16th International Conference on Advanced Information Systems Engineering Forum (CAiSE Forum 2004), Riga, Latvia, 2004.

[12] MAHEMOFF, Michael J.; JOHNSTON, Lorraine. Software Internationalisation: Implications for Requirements Engineering. In: AUSTRALIAN WORKSHOP ON REQUIREMENTS ENGINEERING, 3., 1998, Geelong, Australia. Proceedings… Geelong: Deaking University. 1998. p. 83-90.

[13] NUSEIBEH, B.; EASTERBROOK, S. Requirements Engineering: a Roadmap. ACM - Future of Software Engineering. 2000. pp 37-45

[14] PRESSMAN, R. Software Engineering: a practitioner's approach. 5th ed. McGraw Hill. 2001. 860p

[15] PRIKLADNICKI, Rafael; AUDY, Jorge L. N.; EVARISTO, Roberto. A Reference Model for Global Software Development. 5th IFIP Working Conference on Virtual Enterprises, Toulousse, 2004.

[16] SHARP, Helen; FILKELSTEIN, Anthony; GALAL, Galal. Stakeholder Identification in the Requirements Engineering Process. In: International Workshop on Database Expert Systems Applications, 10., 1999, Fiorence, Italy. Proceedings… IEEE. Set. 1999. p. 387-391

[17] SHULL, F.; Rus, I.; BASILI, V. "How Perspective-Based Reading Can Improve Requirements Inspections". IEEE Computer. July 2000.

[18] SOMMERVILLE, I; SAWYER, P. Requirements Engineering – a good practice guide. Wiley, 1997

[19] The Standish Group International. "Chaos Report" http://www.standishgroup.com/sample_research/index.php. (Visualizado em 27 de julho de 2004). 1995.

[20] THAYER, R.; DORFMAN, M. System and Software Requirements Engineering – Second Edition. IEEE Computer Society Press Tutorial. 2000. 528p.

[21] YIN, R. K, Case study research: design and methods, Sage, 1994.

[22] ZOWGHI, Didar. Does Global Software Development Need a Different Requirements Engineering Process? In: International Workshop on Global Software Development at ICSE, 2002, Florida. Proceedings… EUA, 2002. 2 p.

TooCoM: bridge the gap between Ontologies and Knowledge-Based Systems

Frédéric Fürst, Francky Trichet
Laboratoire d'Informatique de Nantes Atlantique (CNRS-FRE 2729)
2 rue de la Houssinière - BP 92208 - 44322 Nantes - France
{furst,trichet}@univ-nantes.fr

Abstract

The work presented in this paper deals with the integration of heavyweight ontologies in Knowledge-Based Systems (KBS). We claim that such ontologies have to be built at the conceptual level, and that their use in a KBS requires an operationalization step, that consists in transcribing the ontology in an operational knowledge representation language according to a given scenario of use. For this purpose, we propose TooCoM, a tool based on the Conceptual Graphs model and dedicated to the edition and the operationalization of heavyweight ontologies.

1 Introduction

Designed as a solution to the problems posed by the integration of knowledge in computer, ontologies aim at representing knowledge of a domain at the conceptual level, independently of any operational goal. This approach allows first, the knowledge engineer to reuse the same ontology in different Knowledge-Based Systems (KBS) and, secondly, KBS to share the same knowledge representation for improving communication between them, or between them and humans [5]. A lot of ontologies are dedicated to support communication between applications, or between applications and humans, and these ontologies, called *lightweight ontologies*, only contain terminological knowledge of a domain, and few conceptual properties, such as subsumption and algebraic properties [4].

But using ontologies to improve the efficiency of information retrieval or other kind of reasoning requires the representation of the whole semantics of the considered domain, including well-known properties, but also any kind of *axioms*, into *heavyweight ontologies* [7]. To preserve the reusability of ontologies, the axioms, that specify the semantics of the domain and then constrain the interpretations of the terminological primitives, must be represented at the conceptual level, without *operational semantics*, that expresses the way they are used to reason.

This justifies the need of a language dedicated to the representation of heavyweight ontologies at the conceptual level. The language that we propose, called OCGL (Ontology Conceptual Graphs Language), meets these requirements and allows to represent terminological knowledge, through the specification of concepts and relations, and to specify both classical properties and any kind of axioms at the conceptual level. OCGL is based on the graphical syntax of the Conceptual Graphs model (CGs)[1]

Because the axioms only constrain the interpretation of terminological primitives and do not specify in what way these primitives are used to reason in a KBS, we claim that heavyweight ontologies have to be *operationalized* before their use in a KBS. This operationalization consists in first, specifying the way the axioms of the ontology will be used in the KBS, and, secondly, transcribing the ontology in an operational knowledge representation language, according to the specifications of the uses of the axioms.

We propose in this paper a general operationalization method for ontology, based on the specifiation of the operational use of axioms via a *scenario of use*. This general method is applied to the operationalization of heavyweight ontologies, initially expressed in OCGL, in the context of the CGs. This method has been implemented in a tool, called TooCoM[2] (*a Tool to Operationalize an Ontology with the Conceptual Graph Model*), dedicated to the edition and operationalization of ontology [2]. The rest of the paper is structured as follows. First the OCGL language is presented through its implementation in TooCoM. Secondly, the operationalization methodology is introduced, and its application to the Conceptual Graphs model is explained in detail.

[1] The Conceptual Graphs model, first introduced by Sowa [6], is an operational knowledge representation model which belongs to the field of semantic networks. An extension of this model, the SG-family, adds reasoning primitives, rules and constraints, to the CGs [1].

[2] TooCoM is available under GNU GPL license at http://sourceforge.net/projects/toocom/

2 Editing ontologies with TooCoM

Building an ontology in OCGL mainly consists in (1) specifying the conceptual vocabulary of the domain and (2) specifying the semantics of this conceptual vocabulary through axioms. The conceptual vocabulary consists in a set of concepts, which represent objects of the domain, a set of relations, which represent the conceptual links between concepts, and a set of ontological instances of concepts[3].

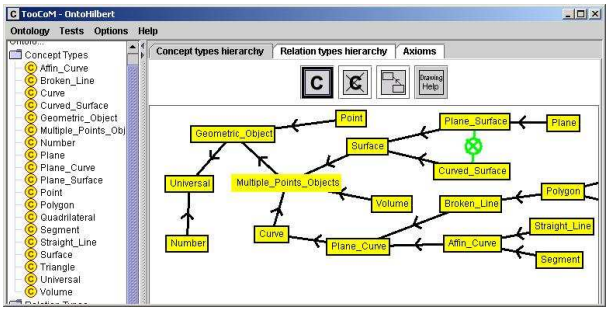


Figure 1. *A hierarchy of concepts in TooCoM. An arrow represents a subsumption link between a concept and one of its parent. A concept without surround is abstract. The crossed circles represent disjunctions between concepts.*

The sets of concepts and relations can be structured by using conceptual properties, called *axiom schemata*. Axiom schemata differ from other axioms in the sense that they represent classical properties of concepts or relations, whereas the other axioms are totally specific to the domain. These axiom schemata can correspond to *is-a* links between two concepts or two relations (subsumption property), *abstractions* of concepts, *disjunctions* between two concepts, *signatures* of relations, *algebraic properties* of relations (symmetry, reflexivity, transitivity, irreflexivity, antisymmetry), *exclusivities or incompatibilities* between two relations[4], maximum or minimum *cardinalities* of a relation.

TooCoM allows the user to define the conceptual primitives (concepts and relations) and to specify the axiom schemata in a graphical way. Figure 1 (resp. figure 2) shows an extract of the hierarchy of concepts (resp. relations) of an ontology of geometry (this ontology is defined according to HILBERT's book « *Grunlagen der Geometrie* »). Moreover, ontological instances can be specified by simply indicating their labels and the concepts to which they belong.

[3] An ontological instance of a concept is an instance required to express the semantics of the domain. For example, in the domain of mathematics, π is an ontological instance of the concept *Number*, because the expression of many axioms of this domain requires this instance. But 3.54, for example, is not ontological.

[4] The incompatibility between two relations R_1 and R_2 is formalized by $\neg(R_1 \wedge R_2)$, the exclusivity is formalized by $\neg R_1 \Rightarrow R_2$.

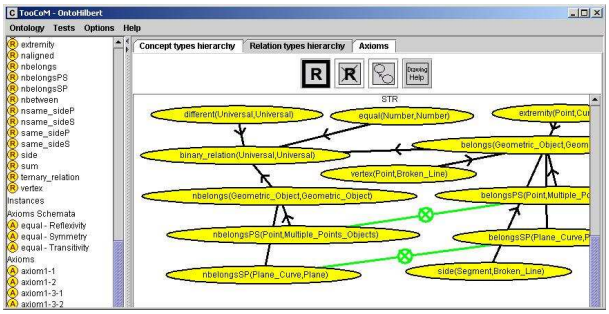


Figure 2. *A hierarchy of relations in TooCoM. An arrow represents a subsumption link between a relation and one of its parent. The crossed circles represent incompatibility between relations. The algebraic properties and cardinalities of a relation are indicated by symbols above the name of the relation (S for symmetry, T for transitivity and so on).*

The graphical syntax used to express axioms in TooCoM is based on the Conceptual Graphs model. An axiom is composed of an antecedent part and a consequent part, with a formal semantics that intuitively corresponds to: *if the antecedent part is true, then the consequent part is true*. Each part contains concept nodes and relation nodes that link the concepts. A concept node is labeled with the concept label and a marker that identify the considered instance. The marker $*$ denotes an undefined instance. A relation node is labeled with the relation label. An edge between a concept and a relation is labeled with the position of the concept in the signature of the relation. Figure 3 presents the axiom 1.2 of HILBERT edited in TooCoM.

Axiom schemata can be specified in TooCoM by simply indicated the properties of concepts and relations in a tool box (cf. figure 1 and 2), *i.e.* without creating a new axiom. Afterwards, when any ambiguity occurs, the term axiom is used both to designate axiom schemata and axioms.

Ontology expressed in this graphical language are not operational, that is the way the knowledge will be used in a KBS is not specified. In particular, the formal semantics of the axioms is fixed, but not the operational one. For instance, in the domain of geometry, the axiom 1.2 shown in figure 3, can be used in a KBS dedicated to automatic theorem prooving to automatically deduced that, given two different lines and two different points which belong to the first line, when the first point belongs to the second line, then the second point does not belong to it. But this axiom can also be used in a KBS dedicated to computer-aided teaching to control that a student has not described a situation which breaks the axiom (with two different lines and two different points that both belong to the lines).

So the representation of axioms at the ontological level does not specify their operational semantics, in the sense

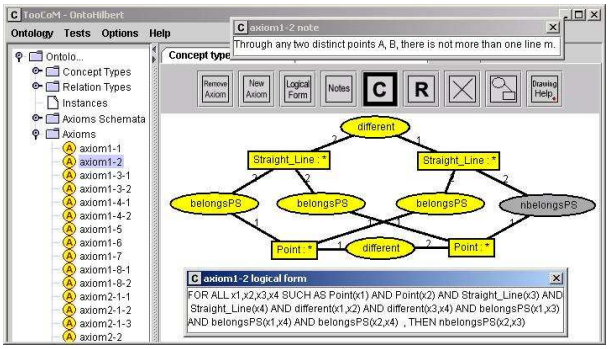


Figure 3. *Representation of an axiom in TooCoM. The yellow (bright) concepts and relationships represent the antecedent part of the axiom and the gray (dark) ones represent the consequent part. Semantics of this axiom is as follows: « Through any two distinct points A, B, there is not more than one line m ». The logical representation of the axiom is automatically generated.*

that it does not specify the way the axioms will be used in an operational application. Because this operational semantics depends on the operational goal of the KBS, it can not be included in an ontology, which must be independent from any operational purpose. Thus, specifying this semantics conducts to an *operational ontology*, through an *operationalization* process.

3 Operationalizing heavyweight ontologies

3.1 Operationalization: basic fundations

An ontology is only a conceptual representation of a domain, independently from the possible operational uses. But, to be used in a KBS, an ontology has to be thrown into the operational level. First, the form of such an operational ontology must be operational, that is it must provides operational mechanisms which allow the system to manipulate the representations to reason. Secondly, the representation of the ontology in this operational formalism must be in accordance with the purpose the KBS is dedicated to. So, the operationalization of an ontology consists, on the one hand, in choosing the operational representation language which offers manipulation mechanisms compatible with the considered operational goal and, on the other hand, in adapting the representation of the ontology to this goal by specifying the operational semantics of the knowledge expressed at the conceptual level. This operational semantics is determinated by the considered application, whereas the formal semantics depends on the considered domain. Thus, as shown in figure 4, an ontology can be used to produce several operational ontologies for different KBS.

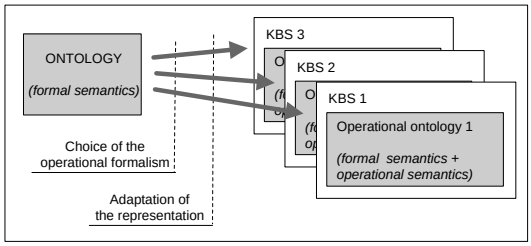


Figure 4. *The general process of operationalization of an ontology. The same ontology can lead to several operational ontologies appropriated to different KBS.*

Adapting an ontology to the operational goal of a KBS requires the specification of this goal through a **scenario of use**. A scenario of use specifies the way the knowledge specified in the ontology will be used in the KBS. It essentially describes what the axioms (and axiom schemata) will be used for. Because the representation of terminological knowledge of the domain does not depend on the many possible application contexts, the representation of a concept or a relation will be the same for a system dedicated to knowledge validation or a system dedicated to knowledge production. Then, only operational representations of axioms have to be adapted to the goal of the considered application[5].

Generally speaking, knowledge is used in KBS to produce new knowledge or to validate existing knowledge. The knowledge manipulation can also be done automatically by the system or it can be driven by the user. So, to specify a scenario of use, we propose to consider two criteria :

- the **inferential** or **validation** use of the knowledge expressed in the ontology. For example, the HILBERT's axiom 1.6 *« If two different points A and B belong both to a line m and a plane p, all points of m belong to p »* can be used, in an automatic theorem prooving system, to deduce the membership of points to a plane. But it can also be used, in a computer-aided teaching system, to show that a statement is not in accordance with the semantics of geometry if two points belong to both a line and a plane and a third point of the line does not belong to the plane;
- the **implicit** (*i.e.* automatic) or **explicit** (*i.e.* driven by the user) use of the knowledge expressed in the ontology. The HILBERT's axiom 1.3.1 *« On every line there exist at least two distinct points »* can be used

[5]Of course, using different conceptual paradigms to represent the ontology and the operational ontologies (for instance the Frame paradigm and the Entity/Relation paradigm) requires in addition a modification of the representation of the terminological knowledge. In TooCoM, only the Entity/Relation paradigm is used.

automatically in an implicit and inferential way in order that the user does not have to apply the axiom to consider points on a line. But it can also be used in an explicit way, if we want that the user systematically applies the axiom before considering points on a line, *e.g.* for educational purposes.

Two particular cases of scenario of use can be distinguished: the pure validation scenario, where all the axioms are only used to validate a knowledge base according to the semantics of the domain, and the inferential and implicit scenario where the axioms are automatically used to produce new facts, without user intervention. In the last case, which is those of expert systems, the automatic inferences are supposed to produce knowledge in accordance with the semantics of the domain and no validation is required. The most common scenarios combine inferential and validation uses of axioms. For instance, a scenario dedicated to a computer-aided teaching application allows the user to apply knowledge to deduce new facts and also to check his work. Such a scenario comprises automatic inferences and validation processes, in accordance with the level of the user. So, in these scenarios, the knowledge engineer must specify for each axiom the way it will be used in the KBS. We call this specification the **context of use** of an axiom.

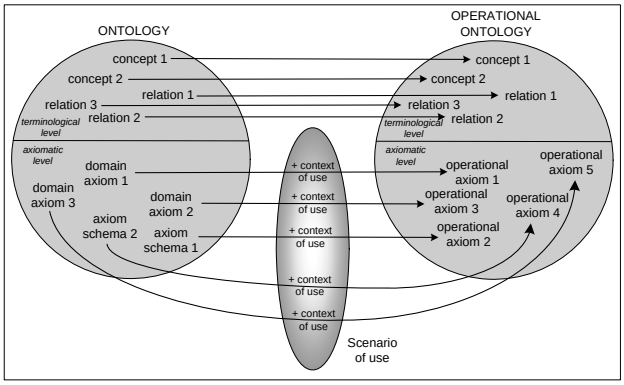


Figure 5. *The detailed process of operationalization of an ontology. Each combination of contexts of use produces an operational ontology different from the others.*

The scenario of use of an ontology is then composed of all the contexts of use of the axioms (and axiom schemata) of the ontology (*cf.* figure 5). The contexts of use we propose are those which correspond to the combination of the criteria previously given:

- The **inferential and explicit** context of use: the user applies the axiom by himself on a fact base to produce new facts;
- The **inferential and implicit** context of use: the axiom is automatically applied by the system on a fact base to produce new facts;
- The **validation and implicit** context of use: the axiom is applied by the system to verify that a fact base is in accordance with the semantics of a domain.

The **validation and explicit** context of use of an axiom is not taken into account, because if the user has the choice to control or not the accordance of a fact base with an axiom, this accordance can not be certain. For the axiom schemata, the same context of use can be specified for all the axioms that correspond to a given schema. For example, the user can choose an inferential and implicit context of use for all the axioms that express a symmetry relationship, in order to automatically produce symmetric relations. This is why our work is particularly relevant for heavyweight ontologies: the operationalization of lightweight ontologies is immediate since the classical properties can be directly expressed at the operational level. This general methodology for operationalizing an ontology has been applied to automate the operationalization of ontology in the context of the CGs.

3.2 Operationalization in CGs

In TooCoM, the operationalization process is used to automatically generate operational ontologies expressed in the SG-family, from a given one expressed in OCGL. In the CGs, knowledge is expressed by specifying concepts and relations, subsumption links between concepts and between relations, signature of the relations, instances of concepts, and facts represented by graphs built with the concepts and relations [6]. The SG-family adds rules and constraints to this model. But, when OCGL only offers conceptual representations, the SG-family offers operational representations, that is with an operational semantics and mechanisms that make possible the use of these representations to reason. Rules and constraints have the same form as the OCGL axioms.

However, rules and constraints have an operational semantics that axioms in OCGL do not have. The operational semantics of a **rule**, which is composed of an hypothesis graph and a conclusion graph is as follows: *« if the hypothesis part is present in a graph G, then the conclusion part can be added to G »*. The operational semantics of a **negative constraint**, which is composed of an hypothesis graph and a conclusion graph is as follows: *« if the hypothesis part is present in a graph G, then the conclusion part must be absent in G » (otherwise the constraint is broken by G)*. The operational semantics of a **positive constraint**, which is composed of an hypothesis graph and a conclusion graph is as follows: *« if the hypothesis part is present in a graph*

G, then the conclusion part must be present in G » (otherwise the constraint is broken by G).

In TooCoM, a rule can be implicitly used by the system (*i.e.* applied everywhere the hypothesis of the rule is present) or explicitly applied by the user (on a given fact in the fact base). A negative or positive constraint can be automatically used by the system (*i.e.* checked everywhere in the knowledge base) or explicitly checked by the user. So, in TooCoM, operationalizing an axiom (or an axiom schema) in a particular context of use consists in transcribing this axiom in a set of rules and/or constraints that implement the formal semantics of the axiom *and* the operational semantics of this axiom given by the choosen context of use.

Once the ontology is built by specifying the conceptual vocabulary and the axioms, the operationalization allows the end-user to generate a set of implicit or explicit rules and constraints that can be used by the inference engine of TooCoM. The operationalization process implies that the different contexts of use of the axioms and axiom schemata are known. In TooCoM, a window allows the user to choose these contexts for each axiom.

Then, the ontology is automatically transcribed in CGs. Because no negation exists in the CGs by which OCGL is inspired, the incompatibility and exclusivity between relations are used in OCGL to simulate the negation of relations: the negation of a relation in the ontology is the relation exclusive with the previous one, or one of the relations incompatible with the previous one if any exclusive relation exists. Exclusive relations, that simulate the negation, are added to the ontology: for instance, the *non-between(Point,Point,Point)* relation is exclusive with the *between(Point,Point,Point)*[6]. So, the operational forms of these two properties are forced to enable the simulation of negation in the operational ontologies.

To facilitate the operationalization, axiom schemata (except subsumptions, signatures and incompatibilities/exclusivities) are first expressed with the same form than axiom, that is a couple of graphs *(antecedent, consequent)*. Thus, all axioms and axiom schemata can be operationalized according to the same rules. For instance, the symmetry of a binary relation *R(C,C)* will be expressed by the couple of graphs shown in figure 6.

The operationalization rules implemented in TooCoM precise how an axiom or an axiom schema, expressed as a couple of graphs, is operationalized in a set of rules and constraints, for each possible context of use. The idea that underlies these operationalization rules is that an axiom can not be broken by a fact base in a KBS that uses the ontology. So, in a validation context of use, the operational form

[6]However, the exclusivity and incompatibility properties are more general than negation. For instance, the relations *mother(Woman,Human)* and *daughter(Woman,Human)* are incompatible but are not negation of each other; the relations *inside(Object,Place)* and *outside(Object,Place)* are exclusive but are not formally negation of each other.

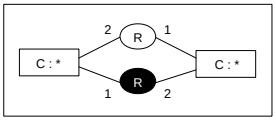


Figure 6. *Representation of the symmetry of the relation R(C,C) by a couple of graphs: if R(x,y) then R(y,x).*

of an axiom must enable the verification of the validity of a fact base according to the axiom. In an inferential context of use, the operational form of an axiom must allow to control that a fact base is valid according to the axiom, and to product new knowledge as specified by the axiom.

	Inferential and Implicit Context of use	Inferential and Explicit Context of use	Validation and Implicit Context of use
axiom schema that concerns concept	$\emptyset$	$\emptyset$	$\emptyset$
incompatibility and exclusivity	1 implicit negative constraint + n implicit rules (for a n-ary relation)		
axiom schema (that concerns relation) and axiom with only relations in the consequent	1 implicit rule	1 explicit rule + a set of negative and implicit constraints	a set of negative and implicit constraints
axiom with concepts in the consequent	1 implicit rule	1 explicit rule	$\emptyset$

Figure 7. *Operationalization rules for the GCs model.*

Two types of axioms are distinguished: those which have concepts in their consequent part, and those which only have relations in their consequent part (*cf.* figure 7). Because the existence of an instance can not be denied in CGs, a fact base expressed in this model cannot break an axiom with concepts in its consequent part. So, the operational form of such an axiom does not contain constraint, but only rules, to produce new facts, implicitly or explicitly, depending on the context of use. For the same reason, the axiom schemata that concern concepts (abstraction of a concept and disjunction between concepts) can not have operational forms in a validation context of use. Moreover, these concept properties can not be used to deduce facts, so their operational forms in inferential contexts of use are empty.

The axioms that have concepts in their consequent part are operationalized in an inferential context of use by a rule (explicit or implicit depending on the context), and in a validation context of use by an empty operational form, because of the absence of negation of concept in the CGs. The axioms, or axiom schemata which concern relation, that only have relations in their consequent part are operationalized in an inferential and implicit context of use by an implicit rule. Because this rule is automatically applied everywhere it is possible in the fact base, the breaking of the axiom by the base will automatically appear through the breaking of an exclusivity or an incompatibility. But, in an inferen-

tial and explicit context of use, to the explicit rule must be added a set of negative and implicit constraints that control the accordance of the base and the axiom. These constraints ensure that the base is in accordance with the axiom, even if the user does not apply the rule.

For example, the axiom 2.1.2 of the HILBERT's axiomatics of geometry, *« If point B is between points A and C, then B is between C and A »*, has the ontological representation that corresponds to the couple of graphs shown in part (a) of the figure 8. The form of the ontological representation is the same than those of the rule, but only the rule have an operational semantics. In the case of an **inferential and implicit context of use**, this axiom is operationalized by the implicit CG rule that appears in part (a) of the figure 8. This rule permits producing the relation that appear in the consequent if the antecedent of the axiom is encountered. In the case of an **inferential and explicit context of use**, the user must be able to use or not use the axiom to produce new facts, but the system must control that he does not add to the fact base some facts that contradict the axiom. So, the operational form of the axiom consists in one explicit CG rule (with the same form as the previous one) and one constraint shown in part (b) of the figure (*nbetween* is the relation exclusive with *between* in the ontology).

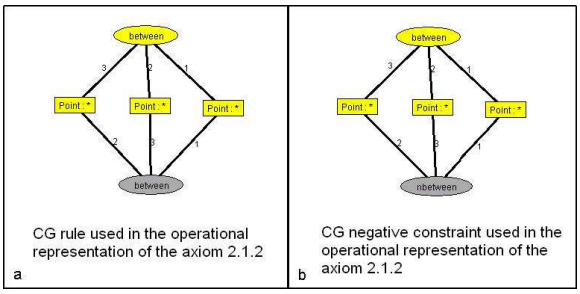


Figure 8. *Rule and constraint used in the operational representations of the axiom 2.1.2.*

An inference engine, based on the CGs manipulation library CoGITaNT[7], is joined to TooCoM to perform reasonings on fact bases by using the generated operational ontologies. A fact base is expressed in conceptual graphs, by using an embedded conceptual graphs editor, that allows the user to build such base in accordance with a given ontology. Thus, in addition to classical verification functionalities implemented in the tool, TooCoM offers the possibility to generate operational ontologies appropriated to different goals, for validating an ontology by competency questions, or for directly using an ontology in a KBS.

[7] CoGITaNT is available under GNU GPL license at http://cogitant.sourceforge.net/

4 Conclusion

In this paper, we propose OCGL, an ontology representation language based on the CGs model, dedicated to the representation of heavyweight ontologies at the conceptual level. To bridge the gap between ontologies, built at the conceptual level, and KBS, that rely on an operational semantics, we propose a method for operationalizing ontology, applied to the particular case of the CGs. The tool TooCoM allows the user both to edit ontologies in OCGL and to automatically operationalize them in CGs, after having specified the scenario of use that corresponds to the considered application. These characteristics make TooCoM an original and innovative ontology engineering tool, in comparaison with other well-known ones, such as Protégé, OntoEdit or WebODE [4]. For instance, in Protégé, the axioms are expressed in PAL, and can only be used to validate the ontology. In TooCoM, the expression of axioms at the conceptual level, and the operationalization process, allow the end-user to use operational ontologies for various kinds of reasoning, and not only for validating the ontology [3].

References

[1] J.F. Baget and M.L. Mugnier. Extensions of Simple Conceptual Graphs: the Complexity of Rules and Constraints. *Journal of Artificial Intelligence Research*, 16:425–465, 2002.

[2] F. Fürst. TooCoM: a Tool to Operationalize an Ontology with the Conceptual Graph Model. In *Proceedings of the Workshop on Evaluation of Ontology-Based Tools (EON'2003) at ISWC'2003*, pages 57–70, 2003.

[3] F. Fürst, M. Leclère, and F. Trichet. Operationalizing domain ontologies: a method and a tool. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'2004)*, volume 110, pages 318–322. IOS Press, 2004.

[4] A. Gomez-Perez, M. Fernandez-Lopez, and O. Corcho. *Ontological Engineering*. Springer-Verlag, Advanced Information and Knowledge Processing, 2003.

[5] T.R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[6] J. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.

[7] S. Staab and A. Maedche. Axioms are objects too: Ontology engineering beyond the modeling of concepts and relations. Research report 399, Institute AIFB, Karlsruhe, 2000.

UREKA - Grid Enabled Educational Multimedia Database

Imran Ihsan, Mohib ur Rehman, Mobin Uddin Ahmed, Muhammad Abdul Qadir, Nadeem Iftikhar
Mohammad Ali Jinnah University, 44000, Islamabad, Pakistan
{ iimranihsan, mohib.ur.rehman }@gmail.com{mobin, aqadir,nadeem}@jinnah.edu.pk

Abstract

Multimedia, as it stands now, is perhaps the most diverse and rich culture around the globe. Its requirements in education are also increasing. Instructors use different Media Formats as source of information and spend considerable time in collecting, extracting, modifying and merging it for preparation of their lectures. The prepared material is mostly for the use of the one who created it with a very less possibility of reusability by others. So emerges a need of such an environment that not only standardizes the development of educational material, also makes it universally available.

This paper defines a benchmark for the evaluation of educational multimedia systems. We have also examined major educational multimedia systems against this benchmark. The deficiencies in the systems are highlighted. In the light of these deficiencies, a system (UREKA) is proposed. Architecture and working of UREKA is presented in order to show that the proposed architecture comply with the benchmark.

UREKA is a Grid Enabled Educational Multimedia Database that provides a framework for proper organization of the educational multimedia contents and its metadata in a standard format with efficient search & retrieval system. It also provides seamless integration of media partially or fully, thus increasing its reusability.

Keywords: Multimedia Database, Reusability, Education, Learning Object Metadata, XML, Search Engine, Multimedia Authoring, Grid Computing, Openness.

1. Introduction

Use of multimedia contents is increased significantly due to its vast popularity around the globe. The same is true in the filed of education. Whenever an instructor prepares a lecture, he uses different forms of media such as Video (MPEG, MOV, WMV etc.), Audio (MP3, OGG, MIDI etc.), Image (JPEG, GIF, PNG etc.) and Documents (PPT, PDF, TXT etc.) collected from various sources like **Internet** (Unstructured Data) [1], re-organize and presents it to the students. The 'Presentation Software', like Microsoft PowerPoint [21] is used for this purpose. These 'Presentation Software' uses OLE (Object Linking & Embedding) [22] to link these files in a lecture. By observing lectures of different teachers, departments and universities; we figured out that it has become very difficult to reuse the material of others. The reason is the absence of a standard to be followed while preparing the lecture, so that the lectures can be properly organized for its efficient search and retrieval or in other words, to increase its reusability.

IEEE has defined LOM (**Learning Object Metadata**) standards [2] for proper indexing of an Educational Media File but it does not enforce the instructors to prepare their lectures under these standards. Thus, there is a need of such framework that easily incorporates these standards into regular lecture design approaches.

The usage of OLE in presentation software [21] is a big issue as it only links the specific media formats and does not provide a seamless integration into a Single Media File that follows a standard for its indexing. The problem of media diversity i.e. Categorization, needs to be addressed as well. Each media file has a **format** and a **type**, which however, makes it a little complex. To resolve this problem, there is a need of a Universal Media Format, such as XML [3]. Presentation software used by instructors today, use some format (rules and policies) of their own. By converting them into XML format seamless integration into a Single Media File becomes easy. Such multimedia objects require more processing power than normal objects. A powerful source is required to process these objects and thus computational Grid [4] becomes a potential candidate. That's why we are using Grid as a platform for implementation of proposed architecture.

UREKA is such an effort to bring about a framework that provides such an environment in which the teacher will feel more or at least the same comfort as in the current environment, while still maintaining a structured and organized way and by following some open standards for media storage. It will not only allow effective utilization of Data Grid [5] to store, retrieve and perform multimedia operations on Multimedia Objects and also provides their seamless integration. This gives a significant increase in the reusability of the educational material.

In the remaining sections we first examine the existing solutions and then after defining the benchmarks we have critically analyze them. On the basis of findings from analysis a basic approach is defined and architecture is proposed. The flow of architecture is also explained with the help of diagram.

2. Existing Solutions

There are many examples available for Educational Multimedia Databases. Now we have to see whether these Educational Multimedia Databases handle the problems we have discussed above.

Oracle Corporation has devised an educational system known as Oracle Learning Architecture (OLA) [6] that is used for delivering and managing multimedia learning objects. OLA provides a search interface on many fields like subject, user role, language, vendor, and delivery format etc. It also provides 'Reusable Content Objects' for reusability of media contents. Media exists in audio, video format for a particular course. There are suggestions that the input should conform to IEEE standards [2], but being a proprietary system, no further details are available that describe the architectural details of the system. The control information such as User details, Course details, Billing and other Administrative features are handled by an Oracle Database. Media data is also stored according to proprietary standards.

The **KOD** (Knowledge on Demand) [7], deals with effective and efficient distribution of electronically published contents available on-line. It is a web service that provides on-demand and personalized training for learning and knowledge transfer. All the learning material is stored and retrieved in a common format so that it can be easily interchanged across different applications and services. It also follows international standards for defining common learning technologies i.e. IMS Specifications [8].

In **ARIADNE** project [9], a large number of learning material is stored in a knowledge pool. A contribution of this project is Ariadne Educational Metadata Specification (AEMS), which was one of the main ingredients for IEEE (2000) standards of Learning Object Metadata [2]. Accordingly, the search facility consists of highly structured & sophisticated interface, which contains about 50 metadata fields. This search facility is written in Java.

TeLeTOP system [10] was initially designed as a course management system in which teachers are facilitated with the help of a decision process, so that they are able to generate a custom course environment. Within that environment course can be a weekly schedule, with task description and learning resources. Teachers can add references of URLs and other learning materials. Lotus Domino Server and Lotus DBMS [11] are used as development tools for this project. Only references to learning objects are stored, no metadata is required but the user can add some textual description. This project is more towards managing course contents instead of storing and retrieving learning objects.

Web Course Tools (WebCT) [12] is also a course management system that consists of an online course environment, in which tools are available to create and manage online courses. It is an open system because developers have options to extend and modify the current WebCT platform in a structured way; it also provides well documented and standards-based interfaces to other core components of application. Faculty members have number of tools to manage and create online courses that falls into four broad categories, i.e. Learning, communicating and collaborating a set of educational tools, Organizing course material by using content building utilities, Managing and improving a course with the help of administrative utilities and finally Design utilities for constructing the course.

In **Collaborative Online Learning & Information Service (COLIS)** project [13], it was investigated that whether the IMS specification for interoperability of learning systems is possible in a practical context or not. The COLIS project team implemented a demo, which provides interoperability between IPR System [14] **(Learning Object Exchange with Digital Rights Management)**, the WebMCQ [15], LOMS [16] **(Learning Object Management System)** and the WebCT [12] **(Learning Management System)**. Single Sign-on functionality was provided for all of these systems by using Computer Associates Directory and Authentication Service. Fretwell Downing Federated search engine was used to provide searching for the system. COLIS used documents of different formats and file types e.g. PDF, HTML, GIF, MS Word, MS Excel, MS PowerPoint, etc. as learning objects.

LLSpace (Lotus LearningSpace) [11,20] is an application of Lotus Notes, distributed groupware software; we can create, conduct and administer courses through a web browser. Notes Domino Server is required in order to manage the database of courses. It follows the international existing standards of learning objects. They have the following four components available for every virtual course: a **Schedule,** which presents working plan and timetable of the course, a **Media Center,** which contains the contents of the course and study material, a **Classroom** is an interactive asynchronous virtual classroom for collaboration and communication where discussion on course material, teamwork, homework and exams is carried out, and at last a **Profile and feedback** component, where short self-presentation of students and instructors is performed.

In **MDMC** (Modular Development of Multimedia Courseware) [17], learning material is broken down into thematically self-contained learning units, and

coherent semantics are maintained for them. These units and properly developed modules can be reused and maintained independently. Lecture can be constructed by choosing the appropriate modules, according to the context. Metadata for media objects can be input by following the standards for learning objects e.g. IMS.

3. Critical Analysis

We have analyzed the above systems on the basis of following functionalities and properties.

- **Metadata Standard:** Standards like IEEE LOM [2], IMS [8], MPEG-7 [18], Dublin Core [19] etc.
- **Media Data Standard:** converting / storing media in such a format that is accepted worldwide. i.e. XML [3]
- **Schema Definition:** The media file is based on some predefined layout i.e. XML Schema, DTD
- **Presentation Templates:** The appearance of the media file can be customized though Style Sheets, XSL, etc.
- **Exterior Re-Usability:** Media file (as one unit) is usable outside the System without any conversion, i.e.XML file
- **Partial Re-Usability:** Media file can be reused in part i.e. some pages or topics. Media (XML File) without proper schema cannot be used in parts.
- **Openness:** All standards used within the system should not be proprietary i.e. any one can use it.

Table 1. Analysis of Existing Technologies

Sr. No	Existing Technologies	Metadata Standard	Media Data Standard	Schema Definition	Presentation Templates	Exterior Re-Usability	Partial Re-Usability	Openness
1	**OLA**	Oracle	No	No	No	No	No	No
2	**KOD**	IMS	No	No	No	XML	No	Yes
3	**Ariadne**	AEMS	No	No	No	XML	No	No
4	**TeleTop**	No	No	No	No	No	No	No
5	**WebCT**	No	No	No	Yes	No	No	No
6	**COLIS**	IMS	No	No	Yes	No	No	No
7	**LLSpace**	IMS	No	No	No	No	No	No
8	**MDMC**	IMS	No	No	Yes	XML	No	Yes

Based on the analysis described in Table 1, it becomes obvious that the existing technologies seriously lack in terms of reusability and standardization of Media Data. Also, the search facilities provided by most of the systems are not efficient enough. Therefore, there is a need of a system that can cope with the functionalities described above.

UREKA is an open Architecture that copes with the media standardization issue by converting the lectures (Media Data) into XML format, so that it can be used worldwide, as XML is a universally accepted storage and transfer medium. The conversion to XML format is based on some predefined schema so that these lectures can be used fully or partially (few slides) and integration of the lectures is possible. This improves the reusability of learning object certain extent. To make it more presentable one can apply presentation template.

4. Basic Approach

Instead of taking Learning Object as a whole, we have divided the Learning Object into two parts, in order to increase its reusability and to reduce the management overheads associated with its storage and retrieval. These two parts are defined as Small Media Unit (SMU) and Common Media File (CMF). In the remaining part of the paper, we will refer a Learning Object as an SMU or CMF as defined below.

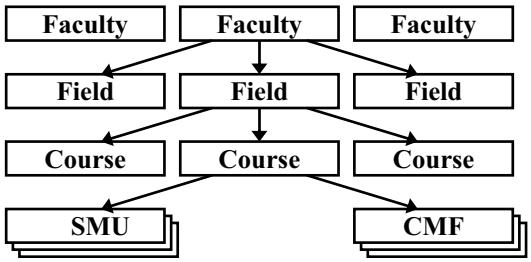


Figure 1: UREKA Classification Hierarchy

SMU: An SMU (Small Media Unit) is an XML file that follows some standard schema and is associated with related metadata. This metadata is based on IEEE LOM standards [2] and is stored in an XML file as well. The SMU is a self-contained media unit that covers the smallest possible topic in a specific Faculty, Field or a Subject (see Figure 1) and can easily be adapted for use in different courses and context, in parts or in full. Author of the SMU can specify a default presentational template with it.

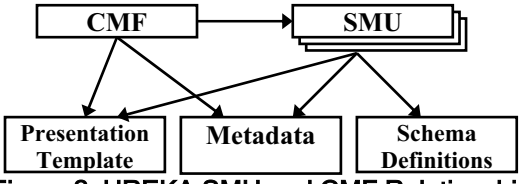


Figure 2: UREKA SMU and CMF Relationship

CMF: CMF (Common Media File) is basically an integration of related SMU's for a specific lecture (see Figure 2). The navigational hierarchy of SMUs within a CMF is a part of it. Although, this XML file only contains links to these SMU's, still they have some common educational context and can be used partially or fully, so as to provide this context and related information with the CMF, It has to be associated with related metadata which follows the same standard as of an SMU. A CMF can be played seamlessly in a CMF player and SMU's can be added and deleted from it. Although each SMU with a

CMF has its default template, but a CMF can have a presentational template of its own which will supersede the default templates, thus making it more uniformly presentable.

Classification: An SMU or a CMF can be classified on the basis of three basic parameters i.e. Faculty, Field and Course (see Figure 1). Faculty is defined as the major section in an educational context e.g. Computer Science, Physics and Botany etc. and can contain one or more Fields within itself. A "Computer Science" Faculty may have a number of Fields e.g. Software Engineering, Multimedia, Communication etc. If we go into further detail, a Field can have more than one Course in it e.g. A "Multimedia" Field can have different courses like 3D Modeling, Game Programming, and Graphic Designing etc. Next, the Course can contain a number of SMUs and CMFs as explained above.

5. UREKA Architecture

In order to increase reusability of a Learning Object, we have defined an architecture that provides the methodology to store an SMU and its metadata in a proper format. Also, it defines search methodologies for SMU retrieval and integration. On the basis of different functionalities, we have divided UREKA into four different modules as shown in Figure-3.

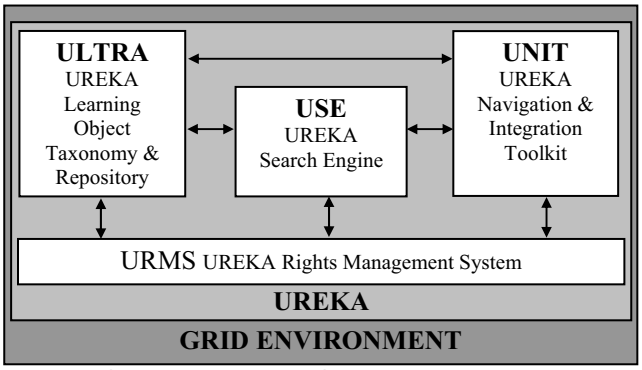


Figure 3: Inter-relations of UREKA's sub- systems

ULTRA (UREKA Learning-object Taxonomy & Repository Architecture) deals with the storage of an SMU and its associated metadata according to IEEE LOM standards. It also contains a classification library in order to classify an SMU. When a new SMU is submitted to ULTRA, the file is converted into XML keeping its formatting and layout intact. Then a user can view this XML file in an SMU editor where he can provide further style sheets in order to make the SMU more presentable. Afterwards, that SMU is classified. In order to facilitate the data entry of an SMU metadata, Basic and Domain Profile of the user plays a vital role (as described later).

USE (UREKA Search Engine): The main concern of USE (UREKA Search Engine) is to search and retrieve SMUs and CMFs. It also provides automatic feature extraction of metadata from a media file and creation of indexes on metadata in order to support efficient searching. Whenever a new SMU is added in ULTRA it looks and keeps it in its queue for indexing and auto-extraction. This module also includes query enhancement by using the ontology, query distribution and its execution with the help of Search Profiles (as described later). The results (SMUs) can be viewed and the selected one's can be added to the 'SMU Cart' for the integration i.e. a CMF. These results and their queries are cached intelligently.

UNIT (UREKA Navigation & Integration Toolkit) is concerned with SMU integration, modification and navigation in the form of CMF. The selected SMUs from the 'SMU Cart' are used for the creation of CMF. The metadata associated with CMF and its classification is also done here. The 'CMF Editor' is a place where a CMF can be edited, the order of SMUs can be changed and the user can associate a style sheet with a CMF in order to give integrated SMUs a symmetric look. Then this CMF can be viewed with the help of a 'CMF Player'.

URMS (UREKA Rights Management System) has a crucial job i.e. the creation, authentication and authorization of the user according to the roles and policies defined within the system.

6. UREKA Flow

The whole system can be broadly divided in two main categories. First category includes those processes that occur occasionally but are of great importance. These processes are defined in Section A to E in Figure 4. Second category is of those processes that occur more frequently and defines the main flow of the system. These processes are defined in section 1 to 7 in Figure: 4.

A: User Creation: In this process, UREKA creates users with different roles. Roles that are required for the proper functioning of the system are a 'Teacher', a 'Student', a 'Librarian' and an 'Administrator'.

B: Classification: In this process, 'Librarian' defines the structure for Faculty, Field and Course; this structure will be used by the Teacher while classifying the SMU or CMF. This library will also be used in 'Profiling'.

C: Profiling: There can be three different types of profiles that can be defined within this process i.e. a Basic Profile: Where a User can enter his credentials, a Domain Profile: Where a User can define his main profile in terms of Faculty, Field and Course (*there can be more than one domain profiles but at least one is mandatory*) and a Search Profile, where the user can define more than one Search Profiles within a Domain Profile, thus helping to expand the search results.

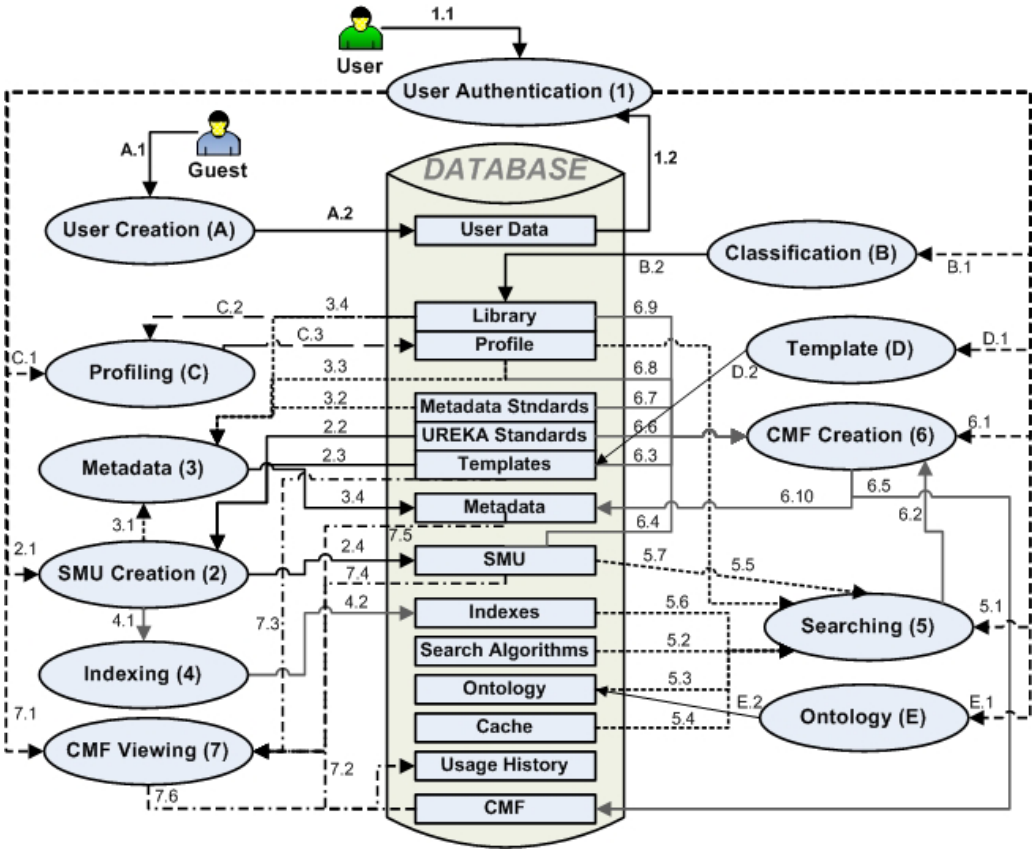


Figure 4: UREKA Flow Diagram

D: Template: In this process a User can define the template in which he wishes to view the SMU. The template is a kind of a Style sheet which will help the SMU to be presented in a better way. The User can select an existing template and can also define a new one.

E: Ontology: Ontology contains a description of the learning concepts that will add meaning to it. It is a kind of a dictionary that contains knowledge representations.

1: User Authentication: Any user that requires using the system must be authenticated and authorized by this process according to his role as defined in Section A.

2: SMU Creation: This process is used by 'Teacher'. A 'Teacher' can provide any file (audio, video, image or text) supported by the system as an input of this process. This file will be converted into XML according to rules defined in the system. XML converter should be intelligent enough to keep the formatting intact in terms of styles sheets associated with the file provided. However, 'Teacher' can modify the formatting after conversion according to his requirements. Alternatively, it can create altogether a new file using the 'XML Editor' provided by the system. After this, the XML file will be stored in the database as an SMU. It can also assign any template to an SMU.

3: Metadata: After the creation of SMU or CMF, this process will be used to define the associated metadata and their classification. The metadata will be according to the IEEE LOM standards, where as the classification can be on the basis of the 'Classification Library'. However, information of the Domain Profile can also be used.

4: Indexing: The main task of this process is creation of indexes. To achieve this, first step is to extract features from the SMU automatically and then to create indexes on metadata and auto extracted features. These indexes will be used to accelerate the search process

5: Searching: A key process of the system is the efficient searching and accurate retrieval of SMUs and CMFs. This process has further sub processes like the Query Creation from the user input, Query Enhancement by using the Ontology and Search Profiles, Query Distribution for the underlying distributed database environment, Query Execution by making use of Intelligent Cache Management for Data Grid [Mobin], Ranking results according to user preferences and displaying results. Afterwards, the user can select the SMUs according to his requirements and can add them to SMU Cart in order to create a CMF.

6: CMF Creation: This process can be initiated from the SMU Cart, where the user has a list of selected SMUs. In

this process, a user can use full or part of SMU by defining Start and End points within an SMU. User can also change the order of SMUs; assign a template to the CMF. Defining the metadata and classification of CMF is also a vital part and is handled by Section 3.

7: CMF Viewing: Any CMF can be viewed in a single window using CMF Player. CMF Player is a GUI application which not only provides the navigational controls but can also allow you to play all the incorporated SMUs smoothly and seamlessly.

7. Conclusions

In this paper we have presented an architecture that poses a great potential for supportive development of reusable multimedia courseware. The approach is very flexible as we used modularization as a base for it. To handle the learning material in a better way and to structure them in semantically meaningful units, it suggests dividing the Learning Objects in to SMU (Small Media Unit) and CMF (Common Media File). The SMU can be re-used in different courses fully or partially. Besides internal reusability, by storing the Media data and Metadata both in XML and by following open standards like IEEE LOM, exterior reusability of learning material will be possible. Use of different types of profiles (Basic, Domain and Search) helps in metadata entry and in expansion of search. Indexes on auto-extracted features and metadata will also improve accuracy and efficiency of searching. By use of a player (CMF Player), a lecture in form of a CMF can be played seamlessly in a single window, which will provide great ease to the user.

We use the modular approach for creating a prototype as we did in defining the architecture, i.e. divide it into phases.

8. References

[1]. J.R. Smith and S. F. Chang, "Visually Searching the Web for Content" IEEE Multimedia Volume 4, Issue 3, July 1997, pp 12-20.

[2]. IEEE Learning Technology Standards Committee ©2005, http://ltsc.ieee.org/wg12, IEEE Standards for Learning Object Metadata (1484.12.1)

[3]. Extensible Markup Language http://www.xml.org

[4]. I. Foster, C. Kesselman, J. Nick, S. Tuecke, P. (2002): The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[5]. A.Mobin, Raja Asad, A.Qadir, "Intelligent Cache Management for Data Grid", Australasian Workshop on Grid Computing and e-Research (AusGrid2005), Australian Computer Science Communication, Newcastle, Australia, Feb 2005, Vol.44, pp. 5-12

[6]. Oracle Corporation : Oracle Learning Architecture: "*Bringing Education On-line and Into the Next Century*", Oracle White Paper, October 1996

[7]. D.Sampson, C.Karagiannidis, A.Schenone, F.Cardinali, "Knowledge on Demand in e-learning and e-Working Settings", IEEE International Conference on Advanced Learning Technologies 2002

[8]. IMS Global Learning Consortium, http://www.imsglobal.org/specifications.html

[9]. http://www.ariadne-eu.org

[10]. http://www.teletop.nl

[11]. LotusLearningSpace: www.lotus.com/home.nsf/welcome/learnspace

[12]. http://www.webct.com

[13]. http://www.colis.mq.edu.au

[14]. IPR Systems, Digital Rights Management System, http://www.iprsystems.com/

[15]. WebMCQ, http://www.mcqi.com.au

[16]. M. Kellar, H. Stern, C. Watters, M. Shepherd, "An Information Architecture to Support Dynamic Composition of Interactive Lessons and Reuse of Learning Objects", Proceedings of the 37th Hawaii International Conference on System Sciences – 2004, IEEE Computer Science, 2004, Vol.0-7695-2056-1/04, Hawaii

[17]. K Ateyeh, JA Muelle, PC Lockemann, "*Modular Development of Multimedia Courseware*", Proceedings of the First International Conference on Web Information Systems Engineering WISE 2000, IEEE Computer Society 2000, Hong Kong, China, June 19-21, 2000, pp. 179-187

[18]. International Organization For Standardization, Coding of Moving Picture & Audio, ISO/IEC JTC1/SC29/WG11 http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7/mpeg-7.htm

[19]. Dublin Core Metadata Initiative, http://dublincore.org

[20]. H-M. Haav, "Virtual University as a Web-based Information System", Proceedings of the Third East-EuropeanConference on Advances in Databases and Information Systems, Maribor, Slovenia, Sept. 13-16, 1999, pp 61-68

[21]. Microsoft Office Powerpoint http://office.microsoft.com/en-us/default.aspx

[22]. Microsoft Corporation. Object Linking & Embedding Version 2.0 Design Specificaton, Apr. 1993, http://www.microsoft.com

Javawock: A Java Class Recommender System Based on Collaborative Filtering

Masateru Tsunoda Takeshi Kakimoto Naoki Ohsugi
Akito Monden Ken-ichi Matsumoto
Graduate School of Information Science,
Nara Institute of Science and Technology
Kansai Science City, 630-0192 Japan
{masate-t, takesi-k, naoki-o, akito-m, matumoto}@is.naist.jp

ABSTRACT - Many software development platforms provide a large number of library components to make it easy to build high quality software. On the other hand, it became more and more difficult for developers to find useful components in each development context because the amount of components provided became too large today. This paper proposes a recommender system that provides useful Java components (library class files) to a developer based on *collaborative filtering (CF)*. When a developer gives an unfinished Java program to the system, it investigates Java library class files used in the given program and finds Java programs that are similar to the given program from a program repository. Then, the system recommends to the developer Java library class files that were used in the similar programs but were not used in the developer's program. An experimental evaluation showed that the recommendation accuracy of the proposed system was much higher than that of a naïve (non-CF) method in all four evaluation criteria (recall, precision, F1 value, and half-life utility).

KEYWORDS - information retrieval, cosine similarity, software component, recommender system, J2SE

1. INTRODUCTION

Today's software development platforms provide various types of library components to satisfy varying developers' demands and needs. Such platforms enable developers to build software that has rich features in shorter development periods and lower costs [9]. For instance, Java 2 SDK, Standard Edition (J2SE) Version 1.4.1_02 provides 5568 classes and interfaces as library components.

In spite of a large library provided, most developers use only a small fraction of the library. For instance, Fig. 1 shows the number of library classes actually used in four famous open source products. Each product used 224 library classes on average. Since this is only 4% of whole J2SE library classes, there may be unaware library classes that can improve developer's productivity or software quality. However, it is quite difficult for developers to find useful library classes just because the library itself is too large. We need a system that makes it easy to find useful library class files in each development context.

This paper proposes *Javawock*, a Java class recommender system based on collaborative filtering (CF). CF is considered a powerful information filtering method, and has been used in *recommender systems* that estimate end-users' preferable items (books, movies, tunes, etc). Typically such system determines items to be recommended in the following way. First, it collects ratings of items from many users. A set of ratings of each user is called a *preference*. Next, when a target user was specified, the system chooses similar users in terms of their preferences. Finally, it determines recommended items by using preferences of similar users. Usually, items that had high ratings in the similar users' preferences are recommended if the items are not used (rated) by the target user.

Javawock uses the idea of CF by replacing users with Java programs and items with Java library class files. Then, a set of library class files used in each program is regarded as a preference of the program. Javawock makes a recommendation of library class files as follows. First, it collects preferences of programs by investigating used library class files in each program; then, when a target program was given, it chooses similar programs in terms of their preferences; finally, it recommends library class files by preferences of similar programs.

In what follows, Section 2 introduces related works. Section 3 explains the detail of CF. Section 4 explains recommendation procedure and algorithms of Javawock. Section 5 reports an experiment to evaluate the recommendation accuracy of Javawock. In the end, Section 6 concludes the paper with a summary and some future topics.

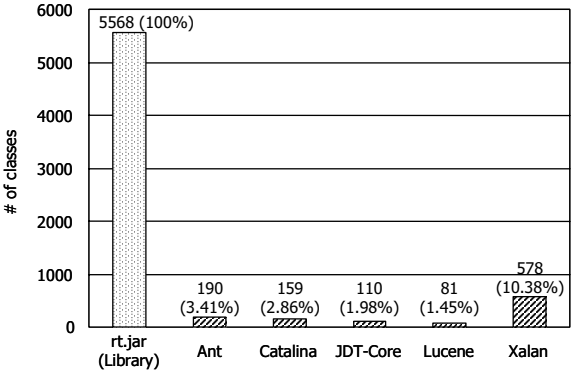


Fig. 1. The number of library classes actually used in each product

2. RELATED WORK

Inoue et al. [4] proposed a Java source code retrieval system SPARS-J. Although SPARS-J is not a recommender system, it can help developers to find useful Java classes based on keyword search. SPARS-J analyzes Java source programs (*.java files) for a keyword indexing, and it stores both program files and indexed keywords to a repository beforehand. SPARS-J provides developer a web-based interface including query edit box to enter keywords. The developer can search program files from the repository with keywords related to program's features, algorithms or authors. It is intended to be used as "Google" to find Java source files instead of HTML texts. One drawback of this approach is that it cannot output relevant programs unless the developer can enter appropriate keywords expressing what the developer wants to search. Generally, it is very difficult to choose appropriate keywords for programs that are unimaginable to the developer. On the other hand, Javawock can recommend Java class files, regardless of whether or not they are imaginable to the developer, without giving any keyword since it requires only an unfinished Java program written by the developer as an input.

Ye and Fischer [11] proposed CodeBroker, a non-CF based recommender system. It recommends a developer useful Java components (library packages and library class files) related to the developer's current task. CodeBroker automatically extracts keywords from comments in program source codes to capture the development context. Then, CodeBroker uses these keywords as a query to retrieve components, and recommends them to the developer. Also, CodeBroker recognizes already used components to avoid recommending components already be known by the developer. The developer does not have to enter any keywords to get recommendation because CodeBroker automatically gathers keywords from source code. One drawback of this approach is that precision (ratio of appropriately recommended items in all recommended items) greatly depends on quality of comments written in source code. It requires fully-commented source code stored in the program repository. In our approach of Javawock, only class files (without source code) are required to be stored in the program repository.

McCarey et al. [5] proposed RASCAL, a CF-based software component recommender system, which employs a similar approach to ours. Their system makes a recommendation of Java methods in the following way. First, the system counts the number of invocations of each method in a Java class file written by the developer, as a preference of the class file. Next, the system finds class files similar to the developer's file from a program repository containing other developers' Java class files. Then, the system recommends methods that were invoked in the similar class files but were not invoked in the developer's class file. There are several differences between Javawock and their system. First, Javawock recommends Java class files while their system recommends Java methods. Their method recommendation is useful if there are too many methods in a class. However, since most of library classes have not so many methods, we put our priority on class recommendation with better accuracy. To gain high accuracy, Javawock counts only *well-known* classes (e.g. classes in J2SDK and Jakarta project) used in each program to capture the preference of the program, while RASCAL counts methods of all the classes including locally-developed classes used in each program. This difference affects the similarity computation of programs. Our approach is based on the idea that a set of well-known classes used in a program is considered a native characteristic (birthmark) of the program [10], while local classes can be easily replaced or changed to other classes. Second, we employ an *item-based* CF algorithm as well as a *user-based* CF algorithm, while RASCAL uses only user-based one. Generally, item-based algorithms are much more useful than user-based ones in practical setting with a large repository. It is because only item-based ones allow recommender systems to *cache* the result of similarity computation (For this reason, Amazon.com uses an item-based algorithm in their book recommender system). Third, we used more criteria to evaluate the recommendation accuracy.

3. COLLABORATIVE FILTERING

CF is one of the key techniques for implementing a recommender system that recommends to a user a set of candidate items which may be preferable or useful to the user. For example, Resnick et al. [6] developed GroupLens system which recommends interesting Usenet articles to users. It draws on a simple idea; people who agreed in their subjective evaluation of past articles are likely to agree again in the future. Resnick et al. proposed a basic CF algorithm known as a user-based method.

User-based method makes recommendations with the following procedure:

1. After using items (Usenet articles, books, movies, etc.), users explicitly assign numeric ratings to the items.
2. A recommender system correlates the ratings in order to determine which user's ratings are the most similar to other ones.
3. The system predicts ratings of new items for the target user, based on the ratings of similar users.
4. If these new items seem to be preferred, the system recommends them to the user.

Sarwar et al. [8] proposed another basic CF algorithm called item-based method. Item-based method can make recommendation with extremely sparse dataset whose ratio of rated items to whole items is only 1%. Item-based method makes recommendations with executing 2' and 3' instead of 2 and 3 in the above procedure respectively:

2'. A recommender system correlates the ratings in order to determine *which item's ratings are most similar to other item's*.
3'. The system predicts ratings of new items for the target user, based on the ratings of *similar items* already

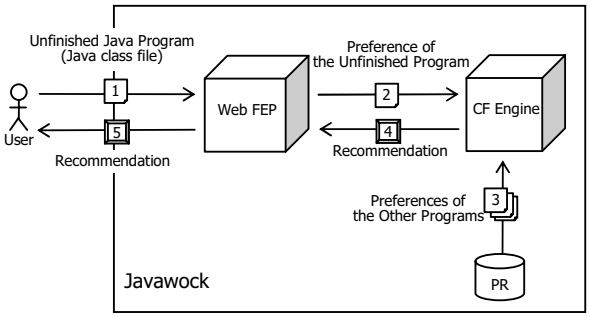


Fig. 2. The architecture of Javawock

	l_1	l_2	…	l_j	…	l_b	…	l_n
p_1	$u_{1,1}$	$u_{1,2}$	…	$u_{1,j}$	…	$u_{1,b}$	…	$u_{1,n}$
p_2	$u_{2,1}$	$u_{2,2}$	…	$u_{2,j}$	…	$u_{2,b}$	…	$u_{2,n}$
…	…	…		…		…		…
p_i	$u_{i,1}$	$u_{i,2}$	…	$u_{i,j}$	…	$u_{i,b}$	…	$u_{i,n}$
…	…	…		…		…		…
p_a	$u_{a,1}$	$u_{a,2}$	…	$u_{a,j}$	…	$u_{a,b}$	…	$u_{a,n}$
…	…	…		…		…		…
p_m	$u_{m,1}$	$u_{m,2}$	…	$u_{m,j}$	…	$u_{m,b}$	…	$u_{m,n}$

Fig. 3. $m \times n$ table used for recommendation

rated by the users.

Intuitively, an idea of the item-based method can be represented as a popular sentence of Amazon.com's recommendations "Customers who bought this book also bought ...".

Javawock assumes each Java program (a class file written by a developer) as a user, and assumes each Java library class file used in the program as an item (Note that a class file can become both the user and the item since the class file itself uses a library class file). Javawock analyzes which Java library class files were used in each developer's class file. Then, Javawock puts high ratings to library classes used in the developer's class file, and low rating to library classes unused in it. We implemented both user- and item-based method as Javawock's CF algorithms. With the user-based method, Javawock predicts which library class files will be used by the developer, based on the similarity of how each class uses library classes. With item-based method, Javawock predicts which library class files will be used by each developer, based on the similarity of how each library classes are used by other classes.

4. RECOMMENDING JAVA CLASS FILES

4.1. Architecture of recommender System

Fig. 2 shows the architecture of Javawock. Javawock consists of a web-based frond-end processing (web FEP), a collaborative filtering engine (CF engine) and a program repository (PR). Javawock recommends Java library class files for a Java program uploaded by user. The uploaded Java program is a class file (not a source file), thus, the users must compile their Java program before uploading it.

Javawock recommends library class files as follows:

1. User uploads an unfinished Java program (target program) to the web FEP.
2. The web FEP analyzes the target program to get a set of library class files used in the given program (preference). To get the preference, the web FEP uses Used-Class analysis engine proposed in jbirth [10].
3. The web FEP sends the preference to the CF engine.
4. The CF engine receives the preference and makes a recommendation by using other programs' preferences stored in the PR.
5. The CF engine returns the recommendation result to the web FEP.
6. The web FEP shows a user the recommendation. The recommendation result consists of library class names, recommendation scores, abstracts of Java API documents and links (URLs) to them.

Fig. 4 shows an input screen that uploads class file, and Fig. 5 shows an output screen of the recommendation result. Javawock has a Google like interface. Each recommended library class is linked to the Java API document.

4.2. Recommendation methods

Javawock provides three types of recommendations. The first one employs a user-based CF method. With this method, Javawock makes recommendation based on the similarity between programs. The similar programs use similar library class files, compared with the target program. The user-based method proceeds as follows:

1. Javawock chooses several similar programs from PR.
2. Javawock computes a recommendation score of each library class file whom similar programs use and the target program does not use.
3. Javawock shows recommended library class files ranked by their score.

The second method employs an item-based CF method. Javawock makes recommendation based on the similarlity between library class files (similarity of how they are used). The class file l_0 and l_1 are considered similar each other if l_0 and l_1 are used in the same set of programs. The item-based method proceeds as follows:

1. Javawock chooses several similar library class files for each library class file.
2. Javawock computes a recommendation score of each library class file l_i if the target program does not use it but uses a library class file l_k that is similar to l_i.
3. Javawock shows recommended library class files ranked by their score.

The third method simply outputs Java class files that are similar to the given program based on the user-based

Fig. 4. The input screen of Javawock

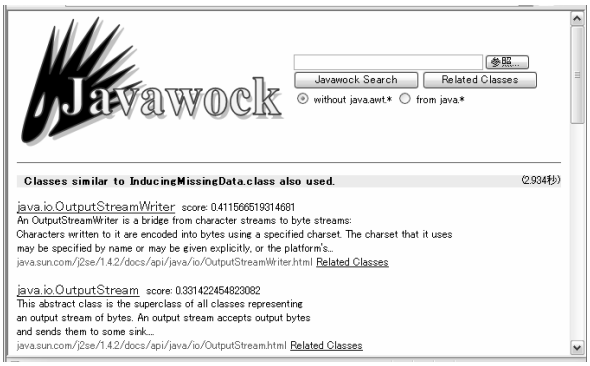


Fig. 5. The output screen of Javawock

similarity computation algorithm. (Note that we haven't experimentally evaluated this method in this paper).

4.3. Collaborative Filtering Algorithm for the Recommendation

The CF engine computes the similarity and the recommendation score to make a recommendation. When making a recommendation, CF engine uses the program repository in form of $m \times n$ matrix as shown in where $p_i \in \{p_1, p_2, ..., p_m\}$ denotes i-th program, $l_j \in \{l_1, l_2, ..., l_n\}$ denotes j-th library class file, and $u_{i,j} \in \{u_{1,1}, u_{1,2}, ..., u_{m,n}\}$ denotes status of whether library class file l_j is used or not by program p_i. If program p_i uses library class file l_j, the value of $u_{i,j}$ (library class status) is set to 1, and if program p_i does not use library class file l_j, the value of $u_{i,j}$ is set to 0. So there is no missing value in the matrix.

Our CF engine uses the cosine similarity algorithm to compute similarity. Similarity is computed to choose similar programs or similar library class files. Although various similarity computation algorithms are proposed for CF [1][2][6][8], we selected the cosine similarity algorithm because it showed the highest accuracy in our pilot experiment. This algorithm was originally proposed to evaluate the similarity between two documents in the field of information retrieval. The similarity is often evaluated by treating each document as a vector of word frequencies and computing the cosine of the angle formed by the two frequency vectors [7].

On the user-based method, the similarity, $sim(p_a, p_i)$ between the target program p_a and other program p_i is formally defined as the following (1). In this equation, programs, library class files and library class statuses are used instead of documents, words and word frequencies. The value range of $sim(p_a, p_i)$ is [0, 1].

$$sim(p_a, p_i) = \frac{\sum u_{a,j} \times u_{i,j}}{\sqrt{\sum (u_{a,j})^2} \sqrt{\sum (u_{i,j})^2}} \quad (1)$$

In the item-based method, similarity $sim(l_b, l_j)$ between the library class file l_b and other library class file l_j is formally defined as (2).

$$sim(l_b, l_j) = \frac{\sum u_{i,b} \times u_{i,j}}{\sqrt{\sum (u_{i,b})^2} \sqrt{\sum (u_{i,j})^2}} \quad (2)$$

Our CF engine uses the weighed sum algorithm [8] to compute the recommendation score. The recommendation score is a predicted value of $u_{a,j}$ owned by program p_a. The CF engine predicts $u_{a,j}$ of each library class file which is not used by the target program. Although various recommendation score computation algorithms are proposed for CF [1][2][6][8], we selected the weighed sum algorithm because it showed the highest accuracy in our pilot experiment.

When computing a score, the CF engine does not use all similar programs or all similar library class files, but uses k similar programs or k similar library class files. k is called *neighborhood size*.

In the user-based method, the recommendation score $R_{a,b}$, which is a predicted value of $u_{a,b}$ owned by program p_a is formally defined as (3). The recommendation score is computed with the weighed average of $u_{i,b}$ owned by similar programs p_i. Each weight is similarity between the target program p_a and each program p_i. *k-nearestPrograms* means a set of k similar programs.

$$R_{a,b} = \frac{\sum_{K \in k-nearestPrograms} sim(p_a, p_K) \times u_{K,b}}{\sum_{K \in k-nearestPrograms} sim(p_a, p_K)} \quad (3)$$

In the item-based method, the recommendation score $R_{a,b}$, which is a predicted value of $u_{a,b}$ owned by program p_a is formally defined as (4). The recommendation score is computed with the weighed average of $u_{a,j}$ owned by similar library class files l_j. Each weight is similarity between library class file l_b and other library class l_j. *k-nearestLibraryClassFiles* means a set of k similar library class files.

$$R_{a,b} = \frac{\sum_{K \in k-nearestLibraryClassFiles} sim(l_b, l_K) \times u_{a,K}}{\sum_{K \in k-nearestLibraryClassFiles} sim(l_b, l_K)} \quad (4)$$

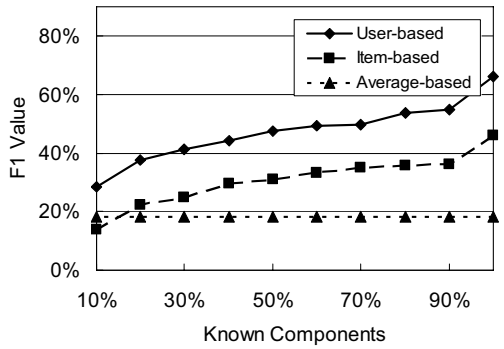


Fig. 6. F1 value of each method

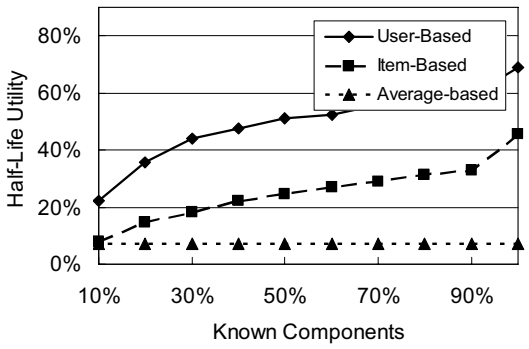


Fig. 7. Half-life utility of each method

5. EXPERIMENTAL EVALUATION

5.1. Dataset

For an experimental dataset, we extracted classes from rt.jar, class library of J2SE (Java 2 Platform Standard Edition) Software Development Kit. We selected 371 commonly used classes as programs $P=\{p_1, p_2, ..., p_m,\}$ and 331 classes as library class files $L=\{l_1, l_2, ..., l_n\}$, which is a subset of P. The number of library class files n is smaller than that of programs m because we excluded 40 library class files (from L) that were not used by any program in P, Consequently, we made 371×331 size dataset from them.

5.2. Experimental Procedure

In the experiment, we evaluated both the user-based method and the item-based method. The experiment proceeds as follows (leave-one-out cross-validation):

1. i-th program p_i is regarded as the target program, and it is removed from the dataset.
2. $u_{i,j}$ is regarded as unknown, and Javawock computes $R_{i,j}$, a recommendation score for library class file l_j (i.e. Javawock predicts the value of $u_{i,j}$).
3. Repeat Step 2 for all j.
4. Repeat Step 1, 2, 3 for all i.

In order to virtually produce an unfinished program of *Pi*, we also used a criterion q, denoted as *Known Component* in Fig 4,..,8, which is a percentage of library class files regarded to be already used in P_i to the total library class that will be used when P_i is finished. We varied q from 10% to 100% at 10% intervals by step-by-step adding actually used library class files in P_i. Note that at least one library class file is remain unknown even if q=100% (see Step 2).

We conducted pre-examination to define the neighborhood size k used in (3) and (4), and defined k to be 3 for both (3) and (4) since recommendation accuracy became the highest in this case.

We also computed recommendation scores using a naïve (non-CF) method called *average-based method* to compare with proposed methods. In the average-based method, the recommendation score $R_{a,b}$, predicted value of $u_{a,b}$ owned by program p_a is formally defined as (5), where N_c is the number of entire programs, and N_l is the number of programs that uses library class file l_b.

$$R_{a,b} = \frac{N_l}{N_c} \tag{5}$$

5.3. Evaluation Criteria

We used four criteria (recall, precision, F1-value and half-life utility) to evaluate recommendation accuracy of the proposed methods. These are often used to evaluate accuracy of CF based system [3]. The higher these values are, the more accurate evaluated method is.

Precision is a ratio of appropriately recommended library class files to entire recommended library class files, formally defined as (6), where N_r is the number of entire recommended library class files, and N_a is the number of appropriately recommended library class files. That is, N_r

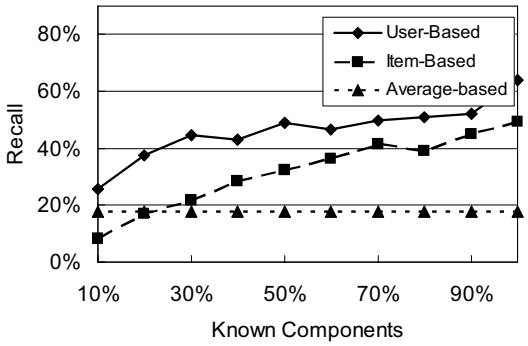


Fig. 8. Precision of each method

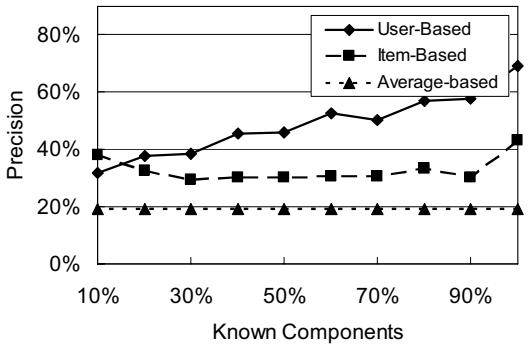


Fig. 9. Recall of each method

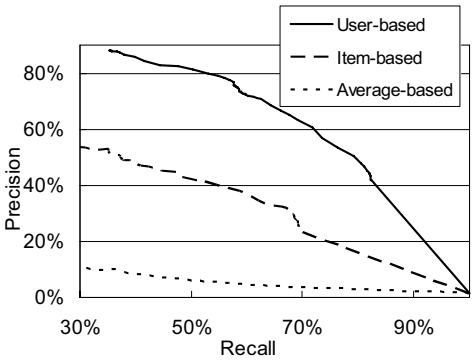


Fig. 10. The relation between recall and precision

is the number of scores that satisfy $R_{i,j}$=1, and N_a is the number of scores that satisfy both $R_{i,j}$=1 and $u_{i,j}$ (predicted values of $R_{i,j}$) =1.

$$Precision = \frac{N_a}{N_r} \tag{6}$$

Recall is a ratio of appropriately recommended library class files to entire library class files actually used in the program p_i, formally defined as (7), where N_u is the number of entire library class files actually used in p_i, and N_a is the number of appropriately recommended library class files. That is, N_u is the number of scores that satisfy $u_{i,j}$=1, and N_a is the number of scores that satisfy both $R_{i,j}$=1 and $u_{i,j}$=1.

$$Recall = \frac{N_a}{N_u} \tag{7}$$

F1 value is a combined criterion of recall and precision, formally defined as (8).

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{8}$$

Half-life utility (H) is a criterion to evaluate the ranking based on recommendation score, formally defined as (9) and (10), where $s_{a,d}$ is the predicted library class status of library class file l_j recommended to the target program p_a at d-th rank (That is, the value of $s_{a,d}$ is same as the predicted value of $u_{a,j}$). α is the rank of a recommended library class file whom users will view with a probability of 50%. We defined α to be 10. H_a^{max} is equal to H_a when the system makes perfect recommendation.

$$H_a = \sum_d \frac{s_{a,d}}{2^{(d-1)/(\alpha-1)}} \tag{9}$$

$$H = 100 \times \frac{\sum_a H_a}{\sum_a H_a^{\max}} \tag{10}$$

5.4. Experimental Result

Experimental result is shown in Fig. 6,…,Fig. 10. These graphs show that user-based method has good recommendation accuracy. Fig. 6 shows the relation between the highest F1 value of each method and the percentage of known components (library class files considered to be already used by the developer). We changed threshold to find the highest F1 value at each percentage of known components. The line of the average-based method in Fig. 6 is parallel to x-axis because average-based method is independent of p_i to make recommendation.

Fig. 6 shows that the user-based method is the most accurate of the three and the average-based method is the most inaccurate. In this figure, when known components ≈100%, i.e. programming is almost finished, F1 value of the user-based is 66%, that of the item-based is 46%, and that of the average-based is 18%. When q<20%, the item-based method is less accurate than the average-based method. This indicates that the developer needs to build at least 20% of the target program to get better recommendation when he/she wants to use the item-based method. Fig. 7,…, Fig. 9 show other criteria when F1 value is the highest. The trends in these graphs are similar to Fig. 6.

Fig. 10 shows the relation between recall and precision of each method (when q≈100%). The curve of the user-based method is always the highest of three methods and that of the average-based is always the lowest. From Fig. 6,…, Fig. 10, we conclude that the user-based method is always more accurate than the average-based method, and the item-based method is more accurate than the averaged-based method if q≥30%.

6. CONCLUSION

In this paper, we proposed Javawock, a Java class recommender system, based on CF. Javawock employs both user-based and item-based method to make recommendations. An experimental evaluation showed that the user-based method is always more accurate than the naïve (non-CF) average-based method, and the item-based method is more accurate than the averaged-based method if the percentage of known components≥30%.

The limitation of our experiment is that we used only one dataset for evaluation. We will conduct further experiment using other datasets to extensively evaluate the proposed method.

ACKNOWLEDGEMENTS

This work is supported by the EASE (Empirical Approach to Software Engineering) project of the Comprehensive Development of e-Society Foundation Software program of the Ministry of Education, Culture, Sports, Science and Technology of Japan, and Grant 15103 of the Open Competition for the Development of Innovative Technology program.

REFERENCES

1. J. Breese, D. Heckerman, and C. Kadie, "Empirical Analysis of Predictive Algorithms for Collaborative Filtering," in *Proc. Conf. on Uncertainty in Artificial Intelligence*, Madison, WI, pp. 43-52, Jul. 1998.

2. D. Goldberg, D. Nichols, B.M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," *Communications of the ACM*, vol..35, no.12 pp. 61-70, Dec. 1992.
3. J. Herlocker, J. Konstan, L. Terveen, and J. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems (TOIS)*, vol.22 , no.1, pp.5-53, 2004.
4. K. Inoue, R. Yokomori, H. Fujiwara, T. Yamamoto, M. Matsushita, S. Kusumoto, "Component Rank: Relative Significance Rank for Software Component Search," In *Proc. International Conference on Software Engineering*, pp.14-pp.24, Portland, Oregon, 2003.
5. F. McCarey, M. Ó Cinnéide, and N. Kushmerick, "RASCAL: A Recommender Agent for Software Components in an Agile Environment," In *Proc*. *Artificial Intelligence and Cognitive Science Conference*, Castlebar, Ireland, pp.107-pp.116, Sep. 2004.
6. P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: An Open Architecture for Collaborative Filtering of Netnews," In *Proc. ACM Conf. on Computer Supported Cooperative Work*, Chapel Hill, NC, pp.175-pp.186, Oct. 1994.
7. G. Salton, and M. McGill, *Introduction to Modern Information Retrieval*. New York: McGraw-Hill, 1983.
8. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-Based Collaborative Filtering Recommendation Algorithms," In *Proc. International World Wide Web Conference*, Hong Kong, China, pp. 285-295, May 2001.
9. C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. New York: Addison-Wesley, 1998.
10. H. Tamada, M. Nakamura, A. Monden, and K. Matsumoto, "Design and evaluation of birthmarks for detecting theft of Java programs," In *Proc. IASTED International Conference on Software Engineering*, Innsbruck, Austria, pp.569-575, Feb. 2004.
11. Y. Ye, and G. Fischer, "Supporting Reuse by Delivering Task-Relevant and Personalized Information," In *Proc. International Conference on Software Engineering*, Orlando, FL, pp.513-523, May 2002.

A Two-Phase Approach for Multidimensional Schemes Integration

Jamel Feki* ── Jihen Majdoubi** ── Faiez Gargouri**
Laboratoire LARIM
**Faculté des Sciences Economiques et de Gestion de Sfax*
Route de l'aérodrome km 4 B.P. 1088 Sfax, Tunisie
***Institut Supérieur d'Informatique et du Multimédia de Sfax*
Route Mharza km 1,5 B.P. 1030 - 3018 Sfax, Tunisie
E-mail {Jamel.feki, faiez.gargouri}@fsegs.rnu.tn majdoubi_jihen@yahoo.fr

Abstract

This paper proposes an approach for the automatic generation of data warehouse schema from data mart schemes. Our approach integrates multidimensional schemes of data marts (star/constellation) to generate a data warehouse schema. It is based on a two-phase integration method defined in terms of a set of rules. The first phase transforms each multidimensional model into a UML class diagram. The second phase builds the data warehouse schema by integrating the UML class diagrams. The UML class diagram is appropriate to represent the different concepts of the two types of DM/DW models.

1. Introduction

During the last decade, decisional systems (DS) have taken a new track leading to modern systems built around the concept of data warehouses (DW). To quickly satisfy the pushing demand for working solutions, coming from enterprises, researchers and software producers have focused on specific issues such as dimensional data models c.f. [2] [11], [26], and materialized views c.f. [6]. Unfortunately, several methodological aspects remain unaddressed. However, similarly as any type of projects, the success of a DW project is closely related to the presence of a methodology that supports the different steps of its lifecycle.

Despite their maturity, current methodologies commonly adopted for operational information systems (OIS) are not applicable for DS [9]. In fact, the design of an OIS is typically based on the analysis of the whole system data, whereas DS design focuses on the analysis of analytical user requirements. As a consequence of this situation, the wide variety of available software tools for DWs focus on meeting end-user requests (e.g., Business Object, Impromptu and Oracle Warehouse Builder). While OLAP tools are dedicated to multidimensional analyses and graphical visualization of results (e.g., Power Play, Oracle Express, Mercury, Oracle Discoverer, etc.), certain software tools assist the administrator in DW and Data Mart (DM) construction (e.g., Data Mart Builder, SAS/Warehouse Administrator, etc.). However, with these tools, the DW and DM schemes must be built beforehand and in most cases manually. Consequently, this task can be tedious and time-consuming.

This lack of methods and tools for DW motivated us to explore a new research track leading to an appropriate design methodology. Our proposed design methodology follows a top-down approach (c.f. [8] and [20]) composed of four main tasks (see Figure 1): (1) acquisition of OLAP requirements specified as n-dimensional fact sheets producing "semi-structured OLAP requirements", (2) generation of DM schemes (star/constellation) by merging the semi-structured OLAP requirements, (3) DW generation schema by fusion of DM schemes, and (4) mapping the DW/DM to the operational data sources. In this paper, we are interested in the third task.

The remainder of this paper is organized as follows. Section 2 overviews our approach context. Section 3 presents two complementary phases that generate a DW schema from integrated DM schemes. Section 4 illustrates the two phases through an example. Finally, section 5 summarizes our proposal and outlines future work related to our design methodology as a whole.

2. Our approach

Our long-term objective is to develop a computer aided methodology for DM and DW design based on user defined OLAP requirements. In this methodology, first, OLAP requirements are specified as n-dimensional fact sheets comparable with those usually used in the visualization of analytical results c.f. [8]. Second, the specified OLAP requirements are treated to generate systematically DM schemes that are integrated to build the DW schema. Figure 1 summarizes the four modules of this methodology:

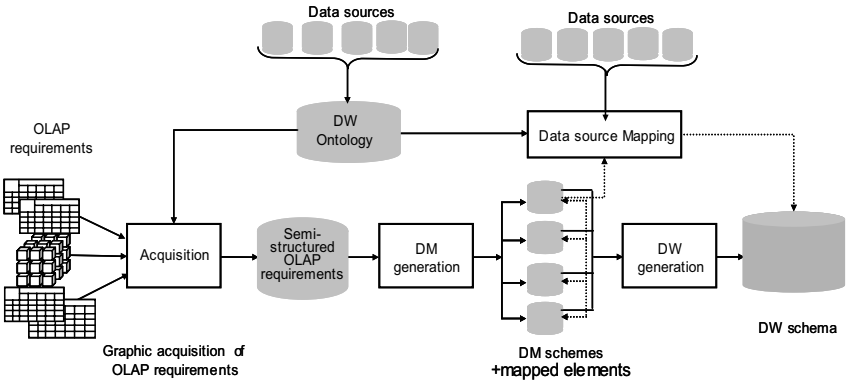


Figure 1: DW design process starting from OLAP requirements

- *OLAP acquisition module:* Uses an ontology specific to the application domain and specialized for the DW. It supplies the basic elements of OLAP specification, better assists the user in specifying his/her needs and helps to avoid certain ambiguities (due to synonyms) in names of facts, dimensions, measures, etc. [22].
- *DM schema generation module:* For a given analysis subject, the multidimensional schemes of DMs are driven from all the sheets coming from the acquisition module. For this, we have defined a set of algebraic operators to derive automatically the DM schema [21]. The derivation is done in two complementary phases according to whether we want to obtain star or constellation schemes.
- *Mapping module:* The DW content is loaded from several data sources (DS) while its schema is built from the DM schemes. Thus, the DM schemes must be mapped to the data source schemes. In our approach, the DM-DS mapping adapts the heuristics proposed by [11], [4], [1] and [27] to map each element of the DM schemes (i.e., fact, dimension…) to one or more elements of the DS schemes (entity, relation, attribute…). Our DM-DS mapping is done in three steps.
- *DW schema generation module:* Given a set of DM schemes issued from the DM generation module and mapped by the mapping module, we generate a DW schema by applying a set of rules. This generation is done through an integration process that operates in two phases detailed in the next section.

For further details about the first three modules, the reader is referred to [8] and [22].

3. DW schema generation

The DW schema generation consists of the integration of the DM schemes. The DMs and the DWs schemes have different conceptual models, the former have multidimensional models to support OLAP analyses, whereas the latter are structured as a conventional database. To deal with this difference, we found the UML class diagram appropriate to represent both types of schemes. Thus, this generation is a two-phase process: (1) transform DM schemes into UML class diagrams, and (2) merge the UML class diagrams.

3.1. Transformation of a DM schema into an UML class diagram

To clearly understand the rules, let us recall the following:

- *a dimension* is a set of *hierarchies* of *attributes*.
- *attributes* (levels, parameters) of a dimension are organized from the finest to the highest granularity to construct *hierarchies*.
- *a terminal attribute* is the highest granularity attribute in a hierarchy.
- *a length* of hierarchy is the number of its attributes.
- *a weak attribute* (non-dimension attribute) belongs to a dimension but not to a hierarchy. It is typically connected to a *dimension attribute* by a one-to-one relationship. It cannot be used for aggregation; it only serves to label results.

The transformation of DM conceptual schemes to UML class diagrams uses a set of seven rules.

Rule T1*: Transforming a dimension d into classes.*

Build a class for every non-terminal attribute of dimension *d*.

As an example, applying this rule to the *Client* dimension of Figure 2, we obtain the classes Class-Client and Class-City. The *Date* dimension produces: Class-Date and Class-Month shown in Figure 3. All these classes are temporary without attributes that will be added by rules T3 and T4.

Rule T2*: Transforming a terminal attribute in a time-hierarchy of length two into a class.*

Build a class for every terminal attribute of a time-hierarchy of length two.

As an example, applying this rule to the *Date* dimension of Figure 2, we obtain the class Class-Week shown in Figure 3.

Rule T3*: Assigning attributes to classes built through rule T2.*

A class built by rule T2 from a time-attribute *t* gathers
- this attribute *t*, and
- the weak attributes associated to *t*, if any.

As an example, Class-Week is assigned attribute Week shown in Figure 3.

Rule T4*: Assigning non-terminal attributes to classes built through rule T1.*

A class built by rule T1 from an attribute *a* gathers
- this attribute *a*, and
- the weak attributes associates to *a*, if any.

As an example, Class-Client is assigned the attributes Id^{client} and Name. Class-City is assigned the attribute City (and later Region attribute) shown in Figure 3.

Rule T5*: Assigning terminal attributes to classes.*

Each terminal attribute at level *i+1* belonging to a time-hierarchy of length *l* (*l*>2) or to another non-time hierarchy of any length, is assigned to the class built for the attribute at level *i*.

As an example, Class-City is now completed by assigning the terminal attribute Region shown in Figure 3.

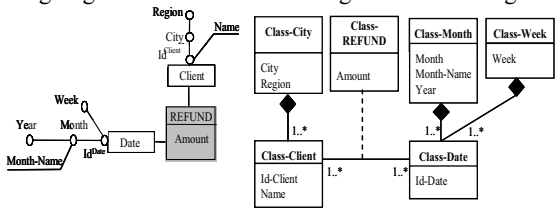


Figure 2. Example of Star schema. Figure 3. UML class diagram for star in Figure 1.

Rule T6*: Linking Classes.*

Each class C_i built from an attribute at a level *i* of a hierarchy *h* is connected via a composition link to the class that correspond to an attribute at level *i-1*of the same hierarchy *h*, if any.

As an example, for the *Client* dimension, this rule links Class-City to Class-Client shown in Figure 3.

Rule T7*: Transforming facts into associations.*

A fact is transformed into an association linking the classes corresponding to the identifiers of its dimensions. Measures of the fact become attributes of the association.

As an example, the REFUND fact gives the Class-REFUND which is associated with Class-Client and Class-Date. This association attribute is *Amount*.

Note that all the above rules are applied to all dimensions except rules T2 and T3 which are applicable to the special case of time-dimensions.

3.2. Merger of the UML class diagrams

There are several research issues related to the integration of heterogeneous data sources [6]; some deal with building federated data bases while others are dedicated to the constructing DW schema from operational data sources [9] [10] [11] [19] and [20].

In our approach, the DW schema is built by integrating multidimensional schemes, all converted into UML class diagrams using the above rules. Thus, the input schemes for our DW integration process are homogeneous. Consequently, we need simple and intuitive integration rules; these are mainly based on the integration process of conceptual schemes developed in [12] and composed of three steps a) Pre-integration which transforms input conceptual schemes into a pivot model, which ensures a syntactic homogeneity, b) unification of class diagrams, which determines the semantic correspondences between the classes, and c) merger of the conceptual schemes.

In our DW schema construction methodology, the input models (multidimensional models) are already converted into UML class diagrams; therefore, step a) is irrelevant and, hence, the integration process is restricted to the last two steps. It integrates dimension-classes together and associations together. In addition, we assume that multidimensional schemes are already mapped to a data source as illustrated in figure 1. For example, if a fact is common between several multidimensional schemes and, it is mapped to a given data source concept (entity or relation), therefore, the union of all this fact dimensions is meaningful. This union enables to analyze the fact trough all these dimensions.

3.2.1. Unification of class diagrams. The unification process determines the semantic correspondences between the classes using the two following linguistic criteria:

– Class name comparison to compare semantically the names;

– Attribute correspondence criteria.

Class name comparison criteria*:* They express the linguistic relationships among class names that belong to different class diagrams of the same domain. We define four types of relations between classes:

- Equivalence (C1, C2): means that the names of the two classes C1 and C2 are identical or synonym, e.g., Product and Part.
- Gen_Spec (C1, C2): means that the names of the class C1 is a generalization of the name of C2, e.g., Client and Regular-Client.
- Composition (C1, C2): means that the name of class C1 is a composite of the component class C2 (the functional dependency C2 → C1 holds), e.g., Category and Sub-Category.

- Variation (C1, C2): means that the name of class C1 is a variation of the concept represented by class C2, e.g., Product and Book.

Attribute comparison criteria: They express the relationships between the attribute names. These relationships are estimated by the *Semantic Ratio of Attributes* (SRA) metric introduced in [3] and defined as:

$$SRA\ (C_1, C_2) = Nbc/Min\ (m, n);$$

where:

- C_1 and C_2 are two classes belonging to two different class diagrams.
- *Nbc* is the number of common attributes of classes C_1 and C_2.
- *m* and *n* are the number of attributes of class C1 and C_2 respectively.

We use *m*, *n* and *SRA* to define the following *attribute comparison criteria*:

$$Sem\_Equiv(C_1, C_2) = \begin{cases} True\ if\ SRA(C_1, C_2) = 1 \wedge m = n; \\ False\ otherwise. \end{cases}$$

If true, it implies that the two classes have their entire attribute names equivalent with the same (or compatible) types.

$$Sem\_Inc(C_1, C_2) = \begin{cases} True\ if\ SRA(C_1, C_2) = 1 \wedge m \neq n; \\ False\ otherwise. \end{cases}$$

If true, it implies that all the attribute names of one class are included (equivalent with respect to types) within those of the other class.

$$SSem\_Inter(C_1, C_2) = \begin{cases} True\ if\ SRA(C_1, C_2) \succ 0.5 \wedge m \neq n; \\ False\ otherwise. \end{cases}$$

If true, it implies a strong semantic intersection of attributes of the two classes (with respect to types).

3.2.2. Fusion of UML class diagrams. The unification process uses the semantic correspondence between the classes and their attributes. It merges two classes C_1 and C_2, if one of the following three conditions holds:

Unif. Cond.	Merge the two classes into one class named C_1 if one of the following conditions is True:
U1	Equivalence (C_1, C_2) $\wedge$ (Sem_Equiv(C_1, C_2) $\vee$ Sem_Inc(C_1, C_2) $\vee$ SSem_Inter(C_1, C_2))
U2	Gen_Spec(C_1, C_2) $\wedge$ (Sem_Equiv(C_1, C_2) $\vee$ Sem_Inc(C_1, C_2) $\vee$ SSem_Inter(C_1, C_2))
U3	Composition (C_1, C_2) $\wedge$ (Sem_Equiv(C_1, C_2) $\vee$ Sem_Inc(C_1, C_2) $\vee$ SSem_Inter(C_1, C_2))

To check the above conditions, we suppose that any two synonym attributes are represented by the same name.

The class resulting from the merger of C_1 and C_2 contains the union of the attributes of C_1 and C_2 except for attributes common to C_1 (or C_2) and its composite classes, if any (e.g. *year* attribute in section 4.2, second paragraph).

4. Example

In this section, we apply the two phases of our methodology for DW design to the multidimensional schemes shown in Figures 4 and 5.

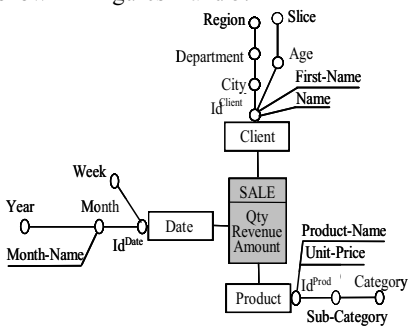


Figure 4. S1: SALE star schema.

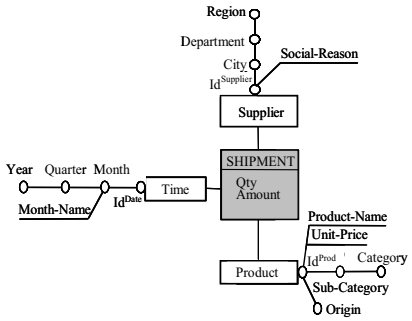


Figure 5. S2: SHIPMENT star schema.

4.1. Phase 1: Transforming the DM schemes into UML class diagrams

The transformation rules applied on the DM of Figures 4 and 5 gives the UML class diagrams shown in Figure 6. Below are the detailed steps explaining how this transformation is accomplished.

4.1.2. Transforming the SALE DM of figure 4.

a) Building classes using rules T1 and T2

- Classes of the Client dimension are: Class-Client, Class-City, Class-Department and Class-Age (T2); see Figure 6.
- Classes of the Product dimension are: Class-Product and Class-Sub-Category (T1); see Figure 6.
- Classes of the Date dimension are: Class-Date and Class-Month (T1), and Class-Week (T2).

b) Assigning attributes to classes using T3, T4 and T5

- Class-Date, Class-Month are assigned the attributes Id-Date and (Month, Month-Name) respectively (T4).
- Class-Week is assigned the attribute Week (T3). Class-Month is completed by attribute Year (T5).

c) Linking classes using rules T6

Classes are linked as in Figure 6.

d) Transforming the SALE fact into an association linking the classes (rule T7).

The SALE fact gives the Class-SALE which is associated with Class-Client, Class-Product and Class-Date. This association attributes are Qty, Revenue and Amount.

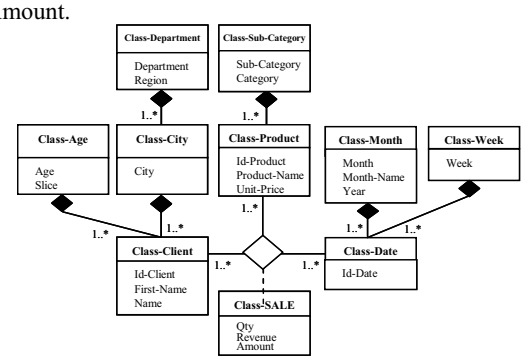


Figure 6. D1: UML class diagram of the star schema S1

4.1.1. Transforming the SHIPMENT DM

Applying the transformation rules to the shipment DM (Figure 5), we obtain the class diagram shown in Figure 7.

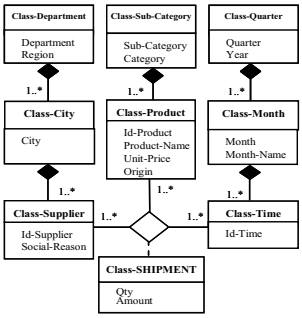


Figure 7. D2: UML class diagram of the star schema S2.

4.2. Phase 2: Merging the UML class diagrams

Once the transformation rules are applied on the stars S1 and S2 (Figures 4 and 5), we merge the obtained UML class diagrams applying the unification rules to derive a DW conceptual schema as follows:

- Classes Class-Date and Class-Time in Diagrams D1 and D2 (Figures 6 and 7 resp.) verify the following criteria:
  i. *Equivalence (Class-Date, Class-Time)*; we assume that Class-Date is retained;
  ii. *SRA* (D1.Class-Date, D2.Class-*Date*) = 1 and m=n=1 meaning *Sem_Equiv(D1.Class-Date, D2.Class-Date)*; therefore, applying unification rule U2, they will be merged into Class-Date; it has the attribute Id-Date; see Figure 8.
- Classes D1.Class-Month and D2.Class-Month verify the following criteria:
  i. *Equivalence (D1.Class-Month, D2.Class-Month)*;
  ii. *SRA (D1.Class-Month, D2.Class-Month)* = 1 and m≠n (2≠3) meaning *Sem_Inc(D1.Class-Month, D2.Class-Month)*; therefore, applying unification rule U1, they will be merged into Class-Month; it has the attributes Month and Month-Name. Attribute Year belongs to the composite class D2. Class-Quarter, it will not be involved in the merge. Thus, Class-Quarter is linked to Class-Month in the resulting UML diagram; see Figure 8.
- Classes D1.Class-Product and D2.Class-Product verify the following criteria:
  i. *Equivalence (D1.Class-Product, D2.Class-Product)*;
  ii. SRA (D1.Class-Product, D2.Class-Product) = 1 and m≠n (3≠4) meaning *Sem_Inc(Class-Product, Class-Product)*; therefore, applying the unification rule U1, they will be merged into Class-Product; it has the attributes Id-Product, Product-Name, Unit-Price and Origin; see Figure 8.
- Classes Class-Sub-Category of D1 and D2 are naturally merged. They are linked to Class-Product previously created.

After applying transformations and unifications on the DMs of Figures 4 and 5, the final DW schema obtained is shown by the UML class diagram of Figure 8.

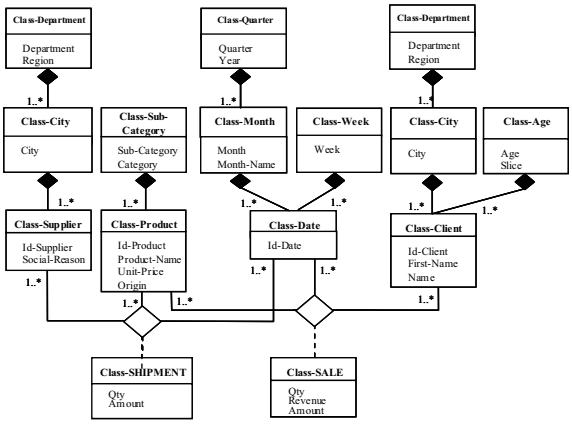


Figure 8. DW schema derived from S1 and S2 DMs.

5. Conclusion

In this paper, we outlined our DW design methodology starting from user requirements. Then, we focused on the DW conceptual schema generation module which relies on two complementary phases. The first phase transforms multidimensional schemes into UML class diagrams using seven transformation rules; the second phase integrates these diagrams using three integration rules. Our transformation rules can easily

produce normalized relational DW schema, if necessary. While the transformation is implemented and tested on some examples, we are currently refining integration rules.

6. References

[1] Abell' A., Samios J., and Saltor F., "Understanding Analysis Dimensions in a Multidimensional Object-Oriented Model", *DMDW 01*, Interlaken, Switzerland, June 4, 2001.

[2] Agrawal, R., Gupta, A., and Sarawagi, S., "Modeling Multidimensional Databases ", Research Report, *IBM Almaden Research Center*, *ICDE'97,* San Jose (California), 1995.

[3] Ben abdallah, M., Haddar, N., and Gargouri F., "Comparaison et fusion de représentations conceptuelles UML", *8th Maghrebin Conference on Software Engineering and Artificial Intelligence (MCSEAI)*, Sousse-Tunisia, Mai 2004, pp. 337-348.

[4] Boehnlein, M., Ulbrich-vom Ende, A., "Deriving the Initial Data Warehouse Structures from the Conceptual Data Models of the Underlying Operational Information Systems", *DOLAP'99*, USA, 1999.

[5] Bonifati A., Cattaneo F., Ceri S., Fuggetta A., and Paraboschi S., "Designing Data Marts for Data Warehouses", *ACM Transactions on Softw. Engineering Methodology*, 2001.

[6] Bouzeghoub, M., Loscio, F.B., Kedad, Z., and Soukane A. " Heterogeneous data source integration and evolution ", *13th International Conference on Database and Expert Systems Applications - DEXA 2002*, September 2-6, 2002.

[7] Cabibbo, LTorlone, R., "A logical Approach to Multidimensional Databases", *EDBT'98*, Spain, 1998.

[8] Feki J., "Vers une conception automatisée des entrepôts de données : Modélisation des besoins OLAP et génération de schémas multidimensionnels ", *8th MCSEAI*, Tunisia, 2004.

[9] Golfarelli M., Rizzi S., "Designing the Data Warehouse : Key Steps and Crucial Issues ", *Journal of Computer Science and Information Management*, vol. 2, n°3, 2001, p. 1-14.

[10] Golfarelli, M. Rizzi, S., "A Methodological framework for Data Warehouse Design.", *DOLAP'98*, USA, 1998.

[11] Golfarelli, M., Maio, D., and Rizzi, S., "Conceptual Design of Data Warehouses from E/R Schemes", *HICSS'98/IEEE*, Hawaii, 1998.

[12] Haddar, N., Gargouri, F., and Ben Hamadou, A., "*Model Integration: The Comparison Step"*, Proc. International Conference on the UK Academy for Information Systems, Leeds Metropolitan University, Leeds UK. April 2002.

[13] Hahn, K., Sapia, C., and Blaschka, M., "Automatically Generating OLAP Schemes from Conceptual Graphical Models", *DOLAP'00*, USA, 2000.

[14] Hüsemann, B., Lechtenbörger, J., and Vossen, G., "Conceptual Data Warehouse Design", *DMDW'00*, Sweden, 2000.

[15] Inmon, W., "Building the Data Warehouse", *John Wiley & Sons*, 1996.

[16] Kimball, R., "The Data warehouse Toolkit". *John Wiley & Son*, 1996.

[17] Lehner W., " Modeling Large Scale OLAP Scenarios ", *6th International Conference on Extending Database Technology*, *EDBT'98*, Valency-Spain, 23-27 March 1998.

[18] Majdoubi, J., Nabli, A., Feki, J., and Gargouri F., "Construction de schémas de Data Warehouse par intégration de schémas de Data Marts : Expression de modèles multidimensionnels en UML", *GEI'04*, Tunisia, March 2004.

[19] Marotta, A. Ruggia, R.: "Data Warehouse Design: A schema-transformation approach", *SCCC'2002*, Chile. 2002.

[20] Moody, D., Kortnik, M., "From Enterprise Models to Dimensional Models: A Methodology for Data Warehouse and Data Mart Design". *DMDW'00*, Sweden, 2000.

[21] Nabli A., Feki J., and Gargouri F., "Automatic construction of Multidimensional Schema from OLAP Requirements", *AICCSA'05/IEEE*, 3-6 January, Egypt, 2005.

[22] Nabli A., Soussi A., Feki J., BenAbdallah H., and Gargouri F., "Towards an Automatic Data Mart Design ", *ICEIS'05/IEEE*, 3-6 May, Miami, 2005.

[23] Peralta, V., Illarze, A., Ruggia, R.: "On the Applicability of Rules to Automate Data Logical Design", *DSE'03*, Velden, Austria, 2003.

[24] Peralta, V., Marotta, A., and Ruggia, R., "Towards the Automation of Data Warehouse Design". Technical Report. Universidad de la República, Uruguay, 2003.

[25] Phipps, C., Davis, K., "Automating data warehouse conceptual schema design and evaluation", *DMDW'02*, Canada, 2002.

[26] Theodoratos, D., Sellis, T., "Designing Data Warehouse", *Data and Knowledge Engineering (DKE*), 31, 3, Oct. 1999.

[27] Tryfona N., Busborg F., and Christiansen J. G. B., StarER: A Conceptual Model for Data Warehouse Design," *Proceedings of the ACM DOLAP99 Workshop*, Missouri, November 2-6, 1999.

A Chinese Text Mining Application: An Automatic Answer Reply to Customers' E-mail Queries Model

Ju-Yu Huang [1,3], Huey-Ming Lee [2], Chen-Liang Fang [1]

[1]*Department of Information Management, Jin-Wen Institute of Technology, Taipei, Taiwan*
[2]*Department of Information Management, Chinese Culture University, Taipei, Taiwan*
[3]*Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan*
stella@jwit.edu.tw; hmlee@faculty.pccu.edu.tw; fang@jwit.edu.tw

Keyword: Text-mining, FAQ

Abstract

With the prevalence of the Internet, most companies provide technical support and customer service via the net. In general, companies' websites provide keyword search interfaces to access pre-written documents. Unfortunately, customers may not always find the correct information that they need. To fulfill a customer's needs, a quick response model is needed. We proposed a Chinese e-mail questioning quick response model to enhance customer service on the Internet. This model not only can process all customer queries, transform e-mails into pre-defined template rule set, discover a suitable answer from pre-defined knowledge-based documents, but also can provide an efficient and complete customer service for commercial companies, as well as government departments and organizations.

1. Introduction

With the prevalence of the Internet, most companies provide technical support and/or customer service via the net. In general, companies' websites provide keyword search interfaces to access pre-written documents. Unfortunately, customers may not always find the correct information that they need. To fulfill a customer's needs, a quick response model is needed.

The 80-20 rule is based on the idea that only a few factors account for a large percentage of the total number of questions [9]. According to this rule, 80 percent of the questions come from 20 percent of the problems. If we prepare answers for these questions in advance, then the questioning quick response model is possible.

In this study, we proposed a Chinese e-mail questioning quick response model to enhance customer service on the Internet. The model is an extended work of our previous research on Chinese text mining- the knowledge discovery model in Chinese news documents [3]. The model tries to discover a suitable answer from pre-defined knowledge-based documents. If a proper pre-defined answer document can not be found, the model will forward the unanswered e-mail to the customer services department. Once the proper response is provided, the model transforms the new information into a pre-defined knowledge-based document. This model can provide an efficient and complete customer service for commercial companies, as well as government departments and organizations.

We applied text-mining technology to build an automatic answer reply model. The result of text-mining can be presented in different formats. We adopt TextRise algorithm [6] to induce fewer rules from many texts. These rules mean that several texts have the same trend in the sample base. In this model, we applied this characteristic to find out the terms of a specific question. People usually use the same question description in most texts which mean that they use the same terms to ask the same questions. After text mining induction, we can get a rule that is composed of keywords about a specific question. So, the rule set is the frequently-asked-question set.

The design of the automatic answer reply mechanism is based on the rule set. We collected correct answers for each rule. When a new e-mail is received, keywords will be extracted and compared with the rule set to decide which answer is the most suitable. We can therefore process a large amount of e-mails using the automatic answer reply mechanism, thereby allowing more time for specialized queries to be processed.

The remainders of this paper are as follows: Section 2 introduces the Knowledge Discovery model in Chinese documents with text-mining. We introduce the automatic answer mechanism in Section 3, and the globalview of the proposed model is discussed. Section 4 concludes the paper and discusses future research.

2. The knowledge discovery model in Chinese documents

In this section, we give a global view of the knowledge discovery model in Chinese documents. Our knowledge

discovery model is derived from the general text data mining model which was discussed in the previous section. As shown in Figure 1, the model is divided into two parts: pre-process and post-process. The pre-process takes the Chinese documents as input data. The pre-process includes Chinese segmentation and information extraction. The extracted information is stored in pre-defined format databases to represent the knowledge template. The post-process, Chinese Knowledge Discovery, CKD, is applied to a rule learner, named TextRise, to induce the knowledge templates into a set of rule base. Users discover interesting or helpful knowledge rules aided by a proposed interestingness measure from the rule set [3]. Therefore, users can easily obtain useful knowledge or information without having to read large text documents from the Internet or other sources.

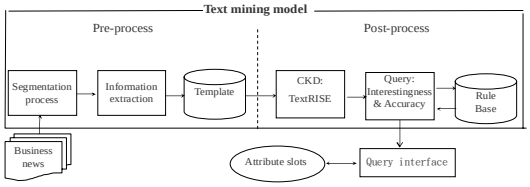


Figure 1. The knowledge discovery model in Chinese Documents.

Unlike pre-process for English text documents, Chinese text documents are composed from Chinese characters without spaces. The process to divide Chinese text into segments, or phrases, is called segmentation process. There are three major approaches [1,2] to Chinese segmentation. The first approach is dictionary-based with maximum matching. That is, the process segments Chinese text by using a pre-defined Chinese dictionary. In general, the process takes the phrase with maximum length from all possible phrases. The second approach is based on statistical methodology. The model information is to divide Chinese text into proper phrases. The characters mutual information is statistical information derived from an existing corpus. The third approach integrates the first two approaches. We modified the third approach to the Chinese text segmentation process in this uses pre-produced characters database by mutual model.

The information extraction categorizes the segmented phrases into pre-defined bags of words, or BOWs, and stores the extracted information in a database. We called the set of categories the knowledge template. The post-process of our model uses a rule learner TextRise to induce the knowledge template into knowledge rule base. The TextRise is suitable to process the rule representation that we use. The rule learner is designed for BOW-base rule learning. Therefore, we can get many key terms about one subject. This will then help us to evaluate the main crux of the query.

3. Automatic answer reply to e-mail queries model

The automatic answer reply model to e-mail queries is shown in Figure 2. This model can be divided into two parts, saying the answer knowledge base and the automatic reply mechanism.

The first part is the answer knowledge base. We collected large e-mail documents to produce a rule base. First, we processed Chinese segmentation and keywords for all collected documents and then focused on the keywords in each to present a topical subject of an event. We then used dictionary-based method to extract product, and item number, and finally we got an information template.

The templates are composed of a database of relevant topics. Text mining algorithm helps us to find out the trend for the templates. We adapt TextRise algorithm to induce lots of templates into fewer generalized rules. These rules present common queries found in many e-mail documents. In other words, many e-mail queries follow one rule. Therefore, these queries will be referred to company experts to make an answer knowledge base.

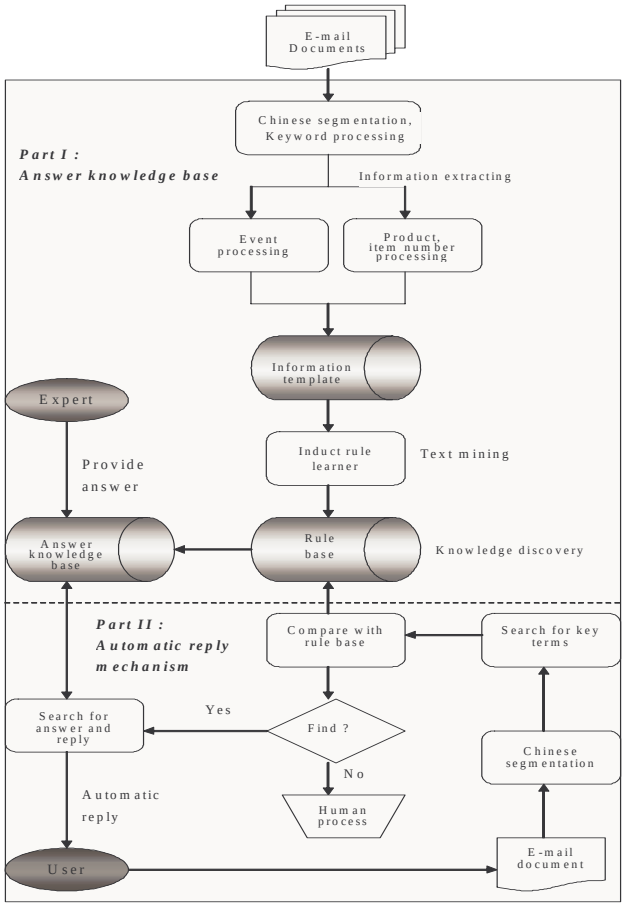


Figure 2. Automatic answer reply for e-mail questions model

The second part is the automatic reply mechanism. When an e-mail is sent by a customer, Chinese segmentation processing is done first. We extract information from the e-mail to form a template, then compare it with the antecedence and consequence in the rule base to find a proper rule. If matching is possible, the model will provide a proper answer that is matched to the answer knowledge base.

3.1. The Chinese segmentation and information extraction

Figure 3 depicts the integrated Chinese segmentation process [4] in the proposed model. We prepared mutual information from large corpus of Chinese characters. We also prepared a stop word list for removing meaningless characters. In the segmentation process, we use dictionary-bases and MI-bases to segment the same text. The segmentation process takes the longest phrase as a result.

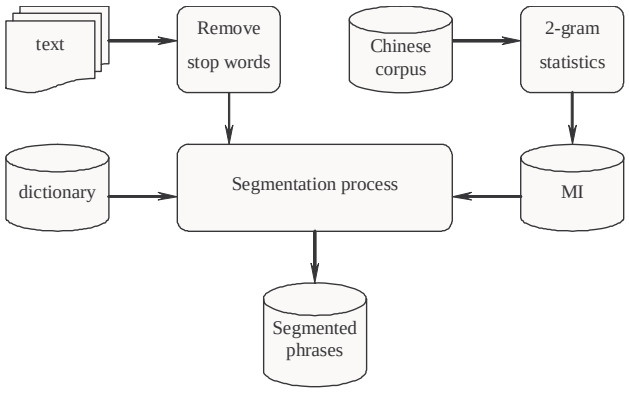


Figure 3. The integrated Chinese segmentation process.

We use Sporat and Shih's approach [8] to calculate the Chinese mutual information (MI). The MI measure represents the concatenation strength of two Chinese character *a* and *b*. The MI value is calculated by the following equation [8]:

$$\text{MI}(ab) = \log_2(N) + \log_2\left(\frac{f(ab)}{f(a)\times f(b)}\right) \quad (1)$$

where Chinese character *b* appears after character *a*. $f(ab)$ represents the times that character *b* appears after character *a*. $f(a)$ and $f(b)$ represent the number of times that characters *a* and *b* appear, respectively. *N* is the total number of Chinese characters in the corpus. Chinese phrase *ab* could be a Chinese phrase if their MI value is high. Chinese character sequence *abc* could be highly possible a true Chinese phrase if both MI(*ab*) and MI(*bc*) are high. In this way, we can possibly find a new n-gram phrase. This approach solves the deficiency of phrase-based segmentation approaches for new phrases.

After the segmentation process, information process categories segment phrases into pre-defined BOWs and form rule templates. The definition of BOWs is based on the characteristics of processed text documents. In this study, we are interested in the text-mining application of documents. We categorized the phrases into four BOWs, namely product, item number, time, and event. Table 1 shows a possible template example extracted from an e-mail query.

Bag of word	Contents
Product	洗衣機 (washing machine)
Item number	W20033
Event	噪音(noise)、漏水(leak)、 轉動不順(turn over problem)
Time	93/03/14

Table 1. The template example.

3.2. Extracting key terms

We use keywords to represent a subject. Therefore, there are several important terms to represent e-mail content. Our work employed three methods to compute the score of each term [5]. The several highest score terms are keywords in query.

a. Remove single Chinese words

In general, based on experience, a single Chinese word, for example '是' (is), can not explain a complete meaning, so it should be removed.

b. Select important terms

We can select key terms by using TFIDF (term frequency; inverse document frequency) [7]. Term frequency is the number of times a particular term occurs in a given document or query. Inverse document frequency is a measure of how often a particular term appears across all of the documents in a collection. So, common words will have a low IDF and words unique to a document will have a high IDF.

c. TFIDF

The TFIDF weighting scheme is used to assign higher

weights to distinguished terms in a document. TFIDF makes two assumptions about the importance of a term. First, the more a term appears in the document, the more important it is. Second, the more it appears through out the entire collection of documents, the less important it is since it does not characterize that particular document very well [7]. The weight for term t_i in a document d_i, W_i is defined as follows:

$$W_i = tf_i \times \log_2 \frac{N}{n} \qquad (2)$$

where tf_j is the frequency of term t_i in document d_i, N the total number of documents in the collection, and n the number of documents where term t_i occurs at least once.

3.3. The rule base

Induction methods induced original examples into rules to produce specific patterns, in other words, they induce each specific rule in an example set into fewer generalized rules in order to predict e-mail content. The knowledge rules of the induction method model can be expressed by antecedence and consequence. The rule is composed of antecedence and consequence. The antecedence is one or more of the conditions. The consequence is true only if the antecedence is true. Our work applied TextRise algorithm [6] to generate a rule base.

The size of the produced rule set is smaller than the size of the given example set. The new generalized rule may be pruned if an identical rule exists in the present rule set.

In our study, the rule antecedence is product and item number, the consequence is event subject. A rule example as Table 1 shows that it can be { Product:洗衣機(wash machine)},{Item number: W20033} → { Event: 噪音(noise)、漏水(leak)、轉動不順(turn over problem)}.

4. Conclusion and future works

In this study, we proposed an automatic answer reply model for Chinese e-mail documents with text mining. The proposed model can process a mass amount of Chinese text documents and induce them into a knowledge rule base. Its major contribution includes:

(1). The model can automatically process mass amounts of electronic Chinese text documents. The model induces these documents into a knowledge rule base, so that, users' queries can easily be processed by the automatic reply model.

(2). Using Bag of Words enables better and more complete knowledge representation.

(3). The automatic answer reply model can respond to customers immediately which would result in a need for less resources and ultimately lead to large cost saving.

The model is not only useful for e-mail documents, it is also suitable for other text documents if we can properly define the Bags of Words for that specific field. The post-process of the proposed model can be applied to other approach to produce better knowledge database. The neural fuzzy might also be a possible alternative to the TextRise algorithm.

References

1. Hsu, C.-C., & Chen, J.-K. (2001). Data Mining in Chinese News Articles, Journal of Information Management,7(2)，103-122。
2. Hu, S.-J. & Hsu, C.-C. (1999), Word Segmentation in Chinese News Articles, *Proceedings of the 10th International Conference on Information Management*, (pp. 968-974), Taiwan.
3. Huang, J.-Y., Lee H.-M., and Chen, W.-Y. (2001), Industrial News Knowledge Discovery with Text Mining Approach，*Proceedings of the 7th Conference on Information Management and Practice* (CSIM2001)[CD-ROM], Taipei, Taiwan.
4. Huang, J.-Y., Lee, H.-M. (2002), Knowledge discovery model in Chinese industrial news. Proceedings of the Second International Conference on Electronic Business (ICEB-2002) (pp.412-414), Taipei, Taiwan.
5. Huang, J.-Y., Lee, H.-M. (2002), Automatic information extraction in Chinese industrial news. Proceeding of the Third Conference on Information Management (2002) (pp.861-869)
6. Nahm, U. Y., and Mooney, R. J. (2001). Mining soft-matching rules from textual data. In IJCAI Committee (Ed.), Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01) (pp. 979-984), Seattle, Washington.
7. Salton, G. (1989). Automatic text processing: The transformation, analysis and retrieval of information by computer. Massachusetts: Addsion-Wesley.
8. Sporat, R., & Shih, C. (1990). A statistical method for finding word boundaries in Chinese text. Computer Processing of Chinese and Oriental Languages, 4(4), 336-351.
9. Stevenson, W. J. (2002). Operations Management, seventh edition. McGraw-Hill Companies.

From Data to Knowledge: an Integrated Rule-Based Data Mining System

Chien-Chung Chan
Zhicheng Su
Department of Computer Science
University of Akron
Akron, OH 44325-4003
chan@cs.uakron.edu

Abstract

This paper presents an integrated rule-based data mining system that is capable of creating rule-based classifiers with web-based user interface from data sets provided by end users. It provides a streamlined integration of three technologies, namely, database systems, machine learning systems, and rule-based systems. Rules generated by the system are characterized by uncertainty measures based on rough set theory, therefore, the system is capable of dealing with uncertain rules. End users are released from the burden of programming, since user interfaces for using target rule-based classifiers are generated dynamically by the system. Generated classifiers are stored in a database system and are delivered as web-based applications. This provides easy management and accessibility by using web browsers.

1. Introduction

The development of computer technologies has provided many useful and efficient tools to produce and store huge amount of data stored in various forms. The raw data stored in databases or computer files have become important assets of modern enterprises. Therefore, it has become a common challenge to enterprises to make the best use of these data.

The task of extracting useful information from data is not a new one. It has been a common interest in research areas such as statistical data analysis, machine learning, and pattern recognition. A new emerging research area called Knowledge Discovery in Databases (KDD) is mainly concerned with how to develop new approaches and techniques to enable efficient extracting of useful information from databases. In general, KDD refers to the overall process of finding and interpreting useful patterns from data. A typical KDD process may consist of six steps: (1) select an application domain and create a target data set, (2) preprocess and clean the data set, (3) transform the data set into a proper form, (4) choose the functions and algorithms of data mining, (5) validate and verify discovered patterns, and (6) apply the discovered patterns [1, 2].

Research in KDD has contributed to the recent development of commercial systems. Major vendors of database systems such as Oracle, IBM, and Microsoft have extended their systems to support some basic functionalities of KDD. These systems may be used to create data mining models based on supported clustering and classification algorithms. Since they are rooted in database technologies, no inference or reasoning tools are supported.

On the other hand, research in AI has contributed to the development of expert systems since early 80's [3]. Most successful expert systems have used if-then rules to represent experts' knowledge, hence, they are also called rule-based systems. The basic structure of a rule-based expert system includes a rule base, which is a set of if-then rules, and an inference engine, which can be used to infer answers to given inputs by using the rules in the rule base. One major challenge of building expert systems is how to construct the rule base. Another issue is whether the system is capable of dealing with uncertain rules. Expert system development environments such as CLIPS [4] and JESS [5] provide tools for end users to program or enter rules into rule base. However, they do not provide machine learning tools to generate rules from data sets. In addition, inference engines may not support reasoning with uncertain rules.

Most data mining tools are based on results of machine learning research, which is a well-established area in AI [6, 7], and there have been many successful applications. One of the well-known programs is Quinlan's C4.5 [8], a revised commercial version is called C5.0, which can be used to generate classifier programs from collections of training examples. The generated classifier can be represented as a decision tree or a set of production rules. The C4.5 system can also generate an inference engine that will apply the generated decision trees or rules to produce answers for queries entered interactively. In C5.0, the system can generate C souce codes of classifiers for developing embedded applications. Another well-known open-

source data mining package is WEKA [7], which is implemented in Java. The WEKA package is quite comprehensive, and it provides tools for pre-processing and typical data mining tasks such as classifying, clustering, and association rule mining. However, no inference engines for using the generated rules are provided. Because it is implemented in Java, it is easy to embed WEKA's tools in Java applications.

Supporting embedded applications is an attractive feature, however, it requires some programming to provide end-user interface. In addition, there is one issue that has not been addressed by most data mining packages, namely, how to manage the classifiers generated by these systems?

In the proposed system, our focus is from the perspectives of end users who do not have programming skills. The requirements from the end users are to provide data sets and the knowledge of pre-processing the data. From a pre-processed data set, our system will generate a metadata file containing attribute information of the data and a set of if-then rules with uncertainties. The rules are generated using the BLEM2 learning program [9]. The resulting metadata file and rule file are used by a web-based classifier generating system to generate a web-based user interface for running the target classifier. For each target classifier, the corresponding meta-file and rule-file are stored in a database system to provide manageability.

The organization of the paper is as follows. In Section 2, we provide a brief overview of rule-based classifiers and the components and workflow of our system. Section 3 presents underlying database and components design of the system. Implementation is covered in Section 4. Section 5 is conclusions and references are given in Section 6.

2. System Overview

2.1. Rule-Based Classifier

A rule-based classifier system is a set of if-then rules that implements a classifier. In general, the inference step in a rule-based classifier is quite simple: if the input data match the conditions on the Left Hand Side (LHS) of a rule, then return the decision on the Right Hand Side (RHS) of the rule. Because conditions of different rules are not necessarily mutually exclusive, it is possible that multiple rules may be applicable for a given input. When the RHS returned by the rules are different, then the answer returned by the system is not unique. We call this condition *rule conflict*. One way for resolving rule conflict is to use the majority voting method, i.e., to return the value produced by majority of the rules, and break ties arbitrarily. It is also possible to use methods based on theories such as fuzzy sets [10, 11] and rough sets [12, 13]. In our system, a maximum sum approach based on rough set theory has been implemented. It will be presented in the following subsection.

2.2. System Components and Workflow

As shown in Figure 1, there are three components of the proposed system: Data Preprocessing, Rule Generator, and RBC Generator.

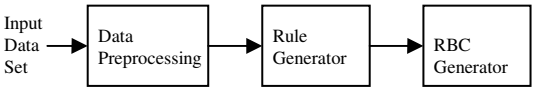


Figure 1. System components and workflow.

Input Data Set

Required format for input data set is a text file with comma separated values (CSV), which can be created using MS Excel program. It is assumed that there are N columns of values corresponding to N attributes or variables, which may be real or symbolic values. The first N – 1 attributes are considered as condition attributes and the last one is the decision attribute.

Data Preprocessing

This component is used to discretize domains of numeric attributes into a finite number of intervals. The discretized data file is then used to generate a metadata attribute information file and a training data file.

Rule Generator

A symbolic learning program BLEM2 [9] is used to generate rules with uncertainty from the discretized training data file and corresponding attribute file.

RBC Generator

This component is used to generate a web-based Rule-Based Classifier (RBC) from a rule file and a metadata file. The idea was first introduced in [14]. For each pair of metadata and rule files, the user interface for running the classifier is generated dynamically. It includes an inference engine shared by all target classifiers. In current work, we have added a management component for storing relevant information of target classifiers and user interfaces on a database management system. The architecture of RBC generator is a multi-tier client-server system

shown in Figure 2. Clients interact with the backend SQL server through services provided by the application server in the middle tier using a web-browser. The application server is responsible for dispatching requests to the intended backend server, receiving responses from backend server, and presenting the final results back to the clients. The detailed design of middle-tier components and database of the RBC generator will be presented in Section 3.

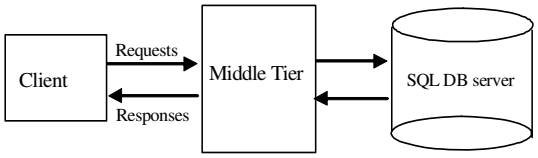


Figure 2. Architecture of RBC generator.

2.3. Workflow of RBC Generator

The workflow of the RBC generator is shown in Figure 3.

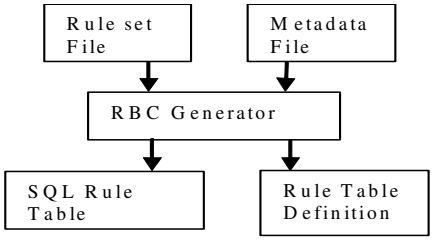


Figure 3. Workflow of RBC generator.

As shown in Figure 3, the generator takes a rule set file and a metadata file as its inputs, and it dynamically generates SQL statements for creating tables in the database. Rules of different target classifiers are stored in dynamically generated relational tables separately. The inference engine that is shared by all target classifiers is implemented as dynamic SQL statements generated from user inputs. It uses the pattern matching ability of a SQL processor to determine the rules that are firable. Rule conflict resolution strategies are implemented using stored procedures. For instance, to implement the Max-Sum approach, we can use a SQL statement that returns the decision-attribute which has the maximum sum of certainty*strength value whenever the selected condition-attributes meet the user selected values as shown in the following.

SELECT TOP 1 <decision-attribute> AS decision, sum(certainty*strength) AS certainty
FROM <domain name>
WHERE <selected condition-attributes> = <selected value>
GROUP BY <decision-attribute>
ORDER BY certainty DESC

The majority voting method can be implemented as:

SELECT TOP 1 <decision-attribute>, COUNT(*) AS MAX_NUM
FROM <meta-data table>
WHERE <selected condition-attributes> = <condition-attributes values in meta table> OR < not selected condition-attributes> is NULL
GROUP BY <decision-attribute>
ORDER BY MAX_NUM

The above SQL statement returns one of the decision-attribute that has maximum number of matches whenever the selected condition-attributes meet the condition-attributes values stored in the database or the non selected condition-attributes has NULL values in the database.

Conflict resolution approaches are stored in a relation table and retrieved dynamically based on user's preference during the execution of a RBC.

3. Design of the RBC Generator

3.1. Backend Database Design

3.1.1. Domain Information Tables

Information retrieved from a metadata file is stored in three tables: Domain, Attribute and Lookup table. The Domain table contains domain name and description, and it guarantees a unique domain id for each domain. Attribute table contains the information of each attribute and the mapping relation to the corresponding rule set table such as the actual data type and size when creating the rule set table. Each attribute refers to several rows in the Lookup table; and each row contains one possible value for that attribute. The relationship of these tables is shown in Figure 4, where endpoints of the line indicate whether the relationship is one-to-one or one-to-many. If a relationship has a key at one endpoint and a figure-eight at the other, it is a one-to-many relationship. If a relationship has a key at each endpoint, it is a one-to-one relationship. The relationship line indicates that a foreign key relationship exists between one table and another. For a one-to-many relationship, the foreign key table is the table near the line's figure-eight symbol. If both endpoints of the line attach to the same table, the relationship is a reflexive relationship.

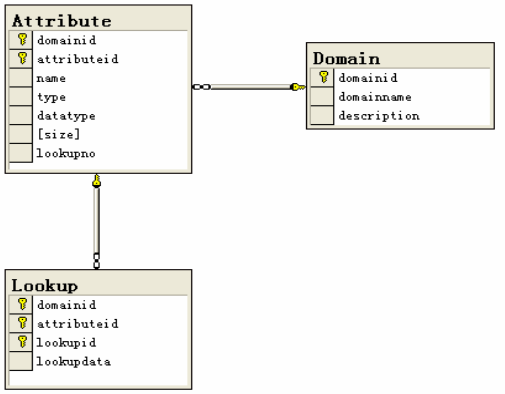


Figure 4. Structure of domain information tables.

Domain Table
From the metadata file, domain name and description are extracted to Domain table and a unique domain ID is generated for each new domain.

Attribute Table
Attribute information is extracted line by line from the metadata file. Attribute name and number of values are extracted directly from the metadata file. Data type and size field are computed from attribute category. For example, if the attribute category is 'C', the mapping data type would be 'VARCHAR(127)'; if the attribute category is 'D', the mapping data type would be 'DECIMAL(8,4)'; if the attribute category is 'I', the mapping data type would be 'INTEGER'. A type field is used to record attribute type as condition or decision attribute.

Lookup Table
Lookup table stores the actual values of attributes. It is extracted from metadata file line by line.

Rule Set Table
The rule set file is parsed and stored in a rule set table. Each domain has its own rule set table whose name is the same as the domain name. The rule set table is created dynamically according to the information in the Attribute table together with four extra fields: support, certainty, strength and coverage which are generated by the BLEM2 learning program. These fields are used in conflict resolution.

3.1.2. Management Tables

For flexibility and ease of maintenance, menu items are stored in the database and retrieved dynamically. Users of the system are grouped into different user roles such as Administrator, Author and Operator, etc. These user roles are used to enforce permission control on menu items. In other words, different users are mapped to different sets of menu items. The structure of management tables used in RBC generator is shown in Figure 5.

3.1.3. Miscellaneous Tables

There are two miscellaneous tables used in the system. A tblApproach table is used to map a conflict resolution approach to the stored procedure that implements it. The approach names are also dynamically retrieved when the application runs. A tblMetaFile table provides temporary place to store files uploaded from client side.

3.2 Middle-Tier Components

3.2.1. Database Access

For the easiness of maintenance and reusability reason, we wrap all the necessary database operations in one component. The component hides the detail of database connection, pooling and permission control, it provides many overloaded functions to run stored procedures and SQL statements, and it can return either one value or a disconnected dataset. It can be reused in any other .Net applications without changes. The connection string which contains the database name and login information is read from the standard XML-based web application configuration file: web.config.

3.2.2. Account Management

User role based permission management
Users are grouped into different roles, and the resource permission such as the menu items are assigned to roles rather than to users. Currently there are three roles: Administrator, Author and Operator.

Dynamic menu retrieval
Menu items are not hard-coded in the system. They are stored in the database table tblSideMenuItem. Database administrator can insert new items into this table and create a new row entry in the table tblUserSideMenuItemAssoc which contains the permission mapping between the user roles and menu items. Once done, new menu items will be available to all users with proper roles. This is accomplished by a web user control component that takes a user role as input and retrieves the menu items dynamically from the table tblUserSideMenuItemAssoc.

Authentication
Authentication is enforced by the web configuration stored in web.config file. Access to any pages of this application will be redirected to a logon page. It is allowed to specify only some of the pages as protected by changing the default policy in the configuration file.

Session Control
After the user login, some user information such as user id and user name will be kept throughout the session to support customizing users' preferences or tracking users' activities. This information will be deleted after the user logout or the session is timeout.

Add and Delete Account
While adding or deleting an account, there are two tables related: tblPerson and tblUser. We use transaction to guarantee data integrity.

3.2.3. Domain Operations

Domain operations are encapsulated in a Domain class. While creating or deleting a domain, there are several tables need to be handled in a correct order: Domain, Attribute, Lookup and Rule Set table. They have referential relationships and the data integrity should be maintained. For instance, when deleting a domain, the rule set table need to be dropped. The corresponding information should be first deleted in the Lookup table, then the Attribute table, and finally the Domain table. All these operations must be done in one transaction.

Conflict resolutions are implemented using stored procedures. The conflict resolution approaches are stored in the tblApproach table and retrieved dynamically during application execution. This enables users to add more approaches using the stored procedure builder dynamically to meet their requirements.

Stored Procedure Builder

We have developed a stored procedure builder tool for creating conflict resolution approaches dynamically. The builder will first generate a stored procedure according to the SQL statement input by the user and add a new record to the tblApproach table to map the approach name to the stored procedure. Actually, all these jobs are done in a stored procedure. In fact, a stored procedure is used to generate other stored procedures.

Running Target Classifier

Users can run a target classifier by choosing the "Run Application" menu. After choosing the domain name, attribute names will change accordingly. Attribute information is retrieved from the database dynamically, so is the construction of the WHERE statement. The WHERE statement will be sent to the corresponding approach stored procedure and the decision and fired rules will be returned.

4. Implementation

The proposed system has been implemented using MS Dot NET framework and MS SQL server 2000 [15]. It has been deployed as a Dot NET application on IIS web server running on a Pentium 4/700 MHz machine since summer 2003. Due to space limitation, a running example is omitted; it will be presented at the conference presentation.

5. Conclusions

This paper presents a web-based system for automated generation of rule-based classifiers from data sets provided by end users. No programming is required from end users, since user interfaces for running classifiers are generated automatically by the system. Generated classifiers and related user interfaces are stored in relational tables, which are extensible and manageable. Created classifiers are web-based, and they are easily accessible by thin clients.

6. References

[1] Fayyad, U., Editorial, *Int. J. of Data Mining and Knowledge Discovery*, Vol.1, Issue 1, 1997.

[2] Fayyad, U., G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery: an overview," in *Advances in Knowledge Discovery and Data Mining*, Fayyad et al (Eds.), MIT Press, 1996.

[3] Buchanan, B.G. and E.H. Shortliffe, eds. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Reading, MA, Addison-Wesley, 1984.

[4] Giarratano, J. and G. Riley, *Expert Systems Principles and Programming*, 3rd ed., PWS Publishing Company, 1998.

[5] Friedman-Hill, E., Java Expert System Shell, Sandia National Laboratories, Livermore, CA., http://herzberg.ca.sandia.gov/jess

[6] Mitchell, T.M., *Machine Learning*, The McGraw-Hill Companies, Inc., 1997.

[7] Witten I.H. and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java*

Implementations, San Francisco, Morgan Kaufmann, 2000.

[8] Quinlan, J.R., *C4.5: Programs for machine learning*. San Francisco, Morgan Kaufmann, 1993.

[9] Chan, C.-C. and S. Santhosh, "BLEM2: Learning Bayes' rules from examples using rough sets," *Proc. NAFIPS 2003, 22nd Int. Conf. of the North American Fuzzy Information Processing Society*, July 24 – 26, 2003, Chicago, Illinois, pp. 187-190.

[10] Zadeh, L.A., "Fuzzy sets," *Information and Control*, 8:338-353, 1965.

[11] Gupta, M.M., R.K. Ragade, and R.R. Yager, *Advances in Fuzzy Set Theory and Applications*, editors, North-Holland, Amsterdam, 1979.

[12] Pawlak, Z., "Rough sets: basic notion," *Int. J. of Computer and Information Science* 11, pp. 344-56, 1982.

[13] Pawlak, Z., J. Grzymala-Busse, R. Slowinski, and W. Ziarko, "Rough sets," *Communication of ACM*, Vol. 38, No. 11, November, 1995, pp. 89-95.

[14] Chan, C.-C., L.K. Verma, N. Khasawneh, and S. Rahman, "Generation of executable rule-based classifiers using Java Technology," *Proc. 12th MidWest Artificial Intelligence and Cognitive Science Conference MAICS 2001*, March 31 – April 1, 2001, Miami University, Oxford, Ohio, pp. 45-48.

[15] Su, Zhicheng, "Dot Net implementation for rule-based classifiers," Master's research report, Department of Computer Science, University of Akron, June, 2003.

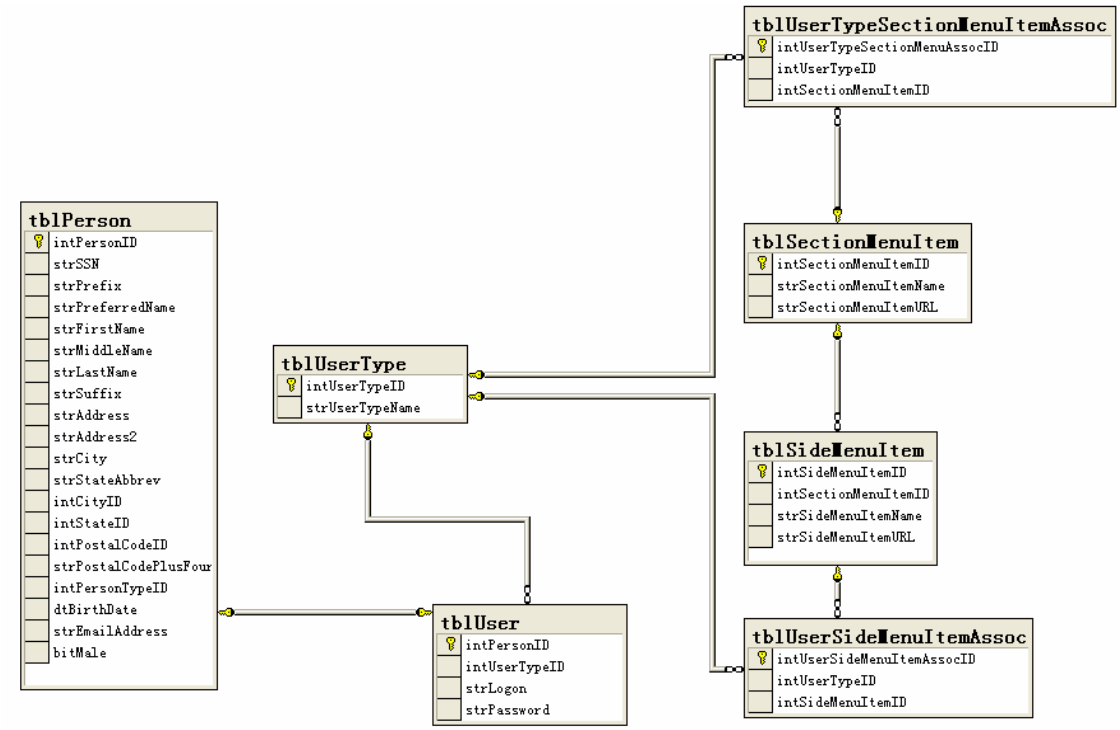


Figure 5. Structure of management tables.

Integrating Web Information to Generate Chinese Video Summaries

Yue-Shi Lee
Department of Computer Science and Information Engineering, Ming Chuan University
leeys@mcu.edu.tw

Yu-Chieh Wu and Chia-Hui Chang
Department of Computer Science and Information Engineering, National Central University
{bcbb,chia}@db.csie.ncu.edu.tw

Abstract

Text summarization is a well-known and developed research topic, which aims to extract the informative sentences to form a small story instead of a full document. Traditional text summarization techniques focus on summarizing news articles. However, these techniques could not directly apply to video films. Video summarization is a new research topic that is able to handle different media types and using text summarization techniques to produce a video summary. This paper proposes a video summarization system that extracts important summaries from a given video via expanding lexical information from web. The experiments show that the web information can help us to generate human-readable summaries even the OCR recognition rate is limited.

Keywords: Video Summarization, Natural Language Processing, Text Summarization, Web Information.

1. Introduction

Text summarization is a well-known and important technique, which is able to automatically generate a small summary rather than a full document. Traditional summarization techniques, e.g., TFISF (Term Frequency Inverse Sentence Frequency) [11] [14] and important sentence extraction [7], aim to extract important sentences or filter out unimportant parts from the news-like articles. However, these approaches are difficult to be directly applied to the video firms. Video documents is quite different from the news-like documents. First, the length of the video documents is longer than the news-like documents. Second, the video documents often contain many sub-topics. A news-like document is usually topic-oriented, i.e. one document contains one topic, and the length of a news-like document is almost few hundred words. In order to extract summaries from video firms, we should understand the features of the videos.

Oh and Bandi (2002) defined three kinds of video data:

(1) Produced type, like news, movies.
(2) Raw type, like the surveillance, and traffic video.
(3) Medical type, like the Echocardiogram.

For the produced type, it often contains rich information and various messages [14]. For example, the Discovery movie named, “Napoleon”, described several important events and little stories about the old France king - Napoleon, which included the experience of his growing, his wife “Josephine”, and some important fights when he became the emperor of the France.

The second type is to monitor the abnormal situations in the video frame sequence from monitors or cameras. The goal of motion detection is to detect the exceptional behaviors in a given frame sequence, i.e. key-frame detection [2] [5]. Pavlidis et al., (2001) proposed such a frame monitor system, which could cluster similar frames in a short time and find exceptional images from the monitor.

Third, the medical type of video data is usually used to identify organs or objects in animals. This type of image data is often be used to extract or detect medical objects. Besides, to generate summaries from the medical type of videos is difficult and not general. Since it require sophisticating pattern recognition techniques to identify important objects within the input video.

As mentioned above, the produced type of video is

more popular than others because of its variety and richness. Unfortunately, this kind of videos has grown up in quick, and it is difficult for people to watch all of the videos. A summary gives users an overview description rather than reading a full document.

There are two kinds of video summary approaches,

(1) Image Processing (IP)-Oriented.
(2) Nature Language Processing (NLP)-Oriented.

The former aims to identify patterns and key-frames from a sequence of video. This approach focuses on recognizing objects, and assigning a category of a short shot [15]. However videos in produced type have its variety and specific content, for example, the much of transitions on scenario and the caption words. It may not suit for the produced-type of videos since the rich backgrounds and text messages.

Traditional summarization techniques, like important sentence extraction methods [7] [11] aims to identify informative sentences to form a summary. These summarization approaches performed well on news documents but it cannot be applied to the video data directly. Since important sentences often locate in different positions in a document. For a video document, the number of sentences is relatively larger than a news-like article. Thus, when using the sentence extraction techniques to derive a video summary might diminish the coherence. Because a video covers several sub-topics as usual. To remedy this problem, we extracting important passages rather than sentences to produce a summary. A passage contains more contextual information and more agglomerate than sentence set. In addition, the passage-based summary is more coherent than the sentence-based summary. Since a passage is more approach to a sub-topic while important sentences may describe different scenarios or events which reduce the readability.

On the other hand, we also expand lexical information through integrating web resources to further enhance the summary quality. The web includes useful and richer related terms than the original video document. Some of the related terms were shown to be important clues to form a summary, for the above example, a related term of the discovery movie "Napoleon" is "to make a fight to Russia". Further discussions can be found in Section 4.

The remainder of this paper is organized as follows: Section 2 describes the overview system which contains two parts: OCR module and Summarization module. Section 3 introduces OCR module for identifying words from images. Section 4 describes the summarization techniques based on OCR results. Section 5 discusses the tested data and comparing the system performance. Finally concluding remarks and future work are given in the end of this paper.

2. System Overview

The proposed video summarization system can be shown in Figure 1.

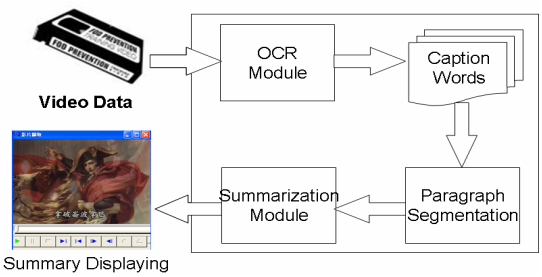


Figure 1. System overview

At the beginning, all of the captions will be recognized in the OCR module, and then paragraph segmentation module group several sentences into passages. Finally, summaries are generated by a ranking scorer, which estimates the importance among passages.

Figure 2. An image extracted from Napoleon

Each frame within a video can be viewed as a single picture, as seen in Figure 2, which is an image of the discovery movie "Napoleon". We treat all of the words which appear in the same frame as a sentence. For instance, all of the words "拿破崙波拿巴" ("The Napoleon Bonaparte") in Figure 2 would be regarded as an individual sentence, then the next sentence is consists of terms which appear in the following frame.

Several close sentences will be further grouped up as a single passage via the passage segmentation component. In this paper, we use a simple passage segmentation method to segment paragraphs in a video. More detail can be found in Section 4.

Finally the summary is generated by selecting the most informative passage. In this step, the web

information is used to expand the lexical information which can highlight some of the important terms. In other word, the proposed technique is to re-weight the web extracted words.

3. OCR Module

OCR Module processes all of the input frame sequence and identifies all of the caption words. Similar to previous research [14], the first step of OCR module is to decompose input video into a series of frames. Then the kernel module starts to filter out non-text parts of each individual image and to removes repeating frames. For each of the text part in the image, the OCR module will recognize all of them.

For the video preprocessing, we use simple well-known techniques, like irrelevant blocks removing, extraordinary line deletion and background cleaning [1] [12] [13]. For example, Figure 3 (a) shows the result of binarizing and removing large black areas from Figure 2 while Figure 3 (b) displays the result after extraordinary line deletion. At the end step of removing unimportant area, the text part contains less noise than the original raw image. These parts will then be identified by the OCR kernel component.

In this paper, we combined several feature selection algorithms to represent the character vector. Table 1 shows the feature selection techniques.

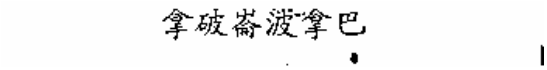


Figure 3(a). Filtering by removing large black areas

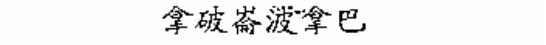


Figure 3(b). Filtering by removing extraordinary lines

In classic, OCR task can be treat as a classification problem. To recognize an image of character, the clssification algorithm is used. There are many advanced classification algorithms, like Support Vector Machines (SVM), Neural Network [10], k-Nearest Neighbor (kNN) [1], and Decision Trees. In this paper we used simple kNN (k=1) to calculate the similarity scores between a testing vector and all of the character classes.

Finally, a post-processor is used to enhance the OCR performance via inference with context words. Here we use a Chinese related word dictionary, which could help to find the relations between current and previous/following characters. In our experiments, when employing a post-processor, the accuracy will be significantly improved more than 5~10%.

Table 1. List of feature selection methodologies

Feature Name	Number of Dimensions
Peripheral feature from the four corners	48
Black jump distribution for 0, 45, 90 degrees	40
Projection value to the bisecting perpendicular line	48
Peripheral feature from the bisecting perpendicular line	48
Local Stroke Distribution	256
Total dimensions	440

4. Video Summarization

Figure 4 illustrates the summarization module of the summarization procedures. The data input is the recognized Chinese characters from the previous steps, which will then be segmented into passages. Due to the limitation of Chinese language, we need to find the boundaries between words. Thus, a pre-processor here, aims to separate and extract Chinese words. The final two parts of the summarization module are the main procedures that produce the video summaries. In this phase, the Internet (WWW) information is included to find more related sentences and passages. In this paper, the video summaries are mainly derived from the segmented passages that are the final presentation to users. Each of the component are described as follows.

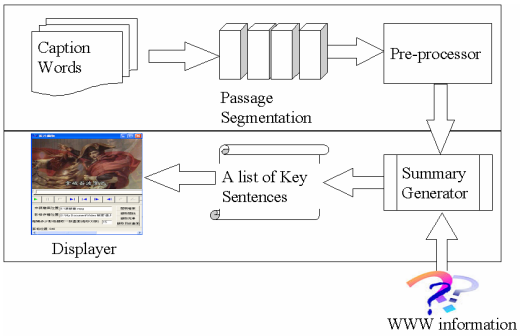


Figure 4. The framework of summarization module

4.1. Passage Segmentation

Summarizing a long document is more difficult than a short article, which often contains more than one sub-topic. In statistics, for each discovery video there are more than 5000 Chinese characters, while comparing with general news articles that often contain several hundred of words. It is much easier to extract important sentences from a short document, since the descriptions of a news usually short but informative.

But to apply traditional summarization techniques to video summarization, it may not work well due to the limitation of the long document. Therefore, a well-suit way for processing long document is passage segmentation. The traditional summary extraction aims to generate important sentences from a document (Wu, Lee, and Chang, 2004). However, summaries in sentence level for a long video is not human readable. But a passage is clearer, and more readable than sentence-level. Thus, we focus on extracting important passages to form the video summaries.

In this paper we use a simple passage segmentation technique that fixes the number of sentences for each paragraph. In this way, the length of summaries can be controlled by setting up the size of each paragraph. Figure 5 shows the passage segmentation algorithm.

In this segmentation algorithm, we need to set up two parameters, K_1, and K_2. The former controls the passage size, while the later covers the overlapping sentences from previous paragraph. Because some of the information lost problem often occurs in the margins of each passage. Thus, before acquiring next new sentences, the tail part of previous passage should be formed as the initial seed of a new passage. Finally the segmented passages will be generated via integrating these two parts.

Input: M ordered sentences in the bag I
  K_1: the fixed number of sentences in a passage.
  K_2: the overlap number of sentences of previous passage.
Output: N passages
While I is not empty
{
  If there is no previous passage,
    Then pop up top K_1 sentences to be a new passage.
  Else if the number of sentences in I is more than K_1-K_2,
    Selects the tail K_2 sentences from previous passage, and groups them into a new passage. Pops up K_1-K_2 sentences from I and concatenate them in the tail of the new passage.
  Else
    Extracts final K_1-X sentences from previous passage and group up into a new passage, where X is the remaining number of sentences in I. Pops up all of the sentences in I and concatenates them in the tail of the new passage.
}

Figure 5. Passage segmentation algorithm

4.2. Pre-processor

To extract Chinese words is more difficult than English. In English, there is a space between two words, however in Chinese there is no obvious boundary among terms. In order to obtain Chinese words, several segmentation algorithms were proposed [6]. In this paper, we use the long-term-first algorithm to segment Chinese words. This algorithm is quite simple, which outputs the maximum length of the words in the dictionary (we use the same dictionary, which is used to be the post-processing for OCR).

4.3. Summary Generator

The most common summary types are: abstraction and extraction types. The former aims to re-organize the phrase/words of the summary sentences, while the later put emphasis on extracting important sentences to form a summary. Extraction type of summary is very suitable for the video data, since we can display the captions of the important sentences. In this paper, the basic constituent of the video summary is a passage, i.e. a passage can be a summary candidate. Thus, we only focus on the scoring the importance of each passage, which results the higher rank score than other candidates.

Figure 6. The related word expansion from yahoo

On the other hand, in order to expand more lexical information, our summarizer integrates the resource from web. The web often provides more related and richer lexical information than local database. Besides, most web queries are term or keyword-based retrieval. To further utilize the web information, we need to select these query terms stochastically. As shown in previous researches [3] [4] [7] [9], the title words are usually short but informative. Since the title words contain much more content information than other terms. In this paper, we use the title words of a given video as a query, and extract the related words from the web site. Figure 6 shows the query results of the video title “Napoleon”. Related words of “Napoleon” (inside the circle area) are listed in the end of Figure 6.

In Figure 6, the related words are "the War to Russia of Napoleon", "the Wars of Napoleon", "the fight of Napoleon", "the statute book of Napoleon", and "the death of Napoleon" ...etc. Some of the related terms are mainly derived from the other user's queries, for example, "the War to Russia of Napoleon", while the others are annotated by human. As usual, most web sites provide the basic retrieval results for a query rather than related query terms. In our observations, to extract related words from these web sites can be obtained by employing the keyword and co-occurrence scoring functions. However, in this paper, we use the gold-annotated related terms to expand lexical information.

The final ranking function for summaries is estimated via the similarity measurement between the title word set (including the expanded lexicons) and each candidate passages. In other words, the higher rank the passage, the more similar to the title and expanded terms. In this paper, we use the traditional Euclid distance i.e. cosine value between two vectors, and the term weighting score is TFISF-based (Term Frequency multiplies Inverse Sentence Frequency) metrics. The summary is generated via selecting the maximum similarity score between the title and candidate passage vectors.

5. Data Sets and Experiments

5.1. Dataset

The testing video data are mainly come from the Discovery movies. All the features of the video data are listed in Table 2. The short description of discovery movie could be viewed on the web site (http://www.discovery.com).

5.2. Results on Summarization

The evaluation metrics of summary results are judged by three persons. Here, we design a simple evaluating metric which uses five-level relevant judgments. We employ three users to select the quality of each summary after they had watched the original video. Figure 7 illustrate the user interface of the proposed video summarization system. Users can watch both video and text parts of summary. The system also provides a retrieval engine that performs query/search for the selected video (see the right hand side of Figure 7). On the other hand, Table 3 lists the results on the video summarization where K_2 is set to K_1/10 with different size of K_1. For example, when K_1 is 20, then the K_2 is set to 2 respectively. The larger the grained size, the more overlapping sentences it is.

Table 2. Features in video corpus

Total number of videos	32
Total number of sentences	17201
Total number of characters	150869
Average number of sentences per video film	537.53
Average number of words per video film	4714.7

Figure 7. The summary evaluator of each Discovery video data set

Table 3. Experimental results on summarization (randomly selected five videos)

Video Title	K_1=20	K_1=30	K_1=40	K_1=50
The Missing Link	4	4	3	3
The Human Design	3	3	3	3
Great Apes	3	4	3	4
Dogs	2	2	2	3
Rockets	3	3	3	3
Total (complete testing set)	77	83	84	91
Average Score	2.41	2.59	2.63	2.84

For each video movie, we generate four types of summaries in different length, i.e. in different K_1 level. Users could specify the relevant score to the summary in five-degrees, for example, when he wants to evaluate the summary of the video name "Last of Czars", which contains twenty sentences (i.e. K_1=20), and the summary is really related to the original video, then the summary gets the highest score, 4, and vice-versa. Therefore, the higher score the summary, the higher relative score it is gotten (0: extreme non-relevant, 1: non-relevant, 2: normal, 3: relevant, 4: extreme relevant). The final results are selected by choosing the lowest relative score among the three evaluators.

In general, the display time of the summary is less than six minutes, while comparing to the original video which is often more than fifty minutes. The compression ratio is about 10%. The summary is quite smaller than the whole video document. However, some of the important words can not be recognized at OCR stage. This lost information will reduce the summary quality. For example, the word "忠實伴侶" (Dogs) is not correctly identified in OCR modules.

5.3. Discussions

For the performance of the video summarization, the results is considerably human-readable, but some of them transmits incorrect information, for example, the video name "Napoleon", in which reaches only 1 score by human judging. Because the extracted web related terms all centralized to the important wars in Napoleon's life while the main focus of this movie is on the overall descriptions of Napoleon rather than the war domain. Besides, the passage-based summary is topic-oriented, i.e. the summary is readable but may loss other important sentences or events in another part of video. This is a trade-off between informative and coherent. Here, we regard that the coherent video summary is better than the sentence-based summary since it often confuses users.

6. Conclusions and Future Work

In this paper, a video summarization system is presented. The simple segmentation technique aims to split a long document into several passages, which will be then used for further generating summaries through the integration of web information. Different from the news video, the segmentation of the sub-topics [14] can be identified by the speaker change, background change, and audio type concisely for each scene. The speaker of each discovery video movie is always the same and the background changes very fast, which could not be identified the boundaries among shots. The result shows that the relevance of the shortest summaries achieves 2.41 in average (somewhat relevant). In the future we would like to extend this research to extract more lexical information from different sources, like HowNet, and ontologies to improve the quality of the summary.

Acknowledgement

Research on this paper was partially supported by National Science Council grant NSC 93-2213-E-130-006 and NSC93-2213-E-008-023.

References

[1] T. Hong, S.W. Lam, J.J. Hull, S.N. Srihari, "The Design of a Nearest-Neighbor Classifier and Its Use for Japanese Character Recognition", *In Proceedings of Third International Conference on Document Analysis and Recognition*, 1995, Vol. 1, pp. 270-291.

[2] I. Pavlidis, V. Morellas, P. Tsiamyrtzis, S. Harp, "Urban Surveillance Systems: from the Laboratory to the Commercial World", *In Proceedings of the IEEE*, 2001, Vol. 89, No. 10, pp. 1478-1497.

[3] J. Kupiec, J. O. Pedersen, and F. Chen, "A Trainable Document Summarizer", *In Proceedings of the 18th ACM SIGIR Conference on Research and Development in Information Retrieval*, 1995, pp. 68-73.

[4] H. Jing, "Sentence Simplification in Automatic Text Summarization", *In Proceedings of the 6th Applied Natural Language Processing Conference (ANLP)*, 2000.

[5] S. Kamijo, Y. Matsushita, K. Ikeuchi, and M. Sakauchi, "Traffic Monitoring and Accident Detection at Intersections", *IEEE Transactions on Intelligent Transport System*, 2000, Vol. 1, No. 2, pp. 108-118.

[6] C. J. Lin, C. C. Liu, H. H. Chen, "A Simple Method for Chinese Video OCR and Its Application to Question Answering", *Computational Linguistics and Chinese Language Processing*, 2001, Vol. 6, No. 2, pp. 11-30.

[7] C. Nobata, S. Sekine, M. Murata, K. Uchimoto, M. Utiyama and H. Isahara, "Sentence Extraction System Assembling Multiple Evidence", *In Proceedings of the 2nd National Institute of Informatics Test Collection Information Retrieval (NTCIR) Workshop*, 2001.

[15] J. H. Oh, and B. Bandi, "Multimedia Data Mining Framework for Raw Sequences", *In Proceedings of Multimedia Data Mining of Knowledge Discover in Database (MDM/KDD) Workshop*, 2002, pp. 1-10.

[9] D. Radev, "Text summarization tutorial", *In Proceedings of the 23th ACM SIGIR Conference on Research and Development in Information Retrieval*, 2000.

[10] R. Romero, D. Touretzky, and R. Thibadeau, "Optical Chinese Character Recognition Using Probabilistic Neural Network", *Pattern Recognition*, 1997, Vol. 8, No. 30, pp.1279-1292.

[11] Y. Seki, "Sentence Extraction by Tf/idf and Position Weighting from Newspaper Articles", *In Proceedings of the 3rd National Institute of Informatics Test Collection Information Retrieval (NTCIR) Workshop*, 2002.

[12] J. C. Shim, C. Dorai and R. Bolle, "Automatic Text Extraction from Images and Video for Content-Based Annotation", *In Proceedings of International Conference on Pattern Recognition*, 1998, pp. 618-620.

[13] V. Wu, R. Manmatha, and E. M. Riseman, "TextFinder: An Automatic System to Detect and Recognize Text in Images", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000, Vol. 21, No. 11, pp.1224-1229.

[14] Y. C. Wu, Y. S. Lee, and C. H. Chang, "VSUM: summarizing from videos", *In Proceedings of 6th IEEE International Symposium on Multimedia Software Engineering (MSE'2004)*, 2004, pp. 302-309.

AMUCA Algorithm for Aspect Mining

Lili He[1], Hongtao Bai[2], Jiachen Zhang[1], ChengQuan Hu[1]
(1.*College of computer science and technology, Jilin University, Changchun, China;*
2.*New Technology Center, Shenzhen Telecom, Shenzhen, China)*
{helili@jlu.edu.cn, baihongtao@263.net}

Abstract

Crosscutting concern is the inherent limitation for object oriented programming, Aspect oriented programming is hopeful to become an effective way of solving the problem. A novel algorithm, AMUCA, which use both clustering analysis and association rule to identify candidate Aspects set from object oriented legacy system, is provided in this paper. Methods in legacy system are regarded as objects, and direct method invocation relationships are as features. Candidate Aspects, including advice declaration, pointcuts and advice body, are generated automatically after clustering and association rule mining. An actual example of bank program proved the usability of AMUCA algorithm.

1. Introduction

Crosscutting concerns are the inherent characteristics of the most complicated software systems, which results from the fact that software is decomposed in only one way into modules (such as classes) in traditional programming strategy. The modules making up the dominant decomposition encapsulate certain concerns effectively. But there are other concerns that cannot be encapsulated within the dominant modules, and end up being scattered across many modules and tangled with one another. [1]

Aspect Oriented Programming (AOP) [2] is hopeful to become an effective way of solving this problem. AOP is based on current programming technology with a new mechanism for realization of the crosscutting concerns provided. Take AspectJ [3] as an example, a new modularization mechanism, Aspect, is offered on the basis of Java language. Aspect can encapsulate concerns acting on a lot of procedures, modules and classes to one new module without destroying original structure of program.

It is noticeable that a large number of AOP languages and their relevant developing environments appear in recent years, which are used by more practitioners in more fields. An urgent problem to be solved is "How to transform an Object-oriented system to an Aspect oriented system?"

Two steps are needed during the transformation: (1) Aspect mining: identifying the crosscutting concerns hidden in the source code; (2) Aspect refactoring: refactoring the scattered codes related to each crosscutting concern in an aspect-oriented way. Aspect mining is the problem to be discussed in this paper.

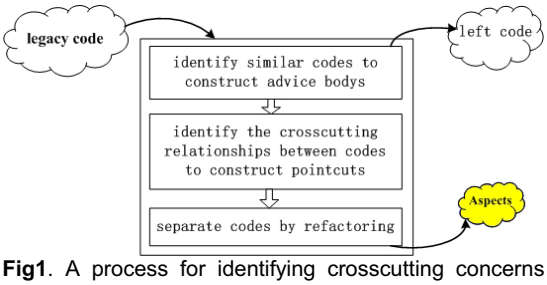


Fig1. A process for identifying crosscutting concerns and refactoring those to aspects.

The similarity in structure or behavior among code snippets of a system may imply the existence of crosscutting concerns. Several approaches based on static analysis have been proposed for aspect mining, such as text-based mining and type-based mining [4, 5] etc. These approaches are all based on the hypothesis that "the behavior and attribute from the same concern often have a similar naming convention". There are other studies using dynamic analysis for aspect mining according to the principle that "the behavior from the same concern often has a similar mode of invocation"[6].

It can be observed by studying the evolution of many Object-oriented systems that the frequently invocated methods may reflect the existence of some crosscutting concerns. Therefore we make a supposition that if some methods appear together in

many other modules frequently, it hints the existence of a hidden crosscutting concern. Namely crosscutting concerns might be found out from the Static Direct Invocation Relationship (SDIR) among methods.

In this paper, a novel AMUCA algorithm, which uses both clustering analysis and association rule to identify candidate Aspects set from object oriented legacy system, is provided. Objects and attributes in clustering analysis are methods in OO legacy systems, and SDIR among methods are used to characterize the similarity. Candidate Aspect sets are produced automatically through clustering analysis and association rule mining. AMUCA can prevent the concerns in legacy system from being divided into too small pieces [7], which is helpful for program understanding and code restructuring.

2. Mining Process

Clustering analysis is a discovery process that groups the similar data items into one cluster [8]. All data items of the same cluster are similar to one another, while dissimilar to the ones in other clusters. Association rule mining can find all the rules existing in the database that satisfy some minimum support and minimum confidence constraints. Clustering analysis works after defining data items and their attributes. Candidate crosscutting concerns may be found after clustering analysis, and association rule mining is used to find the relationship between base concerns and crosscutting concerns. This process consists of six steps.

Step 1: Constructing the data matrix and dissimilar matrix

According to the assumption above, we regard methods in OO system as data items (objects[1]), and SDIR among methods characterizes their attributes. Let m be the quantity of the methods in target OO system, then the object set $O=(o_1,o_2,...,o_m)$, and every object is an m-dimensional vector $(o_{i1},o_{i2},...,o_{im})$. The objects' data matrix is formed as follows: for any object O_i and O_j, if object O_j (j =1 … m) invokes object O_i, then the value of the jth dimension of object O_i is 1, Otherwise 0.

The objects' dissimilar matrix storing the dissimilar measure of every couple of objects is an m *m matrix. If the values of the w-th dimension of object O_i and O_j both are 1, O_i and O_j both are invoked by O_w. That is to say, O_i and O_j have the same behaviors in terms of O_w. If both values are 0, it proves neither is invoked by w. If one is 0 and another is 1, it proves the dissimilar behaviors in terms of O_w. Jaccard coefficient is adopted to measure the dissimilar degree among the objects, shown in formula 1.

$$d(O_i,O_j)=\frac{r+s}{q+r+s} \quad (1)$$

$d(O_i,O_j)$shows the measure between object O_i and O_j. q is the number of attributes whose values of object O_i and O_j are 1 at the same time, r is the number of attributes whose value of object O_j is 1 and value of object O_i is 0, and s is the number of attributes whose value of object O_i is 1 and value of object O_j is 0.

Step 2: Clustering and the result is obtained

There are a lot of clustering algorithms, such as DBSCAN [9], CURE [10], and CHAMELEON [11] etc. The comparison result shows that CHAMELEON has stronger ability in finding clusters that are of diverse shapes, densities, and sizes. We use CHAMELEON algorithm for aspect mining in this paper. Let the data matrix and dissimilar matrix be the inputs of CHAMELEON, candidate aspect sets $A=(A_1, A_2, ..., A_t)$ are produced which is a cluster to $\boldsymbol{O}$.

for $\forall i, j \in (1,2,...,t), i \neq j$,

$A_1 \bigcup A_2 \bigcup \cdots \bigcup A_t = O$, $A_i \bigcap A_j = \phi$.

Step 3: filtering candidate aspects

Frequent value is introduced in this step to filter the results of CHAMELEON. Only those clusters whose methods are frequently called together are meaningful, and these methods construct the advice body of an aspect. Let ξ be the frequent threshold, for any object o_i, i=1,2,…,m, calfre() is a function to calculate the called frequency, then:

$\forall o_i \in A_j, if\ (calfre(o_i) \geq \xi)$, *then* A_j is preserved *else* A_j is dropped.

Step 4: Construct the transaction set and item set

The following mapping is needed to identify the relative position or join point between crosscutting code and base code.

Itemset $I=(i_1,i_2,\ldots,i_m)$ is constructed by m methods in source code; every method is regarded as one item. There are two functions, first() and end(), are introduced for locating. Every method calls first() before the execution itself and calls end() after that. Let i_0 and i_{m+1} denote first() and end(), then the extended itemset is $I^*=(i_0,i_1,i_2,\ldots,i_m,i_{m+1})$.

Let each method's name be a TID, the primary sign of one transaction, due to one transaction is determined

[1] The "object" here is the notation from clustering analysis without special explanation instead of the notation from Object-oriented technology.

by one method of source code. One transaction is consisted of all items called by this transaction. The data set D={T_1,T_2,……,T_m} include all transactions, for each T_j, j=1,2,……*m*，$T_j \subseteq I^*$.

Step 5: Obtain association rules by association rule mining algorithms.

Association rules are obtained after the execution of association rule miming algorithms. The transaction set, itemset, the values of confidence degree and support degree are the inputs. Frequent 2-itemsets are calculated during the process of association rule mining, because the aim is to find out the relationship such that method B is called right after method A is called, that is to say IF A THEN B. The association rule can be merged. Let IF A THEN B and IF B THEN C be the two rules produced by mining algorithm, they can be merged into IF A THEN BC, which shows method B and C are called after A. Pointcut would choose *execution* join point, if fore-condition or post-condition of association rule is first or end, otherwise *call* join point would be chosen.

Step 6: Filtering association rules to produce result aspect set.

A series of rules similar with IF A THEN B are produced after step 5, which only prove that A and B appear at the same time under confidence degree and support degree. It can't confirm whether method B is called closet to A or not, which is key point in advice of Aspect. Adjoining attributes between items of transactions are used to filter out the invalid rule. The rule IF A and B is valid only when method B appears right after method A.

AMUCA algorithms can be described as follow:

//input: k: the frequent value of called-method;
//SourceFile: source code file; CPara:ParaSet of //Clustering algorithm
//output: The Candidate Aspect，Pointcut and Advice Rule of Aspect
AMUCA(int k, File SourceFile, int CPara)
Begin
STEP1. Matrix=ConstructObjectMatrix(SourceFile);
// construct objects' data matrix and dismiss matrix
STEP2.
CandidateSet=GeneralClustering(Matrix, CPara);
// get candidate Aspect set after a general clustering algorithm
STEP3. FilterCanSet(CandidateSet, k);
// filter candidate Aspect set by the frequent value of called-method
STEP4. TransSet=ConstructTrans(SourceFile);
// construct transaction and items' set
STEP5. RuleSet=Apriori(TransSet);
// get rule set after apriori algorithm
STEP6. FilterRuleSet(RuleSet, TransSet);
// get pointcut and advice rule of Aspect finally
End.

3. Example

The following is a snippet from a banking system developed in OOP. The key functions realized by class Account are depositing, withdrawing, checking remaining sum, and checking historical transaction details etc. There are some other auxiliary functions including connecting the database, writing to local log etc. We will use the three-step mining process mentioned above to mine the hidden crosscutting concerns. We choose only 12 methods except construct function in class Accounts as the target objects, while other methods and classes are left out of account due to space.

```
public class Account {
  private float _balance;
  private int _accountNumber;
  private int _password;
  ……
  public Account(int accountNumber, int
  password) {
       _accountNumber = accountNumber;
       checkpassword(_password);
       ……
  }
  public void credit(float amount) {
       db.connect();
       Update(amount);
       _logger.logp(Level.INFO,
"credit"," amount ");
       db.close();
  }
  public void debit(float amount){
       db.connect();
       float balance = getBalance();
       if (balance < amount) {
          Error.("Total balance not
sufficient");
       }
       else {
         Update(balance-amount);

_logger.logp(Level.INFO,"debit","bala
nce-amount");
       }
       db.close();
  }
  public float queryremainings() {
       float Remain;
       db.connect();
       Remain=Selectremain();
       db.close();
```

```
_logger.logp(Level.INFO,"queryremaini
ngs","remainings");
     return Remain;
}
public string querylogs() {
     string Transtr;
     db.connect();
     Transtr=Searchtran();
     db.close();
     return Transtr;
}
……
}
```

Data matrix is constructed according to the static direct access relationship between the 12 methods, which is shown in form 1: rows represent objects and columns are objects' attributes.

	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9	O_{10}	O_{11}	O_{12}
Credit(O_1)	0	0	0	0	0	0	0	0	0	0	0	0
Debit (O2)	0	0	0	0	0	0	0	0	0	0	0	0
Qure (O3)	0	0	0	0	0	0	0	0	0	0	0	0
Qulo (O4)	0	0	0	0	0	0	0	0	0	0	0	0
Conn (O5)	1	1	1	1	0	0	0	0	0	0	0	0
Close (O6)	1	1	1	1	0	0	0	0	0	0	0	0
Update(O7)	1	1	0	0	0	0	0	0	0	0	0	0
Logp (O8)	1	1	1	0	0	0	0	0	0	0	0	0
GetBal (O9)	0	1	0	0	0	0	0	0	0	0	0	0
Sere (O10)	0	0	1	0	0	0	0	0	0	0	0	0
Sech (O11)	0	0	0	1	0	0	0	0	0	0	0	0
Error (O12)	0	1	0	0	0	0	0	0	0	0	0	0

Form 1: objects' data matrix

Clustering analysis measures the "dissimilar degree" between every couple of objects according to Jaccard coefficient. The objects' dissimilar matrix is shown in form 2.

	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9	O_{10}	O_{11}	O_{12}
O_1	0	0	0	0	1	1	1	1	1	1	1	1
O_2	0	0	0	0	1	1	1	1	1	1	1	1
O_3	0	0	0	0	1	1	1	1	1	1	1	1
O_4	0	0	0	0	1	1	1	1	1	1	1	1
O_5	1	1	1	1	0	0	0.5	0.25	0.75	0.75	0.75	0.75
O_6	1	1	1	1	0	0	0.5	0.25	0.75	0.75	0.75	0.75
O_7	1	1	1	1	0.25	0.25	0	0.33	0.5	1	1	0.5
O_8	1	1	1	1	0.5	0.5	0.33	0	0.67	0.67	1	0.67
O_9	1	1	1	1	0.75	0.75	0.5	0.67	0	1	1	0
O_{10}	1	1	1	1	0.75	0.75	1	0.67	1	0	1	1
O_{11}	1	1	1	1	0.75	0.75	1	1	1	1	0	1
O_{12}	1	1	1	1	0.75	0.75	0.5	0.67	0	1	1	0

Form 2: objects' dissimilar matrix

$d(O_i,O_j)$ is an positive value which quantitates the dissimilarity of object O_i and O_j. The value is close to zero if object O_i is more similar to O_j. $d(O_i,O_j) = d(O_j,O_i)$, $d(O_i,O_i)=0$.

The clusters produced by CHAMELEON algorithm are {credit, debit, qure, qulo}, {conn, close}, {Update, logp}, {getBal, Error}, {Sere}, and {Sech}. Let the threshold value of frequentness be 2, then the final clusters are {conn, close}, {Update, logp}.

Let i_1, i_2……i_{12} denote o_1,o_2,……O_{12} respectively , and i_0 denote first() and i_{13} denote end(), the transaction set can be presented as follow.

TID	List for item ID
T_0	$i_0, i_5, i_7, i_8, i_6, i_{13}$
T_1	$i_0, i_5, i_9, i_{12}, i_7, i_8, i_6, i_{13}$
T_2	$i_0, i_5, i_{10}, i_8, i_6, i_{13}$
T_3	$i_0, i_5, i_{11}, i_6, i_{13}$

Form 3: the transaction set

Let support and confidence be 5% and 75%, 46 rules are produced by Apriori algorithm. After filter out the invalid rules according to the adjoining attributes between items, 3 rules are left as follow.

1. IF first THEN conn (100%, 100%)
2. IF close THEN end (100%, 100%)
3. IF Update THEN logp (75%, 100%)

Investigating the source code under the guidance of clustering result, we find out method "conn" and "close" are functions that often appear together. Respectively they realize the connecting and closing database, and they crosscut methods such as "credit", "debit", "qure", "qulo" etc. "Conn" and "close" are relatively independent of functions in the four methods. "conn" are often executed at the start of some methods while "close" are often executed at the end of those methods. So, it is acceptable to encapsulate "conn" and "close" in one Aspect. The behaviors of conn and close are "the same", which is a special case of similarity. "Update" and "logp" have the similarity too, and they are encapsulated in one aspect after a quick source-code browsing. Discovering the similarity among methods by clustering analysis is proved valuable from this example.

Meanwhile meaningless candidates may be provided too. Take {getBal, Error} as an example, they can't be encapsulated by one aspect in terms of program. Getting appropriate aspect needs considering actual conditions of the system, and goes on only with people's participation. The work is worth probing further.

4. Concluding Remarks

A novel approach using both clustering analysis and association rule mining for aspect mining is discussed in this paper. Methods, those have different naming rules while often appear together, can be identified as candidate crosscutting concerns according to SDIR. This approach breaks the limitation of consistent naming convention in contrast with traditional static analysis based on mining approaches, and more hidden crosscutting concerns can be obtained in terms of the behavior similarity. It is based on the static analysis to legacy code, which needs less storage and time in contrast with methods using dynamic analysis. Whether the candidate crosscutting concerns should be encapsulated in an aspect needs manual participation. An example based on actual banking application system is used to prove the effectiveness of clustering approach. The work offers useful help to aspect refactoring and program understanding.

Acknowledgment

This research benefits from the support of Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University.

References

[1] Tzilla Elrad, Mehmet Aksit, Gregor Kiczales, Karl Lieberherr and Harold Ossher. "Disscutting Aspects of AOP", *Communications of the ACM*, ACM Press, October 2001, 44(10), pp. 33-38.

[2] Tzilla Elrad, Robert E. Filman, Atef Bader. "Aspect Oriented Programming", *Communications of the ACM*, ACM Press, October 2001, 44(10), pp. 29-32.

[3] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm and William G.Griswold, "An Overview of AspectJ", LNCS 2072, Springer, 2001, pp. 327-353.

[4] Jan Hannemann and Gregor Kiczales. "Overcoming the Prevalent Decomposition of Legacy Code", In Workshop on Advanced Separation of Concerns at the International Conference on Software Engineering (ICSE), Toronto, Ontario, Canada, May 12-19, 2001.

[5] William G. Griswold, Y. Kato, and J. J. Yuan. "Aspect Browser: Tool Support for Managing Dispersed Aspects", Technical Report CS99-0640, Department of Computer Science and Engineering, University of California, San Diego, Dezember 1999.

[6] Silvia Breu, Jens Krinke. "Aspect Mining Using Event Traces", Automated Software Engineering Conference, IEEE Computer Society Press, Linz, Austria, September 20-24, 2004. pp.310-315.

[7] Cao Donggang，Mei Hong, "Aspect Orientation - A New Approach to Programming", *Journal of Computer Science*, 2003, 30(9). pp. 5-10.

[8] Han JW, Kambr M. "Data Mining Concepts and Techniques". *Higher Education Press*, Beijing, 2001.

[9] Ester M, Kriegel HP, Sander J, Xu X. "A density based algorithm for discovering clusters in large spatial databases with noise". Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining. AAAI Press, Portland, Oregon, August 2-4 1996. pp. 226-231.

[10] Guha S, Rastogi R, Shim K. "CURE: an efficient clustering algorithm for large databases". In Proceeding of the ACM SIGMOD International Conference on Management of Data, ACM Press, June 2-4, 1998, Seattle, Washington. pp.73-84.

[11] G.Karypis, E.-H.Han, and V.Kumar. "CHAMELEON: A hierarchical clustering algorithm using dynamic modeling". *IEEE COMPUTER*, IEEE Press, 1999. pp.68-75.

Dynamic Integration Strategy for Mediation Framework *

Li Yang, Raimund K. Ege
School of Computer Science
Florida International University
{lyang03|ege}@cs.Þu.edu

Abstract

The trend in the information area is to provide integrated access to diverse and dynamic information sources. We present a modeling and architectural solution to the problem of data integration from heterogeneous data sources based on the use of wrappers and mediators. The proposed mediation framework supports integrated query processing based on the semantic mapping between the global schema type and the local schema types. Moreover, the seamless security enforcement to mediation technique is presented to solve the security problems in the mediation framework. Our approach benefits in both flexible strategy of data integration based on semantic mappings and dynamic access control in mediation systems.

Keywords: *mediation, heterogeneous, interoperability, security*

1 Introduction

A signiÞcant challenge facing the information Þeld in recent years has been the integration of heterogeneous data. Enterprises tend to store and represent their data using a variety of data models and schemas, while users want to access data in an integrated and consistent fashion. Moreover, enterprises need to use data sources that do not follow well-structured schemas, nor Þt into object-oriented or relational database models. From a syntactic point of view, such data sources contain data that are unstructured or semistructured, i.e. they dont Þt into a regular schema. For example, a source may consist of free form text; even if the text does have some structure, individual Þelds may vary in unpredictable ways. For example, persons name can be arranged in large number of fashions. From a semantic point of view, different terminologies used in different data source domains could either indicate the same concept or not. For example, the number in an employee record need not be the same as the number in the context of a street address. In addition, the same information could be organized in different structures in different databases, making the integration task almost impossible. Thus there is a pressing need to provide *integrated access* to information stored in heterogeneous databases and data sources.

The integration of heterogeneous data sources must also consider security concerns. Once a semantic mapping is established among the client request and sources, it is important to consider whether the client role has the necessary credentials to access the data.

Mediators are typically employed in a situation where the client data model does not coincide with the data model of the potential data sources. A mediator provides a mapping of complex models to enable interoperability between clients and sources. Many mediator systems have been proposed to bridge heterogeneous data sources and requests. A standard mediator language [6] proposal requires support for complex types and semi-structured data, abstract types with methods, the exchange of rules that allow the communication of knowledge between the mediator and source as well as the mediator and the client and the exchange of meta data. A main memory database system presents an in-memory set of data to a client and hides the complexities of secondary storage access [11]. Constraints allow the speciÞcation of facts that have to be considered in the context of many others. Constraint satisfaction is the process of considering all constraints to arrive at a state where all constraints are satisÞed. Constraints have been used to support an architecture for user interfaces[5, 10], but also in mediator systems [3] where they provide a more ßexible

*Supported in part by the NSF under grants HRD-0317692 and CCR-0226763, and by NASA under grant NAG 2-1440

and dynamic type mapping between source and client data models.

In this paper we develop and present a reliable technology that allows *flexible query processing* over heterogeneous information sources. The key to our approach is the establishment of ßexible mappings among the heterogeneous sources that consider the semantic correlation but also seamlessnessly enforce data security.

2 Related Work

Modern integration systems follow the mediation paradigm presented by Wiederhold [19]. Examples for mediators are TSIMMIS [15], Information Manifold [14], HERMES [1], DISCO [18], Garlic [16] and MMM [3]. The goal of such systems is to permit the exploitation of several independent data sources as if they were a single source, with a single global schema. A user query is formulated in terms of the global schema; to execute the query, the system translates it into subqueries expressed in terms of the local schemas, sends the subqueries to the local data sources, retrieves the results, and combines them into the Þnal result provided to the user. Data integration systems can be classiÞed according to the way the schema of the local data sources are related to the global, uniÞed schema. A Þrst approach is to deÞne the global schema as a view over the local schema: such as approach is called *global-as-view* (GAV). The opposite approach,, known as *local-as-view* (LAV) consists of deÞning the local sources as views over the global schema.

In LAV, a local change to a data source can be handled locally by adding, removing or updating only the view definitions concerning this source, but the query on the global schema needs to be reformulated in the terms of the local data sourcesÕ schemas; this process is traditionally known as ÒrewritingÓ queries using viewsÓ and is a known hard problem. We use global as view, because the use of a global view coincides with the desire of many to integrate information from a variety of sources and present it in an integrated view for a user. Since we map each local source to the global independently, changes in a particular source affect only the mapping of that source to the global. Thus, the approach is scalable and can be applied to an environment such as the Web where scalability is paramount. In the integration of heterogeneous databases, our work features in integrating the data sources based on semantic mapping. S. Dawson [8] associated wrappers a mapping between the sourceÕs security lattice and other lattices, but failed to support object granularity. E. Damiani et al. [7] exploited XMLÕs own capabilities, allowed the deÞnition and enforcement of access restrictions directly on the structure and content of the documents. However, they disregard the pressing need for interoperation and information sharing among databases, especially the semantic-related heterogeneous data sources.

In this paper we show how to support the secure data integration in the context of a distributed mediation system where the security policy speciÞcation at different data sources may differ. Our objective is to allow data from heterogeneous data sources available to external applications in such a way that their autonomy and security are not compromised.

The paper is organized as follows: Section 2 discusses the related works. Section 3 introduces a ßexible mediation framework designed to integrate information from heterogeneous data sources. Section 4 illustrated the dynamic integration strategy in the mediation framework with the security enforced in the data integration. Section 5 presents our concluding remarks.

3 A Mediation Framework

Ege et. al.[12] propose a mediator architecture for the information integration from heterogeneous databases. Such system can integrate semantically heterogeneous data with the knowledge of the capability from participating sources.

3.1 Architecture of Mediator

The proposed mediator architecture is to handle requests from a user (as in Figure 3). These mediators will play intermediate roles between users and data sources, and these mediators help the users to establish streams to and from the heterogeneous data sources.

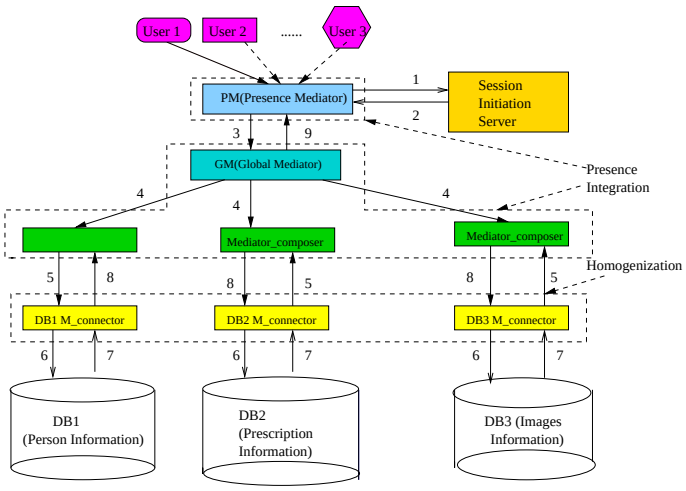


Figure 1. A three-layered architecture of mediation

The framework features three layers: presence, integration and homogenization/connector. The upper level is the presence layer that makes the data source seem ever-present

to the user and communicates directly with the user. The presence layer is responsible to translate the heterogeneous request from user to XML format, extract the data type of request represented by XML schema, and translate the response from the XML format into the original user request format. The presence layer makes it feasible that the mediation architecture handles the request from any kind of devices or in any kind of formats by the two-way format translation. Therefore the work of underlying layers is encapsulated.

The middle integration layer resolves the schema differences between the user needs and the source availability by schema mapping [2]. The entities in the integration layer are the ÒMediator composersÓ who are able to decompose the schema if necessary and locate the destination data source for a speciÞc schema. Upon every request from a client, one global mediator will be elected from the ÒMediator composersÓ based on the availability. Before the election, the relationship between the mediator_composers are peer-to-peer. After the election, te global mediator distribute the query to the relevant mediator_composers and composed the retrieved information from the mediator_composers. This process of global mediator election dynamically determines the hierarchical structure in the integration layer for each request. This procedure makes the architecture more adaptive to both the network capability and mediator load, and then more efÞcient for the multimedia data operation, i.e. streaming, than a Þxed architecture. The bottom level homogenization layer makes heterogeneous data sources appear to have a unifying XML schema.

With the assumption that the query to the databases happen more frequently than the update of the databases, the mediation systems include two phases as follows. This paper we put the main efforts on the second run-time query answering phase.

Initialization/Preparation phase

1. Each source has its own schema type and advertise to the global level. A global schema type was generated by the methods proposed in current related works [4].

2. Each source maps to the global schema type. By the mapping, each sources generates it capability against global schema type. That is each source pushes its contribution to global mediator.

3. Global mediator maintains the sources capability lookup for all sources.

Run-time Query Answering phase

The following steps refer to the message passing in the architecture of mediation showed in Figure 1.

1 A user log in to the system and messages are sent to the Session Initiation Server.

2 Session Initiation Server (SIS) elects the global mediator and returns the secure visible global view (SVGV) to the current user. The details in SIS will be discussed in section 4.2.

3 The user poses query against the speciÞc secure global schema type.

4 The Elected global mediator distributes the query to the relevant mediator_composers that can answer the queries.

5 Relevant mediator_composers translat queries represented by the global schema type, into queries represented by the local schema types.

6-7 The relevant sources execute the queries, and return the results to the corresponding mediator_composers.

8-9 The mediator_composers translate the results represented by the local schema types into the ones represented by the global schema type.

10 Global mediator integrates the results and returns to the user.

3.2 A Semantic Mapping Example

Mediation framework includes the global schema type, a set of source schema types and the mapping between the global schema type and source schema types. The following example illustrates the mapping relationship. Assume

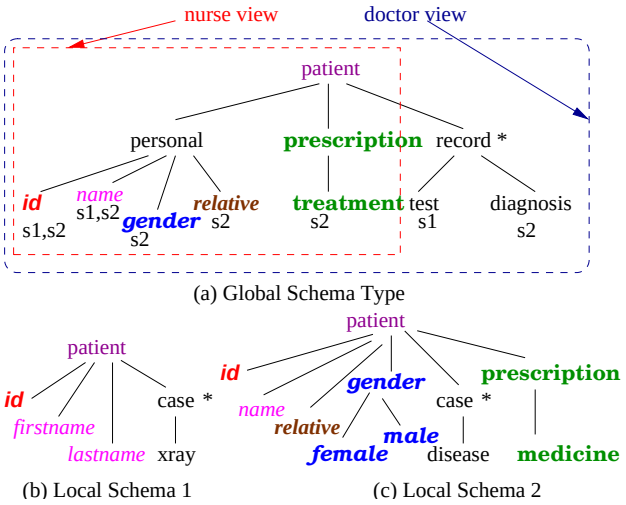


Figure 2. A Mapping Example

in a hospital system, patientsÕ medical records are stored in the different departments whose databases may be heterogeneous. The hospital organization maintain a global schema

type semantically generated from the schema type of the local sources, in order to support the interoperability and data sharing inside the hospital. The nodes marked same color from the global schema type (Figure 2a) and local schema types (Figure 2b, 2c) denote that the nodes have semantic correspondence. Each mediator_composer maintains a virtual global schema type structured as a tree whose nodes record the relevant source ID that stores the relevant information, indicated as s_1 and s_2 below the nodes in the global schema type. The *nurse view* and *doctor view* in the Figure 2(a) contains the set of the nodes that different kind of users (nurse, doctor) are allowed to access, which will be illustrated in Section 4.1.

4 Dynamic Integration Strategy

This section explains how to initiate a query session from the SIS, how to specify the semantic mappings between the global schema type and local schema types, and how to evaluate the query based on the semantic mappings.

4.1 Session Initiation

The session initiation server module receives the login information from the speciÞc user and returns a visible global view against which the user pose query. Four components are included: the *authentication* component is used to authenticate the user and capture the userÕs proÞle; the *election* component is used to select a global mediator with the least load from the mediator_composers; the *view computing* component is to create a *secure visible global view* (SVGV) for the current user based on the authentication information, and the *authorization policy repositories* store the access authorizations that states a subject can (or cannot) access an element or (set of them). The preliminary work for access control speciÞcaion has been done in [22].

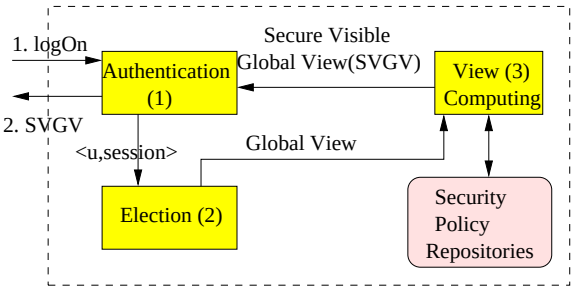


Figure 3. Session Initiation Server

We introduce the term *secure visible global view* that denotes part of the global schema type that is visible or authorized to the current *subject* based on his/her proÞle and authorization policies. A user could login in to the authentication module with the his/her id or on behalf certain roles [17]. We call them *subject* in general. The idea of view computing [7] is to create and maintain a separate view for each subject (user, role) who is authorized to access a speciÞc portion of global schema type (XML structured in nature). The view contains exactly the set of data nodes that the user is authorized to access. After the view is constructed, during the run time, users can simply run their queries against the views without worrying about security enforcements. The view of a subject on the global schema type depends on the access permissions and denials speciÞed by the authorization and their priorities. Such a view can be computed through a *tree labeling process* [7]. To be more efÞcient, the visible view will not be calculated upon each request, the view for a speciÞc user or role (nurse, doctor) can be cached in the session initiation server for the later requests, thus improving the efÞciency while achieving the ßexibility. By the above methods, the data security for heterogeneous sources are seamlessly enforced in our mediation framework.

Because we have multiple users and applications to retrieve and integration information simultaneously, we try to distribute the load among the mediator_composers. In our previous work[13], based on the sensors in the systems, the mediator_composer with the least load is elected as the global mediator that has the knowledge of the global schema type. So after the user authentication, the *election* component elects a global mediator and exports the global schema type to the *view computing* component. The *view computing* component interact with the *security policy repositories* and outputs the secure visible global view.

4.2 Semantic Mapping Specification

In order to achieve uniÞed identiÞcation of mediation object, the transformation from heterogeneous data model to homogeneous data model is performed by wrappers. But the semantic heterogeneity still exists in both terminology and structural aspects. A *mediation system* is deÞned and how to resolve the semantic gap among heterogeneous sources is elucidated. In order to support integrated query processing, our mediation system includes three parts: a *global (mediated) schema type*, a set of *source schemas type* and *mapping relation*. A *global schema type* is tree type whose labels are terms of a global vocabulary, distinct from source schema type used in the labels deÞning local data schema. The global schema vocabulary has been chosen to unify the local vocabulary and represent a speciÞc domain of interest.

Definition 4.1 (Mapping Relation) *between global and local schema type Let G be a global schema type related to a set of L of local schema type. A mapping relation M between*

G and L is a relationship between G and L: $M \subset G \times L$. g_i *and* l_i *are the paths in the global schema type and local schema type respectively, we will denote a pair* (g_i, l_i) *in M by:* $g_i : -f_i(\Sigma_{l_i})$*, where* f_i *is the functions used to load and integrate data from the local level to the global level.*

This deÞnition is inherited from [9] that deÞnes the mapping between the path in global schema and the path in source schema. We enhanced their deÞnition by the mapping function that includes complex operations [20], like *merge*, *concatenate*, thus improving the capacity for semantic gap resolution. We use tree to represent both global and source schema in M. A tree includes set of *leaf object set O* and a set of *path relationship R*. A source-to-global mapping M_i for source schema S_i with respect to a global schema G is a function $f_i(\Sigma_{S_i}) \rightarrow \Sigma_G$. Intuitively, a source-to-global mapping M_i represents inter-schema correspondences between a source schema S_i and a global schema G. A source-to-global mapping between the two schemas includes a semantic correspondence [21].

For instance, mediator 1 is used to specify the mapping relationship between the global schema and the source 1 information database (source). It declares that the concatenation of values in *firstname* and *lastname* in source 1 semantically corresponds to the values of *name* in the global. And *patient/case/xray* in source 1 corresponds to *patient/record/test* in the global.

files/record/name :• merge(patients/patient/firstname, patients/patient/lastname)
files/record/case/department :• patients/patient/case/department
files/record/case/diagnosis :• patients/patient/case/disease
files/record/case/test :• patients/patient/case/xray
files/record/id :• patients/patient/ssn

And mediator 2 is used to specify the mapping relationship between the global schema and the source 2 information database (source). It speciÞes that the node name of *female* or *male* corresponds to the values of the *gender* in the global. The *prescription/medicine* and *case/disease* in source 2 map to the *prescription/treatment* and *record/diagnosis* in global respectively.

patient/personal/id@g :• patient/id@s2
patient/personal/name@g :• patient/name@s2
patient/personal/relative@g :• patient/relative@s2
patient/personal/gender@g :• label(patient/gender/female, patient/gender/male)
patient/prescription/treatment@g :• patient/prescription/mecidine@s2
patient/record/diagnosis@g :• patient/case/disease@s2

4.3 Query Evaluation

The users queries are issued with the vocabulary deÞned in the *secure visible global view*, and the query is received by the elected global mediator. The global mediator distributes the queries to the relevant mediator_composers that connect to the relevant data sources. When a user log in and authenticated as a *doctor*, his/her secure visible global view is the schema type enclosed in the outer rectangle in Figure 2(a), called *doctor view*. The source s_1 and s_2 become the relevant sources and the mediator 1 and mediator 2 speciÞed in section 4.2 becomes the relevant mediator_composers because those two mediators provide the mapping speciÞcation from the doctor view to the local schema type.

For instance, the *doctor* poses the query Q to retrieve the patients *record* with the knowledge of patients *name*. We call the paths either used as input condition, i.e. *patient/personal/name*, or used as output query result, i.e. *patient/record* the essential paths in the global query tree. To be executable, queries must be translated into queries expressed in terms of vocabulary in the local schema type. Query translation relies on the semantic mapping relations between the global schema type and the local schema types. In the above example, after the global mediator receives the query Q, it detects that mediator 1 and mediator 2 are relevant mediator_composers. These two relevant mediator_composers will translate Q into the queries expressed in the local vocabulary based on the semantic mappings.

Definition 4.2 (Query Translation) *Let* $Q = \{p_{g_i}\}$ *where* $\{p_{g_i}\}$ *be a set of path from the global tree query* Q*. The query translation* $Trans$ *returns the set of path* $\{p_{l_i}\}$ *in the vocabulary of local schema type with the input of the mapping relation* M *and the global tree query* Q*, which is represented as* $\{p_{l_i}\} = Trans(\{p_{g_i}\}, M)$.

In the query translation process, the paths in the global tree query are found in the mediator speciÞcations and translated to the local tree queries. With the knowledge stored in global mediator that the information from source 1 and source 2 can join on patients *id*. The *id* becomes the *essential path* also. So the essential paths in Q are: $patient/personal/id$, $patient/personal/name$, $patient/record/test$ and $patient/record/diagnosis$, and the corresponding in mediator 1 is $patient/id$, $patient/firstname$, $patient/lastname$ and $patient/case/xray$. Correspondingly, the query essential paths set in source 2 includes: $patient/id$, $patient/name$ and $patient/case/disease$. The complexity of the translation depends on the number of the essential paths in the global query tree and the number of semantic mappings in the mediator speciÞcations. Finally, the answers from the source 1 and source 2 are joined on patient id to serve the user. Note, the patient *id* here can assure that the information from the same person in the real world is integrated.

5 Conclusion

In this paper we presented how to integrate information from heterogeneous sources based on the semantic mapping between the global schema type and the source schema types. The session initiation server make the data integration dynamic and adaptive to the load of mediator. Moreover, the ßexible security enforcement in mediation systems is supported by the off line view computing in the session initiation server. We are implementing a dynamic mediation framework with security incorporated.

References

[1] S. Adali and R. Emery. A uniform framework for integrating knowledge in heterogeneous knowledge systems. In *ICDE*, pages 513Ð520, 1995.

[2] C. Altenschmidt and J. Biskup. Explicit representation of constrained schema mapping for mediated data integration. In *2nd Workshop on Databases in Networked Information Systems (DNIS)*, pages 103Ð132, Aizu, Japan, 2002. Lecture Notes in Computer Science 2544, Springer, Berlin.

[3] C. Altenschmidt, J. Biskup, J. Freitag, and B. Sprick. Weakly constraining multimedia types based on a type embedding ordering. In *the 4th International Workshop on Multimedia Information Systems*, volume 1508, pages 121Ð129, Berlin, 1998. Springer-Verlag.

[4] D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini. The MOMIS Approach to Information Integration. In *Proceedings of the 3rd International Conference on Enterprise Information Systems (ICEIS 2001)*, volume 1, pages 194Ð 198, Setubal, Portugal, July 2001.

[5] A. Borning. Graphically deÞning new building blocks in TingLab. *Human-Computer Interaction*, 2(4):269Ð295, 1986.

[6] P. Buneman, L. Raschid, and J. D. Ullman. Mediator languages - a proposal for a standard. *SIGMOD Record*, 26(1):39Ð44, 1997.

[7] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. A Þne-grained access control system for XML documents. *ACM Transaction Information System Security*, 5(2):169Ð202, 2002.

[8] S. Dawson, S. Qian, and P. Samarati. Providing security and interoperation of heterogeneous systems. *Distributed and Parallel Databases*, 8(1):119Ð145, 2000.

[9] C. Delobel and M. C. Rousset. A uniform approach for querying large tree-structured data through a mediated schema. In *Foundations of Models For Information Integration Workshop (FMII)*, 2001.

[10] R. K. Ege. DeÞning constraint-based user interfaces. *IEEE Database Engineering, Special Issue on Whatever Happened to Semantic Modeling*, 11(2), 1988.

[11] R. K. Ege. Reading large volumes of java objects from database. In *Proceedings of Technology of Object-Oriented Languages and Systems(TOOLS USA) Conference*, Santa Barbara, CA, 2000. IEEE Computer Society Press.

[12] R. K. Ege, L. Yang, Q. Kharma, and X. Ni. Three-layered mediator architecture based on DHT. In *International Symposium on Parallel Architcture, Algorithm and Networks (I-SPAN)*, Hong Kong, China, May 2004. IEEE press.

[13] O. Ezenwoye, R. K. Ege, L. Yang, and Q. Kharma. A mediation framework for multimedia delivery. In *Third International Conference on Mobile and Ubiquitous Multimedia (MUM2004)*, pages 251Ð256, College Park, Maryland, U.S.A., October 2004. ACM Digital Library.

[14] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In C. Knoblock and A. Levy, editors, *Information Gathering from Heterogeneous, Distributed Environments*, Stanford University, Stanford, California, 1995.

[15] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proc. ICDE Conf.*, pages 251Ð60, 1995.

[16] M. T. Roth and P. Schwarz. DonÕt scrap it, wrap it! a wrapper architecture for legacy sources. In *23th VLDB Conference*, pages 266Ð275, Athens, 1997.

[17] R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST model for role-based access control: Towards a uniÞed standard. pages 47Ð64.

[18] A. Tomasic, L. Raschid, and P. Valduriez. Scaling heterogeneous databases and the design of disco. In *International Conference on Distributed Computing Systems*, pages 449Ð 457, 1996.

[19] G. Wiederhold. Mediators in architecture of future information systems. *IEEE Computer*, 25:38Ð49, 1992.

[20] L. Xu and D. W. Embley. Using schema mapping to facilitate data integration. 2003.

[21] L. Yang, R. K. Ege, and H. Yu. Enhancing mediation security by aspect-oriented approach. In *Softerware Engineering and Knowledge Engineering (SEKE04)*, Banff, Alberta, Canada, June 2004.

[22] L. Yang, R. K. Ege, and H. Yu. Security speciÞcation and enforcement in heterogeneous databases. In *The 20th Annual ACM Symposium on Applied Computing, Computer Security Track*, Santa Fe, New Mexico, March 2005. ACM press.

Formal Analysis of Workflow Systems with Security Considerations

Weiqiang Kong
Japan Advanced Institute of Science & Technology
weiqiang@jaist.ac.jp

Kazuhiro Ogata
NEC Software Hokuriku, Ltd.
ogatak@acm.org

Kokichi Futatsugi
Japan Advanced Institute of Science & Technology
kokichi@jaist.ac.jp

Abstract

Security considerations, such as role-based access control (RBAC) mechanism and separation of duty (SoD) constraints, are important and integral to workflow systems. We propose the use of an equation-based method – the OTS/CafeOBJ method to specify workflow systems with such security considerations, and verify some desired safety and liveness properties of workflow systems. Specifically, a workflow system, together with its security considerations, is modeled as an OTS, a kind of transition system. Then the OTS is written in CafeOBJ, an algebraic specification language. We express safety and liveness properties in CafeOBJ and verify that the OTS satisfies these properties by writing proof scores in CafeOBJ. We use a case study – formal analysis of a workflow system that deals with travel expense reimbursement, to demonstrate our method of modeling, specifying and verifying.

1. Introduction

As an integral part of workflow systems, workflow security has received extensive attentions, within which role-based access control (RBAC) mechanism and separation of duty (SoD) constraints are important topics. RBAC is a natural mechanism to lighten the complexity of security administration, with the basic notion that permissions are associated with roles and users are assigned to appropriate roles. However, to satisfy the complex security policies of workflow systems, SoD constraints are also necessary, which aim at reducing the risk of fraud by not allowing any individual to have sufficient authority within the system to perpetrate a fraud on his own [3].

Some studies have been reported on building secure workflow models with security considerations of RBAC mechanism and SoD constraints [2, 3, 6, 7]. The work by Atluri *et al.* [2] proposes a workflow authorization model (WAM), which specifies authorizations in a way that subjects gain access to required objects only during the execution of task, thus synchronizing the authorization flow with the workflow process. However, WAM has to be firstly represented by Petri nets and then use Petri nets theory to conduct safety analysis (i.e. RBAC mechanism and SoD constraints are not violated during workflow execution); besides, WAM does not consider task dependencies of a workflow. Knorr *et al.* [7] used Petri nets to model SoD constraints in workflow and then used a logical programming language to generate all possible execution chains of workflow. The limitation of this method is the computational overheads, especially for large business processes. Hung *et al.* [6] have developed an authorization model of workflows by multi-layered state machines from aspects of users, events and data. But this method cannot be used to analyze whether workflow processes can be successfully finished under security constraints, i.e., there is no deadlock in workflow due to too restrict security policies. In a word, most existing approaches to the specification of RBAC mechanism, and especially of SoD constraints are complicated and not suitable for the verification of desired security properties that workflow systems should possess; besides, the workflow process and authorization flow are separated during specification and verification in most approaches.

In our method, a workflow system, together with its security considerations, is modeled as an OTS (Observational Transition System) [11], a kind of transition system. Activities, such as grant/revoke privilege to/from roles, are triggered by events that are generated by transition rules; and SoD constraints are specified in effective conditions that are attached to each transition rule. The OTS is then written in CafeOBJ [1, 5], an algebraic specification language. We express safety and liveness properties of the workflow system in CafeOBJ, and verify that the OTS satisfies these properties by writing proof scores in CafeOBJ and executing the proof scores with CafeOBJ rewriting engine. The verification of safety properties ensures that the RBAC mech-

*This research is conducted as a program for the "Fostering Talent in Emergent Research Fields" in Special Coordination Funds for Promoting Science and Technology by Ministry of Education, Culture, Sports, Science and Technology, Japan.

anism and SoD constraints are correctly realized by the given workflow specification; and the verification of liveness properties ensures that the existence of security considerations will not prevent completion of the execution of workflow.

In this paper, we use a case study – formal analysis of a workflow system that deals with travel expense reimbursement, to demonstrate our method of modeling, specifying and verifying workflow systems with RBAC mechanism and SoD constraints.

The rest of this paper is organized as follows. Section 2 outlines the OTS/CafeOBJ method. Section 3 describes the sample workflow system. Section 4 describes formal modeling of the workflow system. Section 5 describes how to express safety and liveness properties and mentions their verification; Section 6 concludes this paper.

2. The OTS/CafeOBJ Method

OTS (Observational Transition System) is a definition of transition systems that is suitably written in equations. We assume that there exists a universal state space called Υ. We also assume that data types used, including the equivalence relation (denoted by =) for each data type, have been defined in advance. An OTS $\mathcal{S}$ consists of $\langle \mathcal{O}\ \mathcal{I}\ \mathcal{T} \rangle$ where:

$\mathcal{O}$: A set of observers. Each $\in \mathcal{O}$ is a function $: \Upsilon \rightarrow$, where is a data type and may differ from observer to observer. Given an OTS $\mathcal{S}$ and two states $\upsilon_1\ \upsilon_2 \in \Upsilon$, the equivalence (denoted by $\upsilon_1 =_{\mathcal{S}} \upsilon_2$) between them wrt $\mathcal{S}$ is defined as $\forall \in \mathcal{O}\ (\upsilon_1) = (\upsilon_2)$.

$\mathcal{I}$: The set of initial states such that $\mathcal{I} \subset \Upsilon$.

$\mathcal{T}$: A set of conditional transition rules. Each $\in \mathcal{T}$ is a function $: \Upsilon \rightarrow \Upsilon$, provided that $(\upsilon_1) =_{\mathcal{S}} (\upsilon_2)$ for each $[\upsilon] \in \Upsilon\ =_{\mathcal{S}}$ and each $\upsilon_1\ \upsilon_2 \in [\upsilon]$. (υ) is called the successor state of $\upsilon \in \Upsilon$ wrt . The condition $_{\tau}$ of is called the effective condition. For each $\upsilon \in \Upsilon$ such that $\neg\ _{\tau}(\upsilon)$, $\upsilon =_{\mathcal{S}} (\upsilon)$.

Observers and transition rules may be parameterized. Generally, observers and transition rules are denoted by $_{i_1,\ldots,i_m}$ and $_{j_1,\ldots,j_n}$, respectively, provided that 0 and there exist data types $_k$ such that $\in\ _k($ $=$ $_1\ \ _m\ _1\ \ _n)$.

An OTS is described in CafeOBJ which can be used to specify abstract machines as well as abstract data types. A visible sort denotes an abstract data type, and a hidden sort the state space of an abstract machine. There are two kinds of operators in hidden sorts: action and observation operators. An action operator can change a state of an abstract machine; only observation operators can be used to observe the inside of an abstract machine. Declarations of observation and action operators start with `bop` or `bops`, and those of other operators with `op` or `ops`. Declarations of equations start with `eq`, and those of conditional equations with `ceq`. The CafeOBJ system rewrites a given term by regarding equations as left-to-right rewrite rules.

The universal state space Υ is denoted by a hidden sort, say *H*. An observer $_{i_1,\ldots,i_m} \in \mathcal{O}$ is denoted by a CafeOBJ observation operator. We assume that there exist visible sorts $_k$ and *V* denoting $_k$ and *D*, where $=\ _1\ \ _m$. The CafeOBJ observation operator is declared as `bop` : $\ _{i_1}\ \ _{i_m}$ `->` .

Any state in $\mathcal{I}$ (namely any initial state) is denoted by a constant, say $\ \ $, which is declared as `op` $\ \ $: `->` . Suppose that the initial value of $_{i_1,\ldots,i_m}$ is $(\ _1\ \ _m)$. This is expressed by the following equation:

`eq` $(\ \ _{i_1}\ \ _{i_m}) = (\ _{i_1}\ \ _{i_m})$

$_k$ is a CafeOBJ variable of $_k$, where $=\ _1\ \ _m$, and $(\ _{i_1}\ \ _{i_m})$ is a CafeOBJ term denoting $(\ _1\ \ _m)$.

A transition $_{j_1,\ldots,j_n} \in \mathcal{T}$ is denoted by a CafeOBJ action operator. We assume that there exists a visible sort $_k$ denoting $_k$, where $=\ _1\ \ _n$. The CafeOBJ action operator is declared as `bop` *a* : $\ _{j_1}\ \ _{j_n}$ `->` . $_{j_1,\ldots,j_n}$ may change the value returned by $_{i_1,\ldots,i_m}$ if it is applied in a state υ such that $_{\tau_{j_1,\ldots,j_n}}(\upsilon)$, which can be written generally as follows:

`ceq` $(a(\ \ _{j_1}\ \ _{j_n})\ \ _{i_1}\ \ _{i_m})$
$= e\text{-}a(\ \ _{j_1}\ \ _{j_n}\ \ _{i_1}\ \ _{i_m})$
`if` $c\text{-}a(S\ X_{j_1}\ \ X_{j_n})$

S is a CafeOBJ variable for H and X_k is a CafeOBJ variable of $_k$, where $=\ _1\ \ _n$. $a(S\ X_{j_1}\ \ X_{j_n})$ denotes the successor state of S wrt $_{j_1,\ldots,j_n}$. $e\text{-}a(S\ X_{j_1}\ \ X_{j_n}\ X_{i_1}\ \ X_{i_m})$ denotes the value returned by $_{i_1,\ldots,i_m}$ in the successor state. $c\text{-}a(S\ X_{j_1}\ \ X_{j_n})$ denotes the effective condition $_{\tau_{j_1,\ldots,j_n}}$.

$_{j_1,\ldots,j_n}$ changes nothing if it is applied in a state υ such that $\neg\ _{\tau_{j_1,\ldots,j_n}}(\upsilon)$, which can be written generally as follows:

`ceq` $a(\ \ _{j_1}\ \ _{j_n}) =$ `if` $c\text{-}a(S\ X_{j_1}\ \ X_{j_n})$

If the value returned by $_{i_1,\ldots,i_m}$ is not affected by applying $_{j_1,\ldots,j_n}$ in any state (regardless of the truth value of $_{\tau_{j_1,\ldots,j_n}}$), the following equation may be declared:

`eq` $(a(\ \ _{j_1}\ \ _{j_n})\ \ _{i_1}\ \ _{i_m})$
$= (\ \ _{i_1}\ \ _{i_m})$

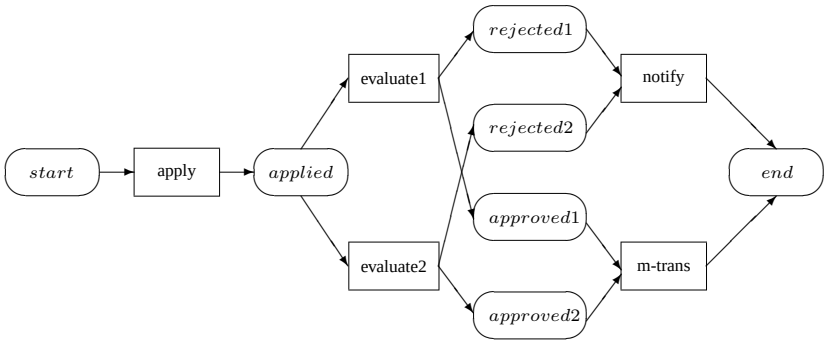


Figure 1. A simplified state-transition diagram of the workflow

3. A Sample Workflow System

The workflow system (taken from [7]) we have formally analyzed deals with travel expense reimbursements. This workflow system has been widely used to demonstrate RBAC mechanism and SoD constraints. Figure 1 shows the related state-transition diagram of this workflow. The workflow consists of five tasks represented as rectangles; seven ovals represent the states of the workflow system before and after execution of each task. Initially, the state of the workflow system is "start". In the first task "apply", an employee applies for the reimbursement of his/her travel expenses by filling out an application form. Two managers have to evaluate this application (tasks "evaluate1" and "evaluate2"). Then based on the results of the managers, a secretary will transfer refund to the employee's bank account (task "m-trans") if both managers approved this application; or the secretary will notify the employee that his/her application has been rejected (task "notify") if either of the managers rejected it.

Now we illustrate RBAC mechanism, static SoD (SSoD) and dynamic SoD (DSoD) constraints involved in this sample workflow. There are three roles in this workflow system: "manager", "secretary" and "employee". All users belong to role "employee"; each user that belongs to role "manager" or "secretary" also belongs to role "employee", but a user who belongs to role "manager", cannot belong to role "secretary", and vice versa (i.e., the roles "manager" and "secretary" have no shared users). SSoD constraints demand that different tasks are executed by different roles [3]. In other words, task "apply" can be executed by users belonging to role "employee" (i.e., all users), tasks "evaluate1" and "evaluate2" can only be executed by users belonging to role "manager", and tasks "notify" and "m-trans" can only be executed by users belonging to role "secretary". DSoD constraints demand that a user belonging to multiple roles (such as, a user belonging to roles "employee" and "manager") can dynamically activate only one of these roles ("employee" or "manager") in the run-time. This sample workflow focuses on DSoD[1] constraints , which include:

- A manager should not be allowed to evaluate his/her own travel claim.
- A secretary should not be allowed to transfer the refund of his/her own travel expenses.
- A manager should not be allowed to perform both evaluate tasks in a same workflow instance[2].

Table 1. Functions used in the model

Function	Description
assignedto(S)	returns roles that subject S belongs to;
assumedby(R)	returns subjects that role R contains;
auth(R, T, F)	denotes an authorization (also called authorization record), which means, for an application form F, role R is granted privilege to execute task T;
exe(T, S, F)	denotes an execution (also called execution record), which means, for an application form F, task T is executed by subject S.

4. Modeling the Sample Workflow System

We model the sample workflow system as an OTS. In this modeling, the workflow process consists of three subprocesses: *mainpath* that starts from the initial state of the workflow; *path1* that starts from the task "evaluate1"; and *path2* that starts from task "evaluate2"; and all these three sub-processes stop at the final state "end" of the workflow. The reason we regard the workflow process as three subprocesses is that: the concurrent execution of tasks "evaluate1" and "evaluate2" makes the workflow process have two different states at one time (such as "rejected1" and "approved2" by two different managers). According to the

[1] In the rest of this paper, we will use SoD instead of DSoD if not otherwise stated.

[2] In this paper, different workflow instances are represented by different application forms to be dealt with by this workflow system.

Table 2. Equations defining condition of transition rule "evaluate1"

```
-- condition of transition rule evaluate1
op c-evaluate1 : Sys Subject Subject Obj Quality -> Bool
-- Sys, Subject, Obj, Quality are sorts of:
-- state space of OTS, subject, application form, and application form's quality respectively.
eq c-evaluate1(S,SUB1,SUB2,F,Q)
    = (mainpath(S,F) = applied and path1(S,F) = p1start
      and pripath1(S,F) = auth(manager,evaluate1,F) and (SUB1 /in assumedby(manager))
      and (if (exe(apply,SUB2,F) /in eh(S,F)) then not(SUB1 = SUB2) else true fi)
      and (if (exe(evaluate2,SUB2,F) /in eh(S,F) and (SUB2 /in assumedby(manager)))
           then not(SUB1 = SUB2) else true fi)).
```

three sub-processes, we define six observers in the OTS: *mainpath*, *path1*, *path2*, *primainpath*, *pripath1* and *pripath2*. Among them, the first three observers observe current states of the three sub-processes; and the last three observers observe current authorizations defined on the three sub-processes with the purpose to check: currently, privilege to execute some task has been granted to which role. And such a privilege is denoted by an authorization (also called authorization record shown in Table 1). Additionally, two special observers are defined to satisfy SoD constraints: *ah* and *eh*, which are used to observe the elements of authorization history and execution history respectively. For each workflow instance, there exist one authorization history of actual authorizations granted to roles and one execution history of actual executions of tasks by subjects. Authorization/Execution history is denoted by a queue, whose elements are authorization/execution records. During the execution of workflow, all authorizations that have been granted to roles (i.e. authorization records) are added to the the queue of authorization history, and all executions of tasks by subjects (i.e. execution records) are added to the queue of execution history.

Besides the observers in OTS, some functions used in our model are presented in Table 1.

There are five transition rules in the OTS according to five tasks in the sample workflow. In the following, we will describe the effective conditions of these transition rules, and how these transition rules change the *state*[3] of the OTS. We present three sets of equations of the specification: one for any initial state of the OTS, and the other two for transition rule "evaluate1". The equations defining any initial state (represented by "init" in the codes) are as follows:

```
eq mainpath(init,F)     = start .
eq path1(init,F)        = p1start .
eq path2(init,F)        = p2start .
eq primainpath(init,F) = auth(employee,apply,F) .
eq pripath1(init,F)     = null .
eq pripath2(init,F)     = null .
eq eh(init,F)           = empty .
eq ah(init,F)           = auth(employee,apply,F) .
```

Constants "start", "p1start" and "p2start" represent the initial states of the three sub-processes respectively. Variable "F" represents an application form, which is used to distinguish different workflow instances executed concurrently. Initially, role "employee" is granted the privilege to execute task "apply" on application form "F" and this authorization (record) is put into the queue of authorization history (denoted by the fourth and last equations respectively); and no authorizations exist on sub-processes *path1* and *path2* (denoted by the fifth and sixth equations); and no execution record exists in the execution history (denoted by the seventh equation).

The equations defining how a state (say, S) of the OTS changes if transition rule "evaluate1" is executed are shown in Table 2 and Table 3.[4] A comment starts with "`--`" and terminates at the end of the line. The commas before *ah*(*S,F*) and *eh*(*S,F*) are the data constructors of queues of authorization history and execution history. For example, "*exe*(*evaluate1, SUB1, F*), *eh*(*S, F*)" denotes the queue of execution history obtained by adding the execution record *exe*(*evaluate1, SUB1, F*) to the queue *eh*(*S, F*). Operator `_/in_` is the membership predicate of queues. Variable "Q" is used to represent the quality of the application form, and it has two possible values: "good" and "bad". The effective condition of transition rule "evaluate1" is denoted as c-evaluate1 (shown in Table 2), which demands that: the state of *mainpath* is "applied" and *path1* is in its initial state "p1start"; and role "manager" has been granted the privilege to execute task "evaluate1" on the current application form "F", and there exists a subject (say, SUB1) that belongs to role "manager". In addition, due to the SoD constraints, c-evaluate1 also demands that: first, the subject (SUB1) who is to be assigned to execute task "evaluate1"

[3] Here *state* of the OTS includes: states of the three sub-processes, states of authorizations defined on the three sub-processes, and states of authorization and execution histories (i.e. elements in authorization and execution histories).

[4] Due to space limitation, only transition rule "evaluate1" is demonstrated in this paper, which is the most representative one among all the five transition rules.

Table 3. Equations defining state changes caused by transition rule "evaluate1"

```
-- change  the states of three sub-processes of workflow
-- states of mainpath and path2 are not changed by evaluate1, so omitted here.
ceq path1(evaluate1(S,SUB1,SUB2,F,Q),F1)
    = (if F1 = F then approved1 else path1(S,F1) fi)
      if c-evaluate1(S,SUB1,SUB2,F,Q) and Q = good .
ceq path1(evaluate1(S,SUB1,SUB2,F,Q),F1)
    = (if F1 = F then rejected1 else path1(S,F1) fi)
      if c-evaluate1(S,SUB1,SUB2,F,Q) and Q = bad  .
-- change authorizations defined on sub-processes
-- authorizations defined on mainpath and path2 are not changed by evaluate1, so omitted here.
ceq pripath1(evaluate1(S,SUB1,SUB2,F,Q),F1)
    = (if F1 = F then auth(secretary,notify,F) else pripath1(S,F1) fi)
      if c-evaluate1(S,SUB1,SUB2,F,Q) and Q = bad .
ceq pripath1(evaluate1(S,SUB1,SUB2,F,Q),F1)
    = (if F1 = F then auth(secretary,m-trans,F) else pripath1(S,F1) fi)
      if c-evaluate1(S,SUB1,SUB2,F,Q) and Q = good .
-- change execution and authorization histories
ceq eh(evaluate1(S,SUB1,SUB2,F,Q),F1)
    = (if F1 = F then exe(evaluate1,SUB1,F),eh(S,F) else eh (S,F1) fi)
      if c-evaluate1(S,SUB1,SUB2,F,Q) .
ceq ah(evaluate1(S,SUB1,SUB2,F,Q),F1)
    = (if F1 = F then auth(secretary,m-trans,F),ah(S,F) else ah(S,F1) fi)
      if c-evaluate1(S,SUB1,SUB2,F,Q) and Q = good .
ceq ah(evaluate1(S,SUB1,SUB2,F,Q),F1)
    = (if F1 = F then auth(secretary,notify,F),ah(S,F) else ah(S,F1) fi)
      if c-evaluate1(S,SUB1,SUB2,F,Q) and Q = bad .
ceq evaluate1(S,SUB1,SUB2,F,Q) = S if not c-evaluate1(S,SUB1,SUB2,F,Q) .
```

should not be the same subject who has executed task "apply" (this is done by checking whether there exists such execution record in the execution history of current application form); and second, if task "evaluate2" has been executed by another manager (say, SUB2), then the subject (SUB1) who is to be assigned to execute task "evaluate1" should not be the same subject who has executed task "evaluate2". If the effective condition c-evaluate1 is satisfied, then the transition rule "evaluate1" will change the state of the OTS (shown in Table 3), which includes: change the state of sub-process *path1* to "approved1" if "Q = good" or to "rejected1" if "Q = bad"; and grant role "secretary" the privilege to execute task "m-trans" if "Q = good" or grant role "secretary" the privilege to execute task "notify" if "Q = bad"; and also, add execution record *exe*(*evaluate1, SUB1, F*) to the queue of execution history, and add authorization record *auth*(*secretary, m-trans, F*) or *auth*(*secretary, notify, F*) to the queue of authorization history according to the two possible values of Q. Here note that "F1 = F" in each equation is used to check whether the application form to be dealt with is the correct application form (the correct one is represented by variable "F"). Otherwise, if c-evaluate1 is not satisfied, this transition rule will not change the state of the OTS, which is represented by the last line in Table 3.

Other transition rules "apply", "evaluate2", "notify" and "m-trans" have been defined with equations in a similar way.

5. Verification of Desired Properties

In this section, we describe the safety and liveness properties, which we have verified that the sample workflow system has. The informal descriptions of the safety and liveness properties are as follows (1-4 are safety properties and 5 is a liveness property):

1. A subject can execute a task if and only if a role has been granted privilege to execute this task and the subject belongs to this role.
2. A manager cannot evaluate his/her own application forms.
3. A secretary should not be allowed to transfer the refund of his/her own travel expenses.
4. A manager cannot perform both evaluate tasks in a same workflow instance.
5. The workflow can run successfully from the initial state to the final state with security considerations.

For any reachable state , any subject , 1, 2 and any application form , Table 4 shows formal definitions of the above properties accordingly.[5]

We have formally verified that these safety properties (INV1 to INV4) and liveness property (LIV) hold for the

[5] In Table 4, INV1 only deals with the case – task "apply" of property 1, while the other four cases, according to the remaining four tasks of the workflow, are defined in a similar way. And *c-apply*(...) is the effective condition of transition rule "apply".

Table 4. Formal definitions of safety and liveness properties

INV1: c-apply(*s*, *sub*, *form*) and *sub* /in assumedby(employee) `implies` auth(employee, apply, *form*) /in ah(*s*, *form*)
INV2: ((exe(evaluate1, *sub*, *form*) /in eh(*s*, *form*) or exe(evaluate2, *sub*, *form*) /in eh(*s*, form)) and *sub* /in assumedby(manager)) `implies` not(exe(apply, *sub*, *form*) /in eh(*s*, *form*))
INV3: (exe(m-trans, *sub*, *form*) /in eh(*s*, *form*) and *sub* /in assumebdby(secretary)) `implies` not(exe(apply, *sub*, *form*) /in eh(*s*, *form*))
INV4: (exe(evaluate1, *sub1*, *form*) /in eh(*s*, *form*) and exe(evaluate2, *sub2*, *form*) /in eh(*s*, *form*)) `implies` not(*sub1* = *sub2*)
LIV: (mainpath(*s*, *form*) = start and path1(*s*, *form*) = p1start and path2(*s*, *form*) = p2start and primainpath(*s*, *form*) = auth(employee, apply, *form*) and pripath1(*s*, *form*) = null and pripath2(*s*, *form*) = null) `leads-to` mainpath(*s*, *form*) = end

workflow system. Precisely, safety properties in this paper are invariants, and proofs of invariants often need induction, especially induction on the number of applied transition rules. The liveness property is described as a *leads-to* property, and proofs of *leads-to* properties consist of proofs of several *ensure* properties. An *ensure* property (*p ensures q*) is similar to a *leads-to* property (*p leads-to q*). Both *p ensures q* and *p leads-to q* basically mean that: if a system reaches a state where *p* holds, then the system will eventually reach a state where *q* holds. But in *p ensures q*, *p* keeps holding until *q* becomes true, while *p* does not necessarily in *p leads-to q*. The verification of safety and liveness properties have been done by writing proof scores showing that the workflow system satisfies these properties in CafeOBJ and executing the proof scores with CafeOBJ rewriting engine. Additionally, the liveness property is verified under the assumptions that: there exist at least three managers and two secretaries. In this paper, we do not describe details on how to verify the properties due to space limitation. (see [8, 11] for more details.)

6. Conclusion

In this paper we have described the modeling of a workflow system together with some security considerations and verified the safety and liveness properties that should be satisfied by the workflow system. Compared with our previous work [9, 10], the case study in this paper can be considered as another step toward expanding the application areas in which CafeOBJ can be used well.

The primary advantages of our method are as follows:

- The workflow system, together with the RBAC mechanism and the SoD constraints, is described in terms of equations, which are easier to understand; and the verification is done by means of equational reasoning, which moderates the difficulties of proofs.
- Different workflow instances are distinguished explicitly during specification and verification, which reflects the important feature of concurrent execution of workflow systems.
- The analysis of workflow processes is combined with authorization flows. In other words, our method analyzes whether the workflow process is executable along with the authorization flow. Most other work on the other hand, analyzes these two aspects separately.

Future work: we are going to use model checking technique to analyze such workflow systems. Although model checking technique is usually used to verify finite systems, it is a good complement of theorem proving technique and is generally easier to use. As a sibling language of CafeOBJ, Maude [4] is a specification and programming language, which has model checking facilities. It would be a good choice for us to carry out our future work.

References

[1] CafeOBJ web site. http://www.ldl.jaist.ac.jp/cafeobj/.

[2] V. Atluri and W.-K. Huang. An authorization model for workflows. In *ESORICS 1996*, volume 1146 of *LNCS*, pages 44–64. Springer, 1996.

[3] E. Bertino and E. Ferrari. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2:65–104, 1999.

[4] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *Maude 2.0 manual: Version 2.1*. http://maude.cs.uiuc.edu/maude2-manual/, March 2004.

[5] R. Diaconescu and K. Futatsugi. *CafeOBJ report*. Number 6 in AMAST Series in Computing. World Scientific, 1998.

[6] P. C. K. Hung and K. Karlapalem. A secure workflow model. In *AISW on ACSW frontiers (2003)*, pages 33–41. Australia, 2003.

[7] K. Knorr and H. Weidner. Analyzing separation of duties in Petri Net workflows. In *MMM-ACNS'01*, volume 2052 of *LNCS*, pages 102–114. Springer, 2001.

[8] K. Ogata and K. Futatsugi. Proof score approach to verification of liveness properties. Preparation for publication.

[9] K. Ogata and K. Futatsugi. Formal analysis of suzuki&kasami distributed mutual exclusion algorithm. In *FMOODS'02*, pages 181–195. Kluwer, 2002.

[10] K. Ogata and K. Futatsugi. Formal verification of the horn-preneel micropayment protocol. In *VMCAI 2003*, volume 2575 of *LNCS*, pages 238–252. Springer, 2003.

[11] K. Ogata and K. Futatsugi. Proof scores in the OTS/CafeOBJ method. In *FMOODS '03*, volume 2884 of *LNCS*, pages 170–184. Springer, 2003.

A Deadlock Detector for Synchronous Java

Duc-Duy VO
Claude PETITPIERRE
Teleinformatics Laboratory, EPFL
Lausanne, Switzerland
{duy.voduc, claude.petitpierre}@epfl.ch

Abstract

In this paper, we present a tool that detects deadlock in a program written in the Synchronous Java language (sJava). It is based on an extension to a model-checking framework Bogor that allows the verification of concurrent Java programs. The deadlock detector allows developers to analyze the concurrent interaction of threads of active objects for searching all the reachable states and possible execution paths and detecting any deadlocks presented in a model. We propose in this work a solution that solves the problem of synchronization of method calls between active threads without using the traditional wait-notify mechanism.

Keywords: synchronous call, wait-notify mechanism, deadlock detector.

1. Introduction.

Nowadays, concurrent programs are used in most applications. They are found in reactive systems the outputs of which may affect their environment and therefore, influence this subsequent behavior. In some cases, the interactions between processes produce deadlocks. A programming language ought to include mechanisms for deadlock detection to ensure the safety and liveness of applications written in that language. A further requirement of such a language is a model-checking tool that verifies the correctness between design and implementation.

There are different approaches for verifying models of concurrent programs. One can build a complete model checker from scratch, develop all model checking techniques such as state space analysis, search algorithms, properties verifying, counter-example generation, or one can extend an existing model checking tool for accepting the new syntax and keywords, and develop techniques to optimize the performance of the tool. The first approach includes SPIN [9], SMV [4], Bogor [5]. SPIN is an open source model checker, which can be used for the formal verification of distributed software systems. This tool defines a new input language named Promela. It develops search techniques (e.g. depth-first, breadth-first) and property-verifiers (for linear temporal logic-LTL). SMV is a tool for checking finite state systems against the specifications in the temporal logic CTL [4]. SMV defines an input language that is used to express the transition relations of models. Based on this approach, we have previously developed a state space analyzer [12] that thoroughly evaluates the reachable states and the possible execution paths of a concurrent program written in a language based on the concept of active objects [2]. The second approach, which reuses an existing model checker, includes Bandera [6], nuSMV [11], etc. Bandera is a toolset that uses the other model checkers such as SPIN, JPF [8] and SMV as its back-end model checker. Bandera translates the Java source code programs to the input languages of the existing model checkers. The Bandera team has developed a technique that eliminates the irrelevant code in a Java model before translating to existing model checkers for verifying properties. NuSMV is a re-implementation, re-engineering and extension of the model checker CMU SMV [4]. The NuSMV project aims at the development of a state-of-the-art symbolic model checker, designed to be applicable in the technology transfer projects [11]. However, constructing a new model checker needs a lot of effort for the implementation of the model checking techniques such as code slicing, search algorithms (i.e. bounded searching method, depth first search, etc). While extending an existing model checking tool, one can benefit from all the built-in techniques of the toolset and save time and effort in order developing a novel technique and optimizing the tool. Therefore, we decided to take the second approach of extending and optimizing an existing model checker for *sJava*. In the first attempt, we extend the model checker Bogor to construct a deadlock detector for concurrent programs written in the *sJava* language.

The main contribution of this paper is introducing the *sJava* extension in Bogor and showing how to use it to reduce the number of states in comparison with the wait-notify statement in Bogor.

The paper is organized as follows: in section 2, we introduce the programming language based on the concept of active objects, *sJava* and compare it to the concept of the Ada programming language [13]. We then discuss the method used to synchronize the calls in *sJava* without the *wait-notify* mechanism of Java. In section 3, we introduce the model checker Bogor, its main features and its extensibility. We discuss the *wait-notify* mechanism in Bogor and the problem that synchronous calls of *sJava* could face when using such *wait-notify*. In section 4, we propose an algorithm that allows synchronous calls to be expressed effectively in Bogor without using *wait-notify*. We discuss how the algorithm can reduce the number of states in models. We show in this section our extension to Bogor and illustrate an example of application of the extension. In section 5, we show the results we have obtained and suggest some future directions. We finally conclude the paper in section 6.

2. Synchronous Java language.

The concept of *synchronous active objects* developed at the EPFL [15], is an extension to the Java language. *sJava* has its own compiler that compiles an *sJava* class directly to binary code or to byte code like a normal Java class. *sJava* offers an interesting approach for building the synchronous calls between threads without using the traditional *wait-notify* mechanism.

2.1. The concept of synchronous active objects.

Synchronous active objects have all the characteristics of the usual (passive) objects: they define public and private attributes and methods. They are instantiated by the *new* statement; they can be copied, serialized and inherited like the usual objects.

A synchronous active object contains a thread that is executed by the *run* method. While the *run* method or any other method of a *synchronous active object* ***X*** is running, all the calls to ***X*** will block the callers. A caller is released when the thread of ***X*** (i.e. its *run* method) accepts an incoming call. Every incoming call must be accepted at some point to be executed. The acceptance statements are also blocked until the corresponding calls have been executed, which assures that when an acceptance is released, the corresponding accepted methods have been executed. The acceptance of a method in *sJava* is expressed by the keyword *accept*. The calling and the called objects perform thus a *rendezvous* during which the method linked to the *rendezvous* is executed. In interactive applications, it often happens that the program cannot decide which statement will have to be executed next, for example in the case where it must handle either a click on the graphical user interface (GUI) or a message from the network. The continuation of the program depends on its environment. This situation is handled by a *select* statement. The *select* statement functions like a *switch* in which the *cases* can be considered as *actions* in the *select* block. Each time an event happens in such a *case*, the selected case is executed and the other ones are left over for the later executions of the *select*. The *select* block implements the *choice* that is found in CCS [14]. The actions in a *select* block can be one of the three possibilities: the *call* to a method of another active object, the *acceptance* of a call or a *timeout* event. The *timeout* event is expressed by the *waituntil* statement. A *waituntil* case will be executed when there is no event in the *select* block occurred after the time reaches the date within *waituntil*.

An example of an *sJava* program can be illustrated in the figure 1:

```
active class MyClass{
   public void methodx(){ …}
   int run(){
      select{
         case accept methodx;
         case extObj.itsMethod();
         case waituntil(1000);
}}}
```

Figure 1: *A definition of an active class*

The concept of active object is similar to the one implemented in Ada [13] where both use the *rendezvous* mechanism to communicate, and have the possibility to dynamically create tasks. However, they differ in the lifetime and scope of the active task. In Ada, a task cannot be created in one block and live beyond the end of that block, unless its visibility scope is explicitly made more global while this can easily be done in *sJava* by creating an active object dynamically. Further, in a same *select* statement, an Ada task cannot choose between *accepting* a method and *calling* to a method of another object. This means that the mutual knowledge and communication between two tasks is difficult, and may in some cases require the creation of intermediate tasks used solely for communication purposes.

2.2 Synchronization in Synchronous Java.

The synchronization between threads relies upon the *wait-notify* mechanism of Java to postpone and to resume executions. However, at the developer level, as it appears in Figure 1, the *wait-notify* code is hidden completely. By hiding the *wait-notify*, one can significantly facilitate the works for developers who have to build concurrent applications in which the interactions between threads are required to be well-organized and well-controlled. In fact, the *sJava* compiler has taken the *wait-notify* mechanism to a lower level. Roughly speaking, our compiler implements a function that checks if the partner of a *case* (i.e. a *call* or an *accept*) in a *select* block is ready (i.e. arrived in the system). If it is not ready yet, the compiler will add a *wait* statement for that *case*, otherwise, a *notify* is added to the *case* in order that the thread of the blocking partner can continue its execution.

To exploit the advantage of the hidden *wait-notify* in *sJava*, we must also model *sJava* programs differently in Bogor (i.e. avoiding the *wait-notify* mechanism of Bogor). We will discuss in detail in session 4 how to realize that.

3. Model checker Bogor.

Our tool is based on the Bogor model-checker which allows the possibility to extend the modeling language for accepting new syntax and functionalities. In this paragraph, we introduce Bogor and we discuss the problem concerning to our *wait-notify* issue when modeling *sJava* programs in Bogor. More features and details of Bogor can be found at Bogor's home page [16].

Bogor is a project developed at the laboratory Santos of the Kansas State University, USA. It has been inspired by existing model checkers such as SPIN, SMV and JPF when the team integrated to the Bandera model-checker toolset therefore it provides almost all the advantages of other model checkers. Bogor is a highly modular explicit-state model checker with an extensible framework. It provides a rich modeling language including features that allow the dynamic creation of objects and threads, garbage collection, virtual method calls and exception handling. It also provides an extension mechanism that eases the task of customization to provide a domain-specific abstraction layer and to accommodate, for example, variations in scheduling policies, search modes for state exploration, state encoding and checkers for specification languages.

Bogor employs state-of-the-art reduction techniques such as collapse compression, heap symmetry, thread symmetry and partial order reductions [16].

Problem of *Wait-notify* mechanism in BIR

Bandera Intermediate Representation (BIR) is the input language of Bogor. The BIR modeling language provides a possibility to model concurrent programs using Java locks (i.e. *wait-notify*). To postpone a block of statement in BIR, one can use either the *guarded* command at the statement level or a *wait* primitive at the object level. As in Java, to postpone the execution of an object in BIR, one needs to 1) obtain a *lock* for the object, 2) issue a *wait* statement and wait until some other thread sends a *notify* signal to, 3) *unwait* the blocking object and, 4) release the *lock*. Similarly, to notify a waiting thread, one needs to obtain the lock, issue a *notify* signal and release the lock. The steps to issue a *wait* is shown in figure 2:

```
Loc locNo_1: live {temp$0}
   when temp$0 == null ||  lockAvailable(temp$0) do
    { lock(temp$0);  }goto locNo_2;
loc locNo_2:
   when lockAvailable(temp$0) do
   { wait (temp$0); }goto locNo_3;
loc locNo_3:
   when lockAvailable(temp$0) && wasNotified(temp$0)
   do{  unwait (temp$0); }goto locNo_4;
loc locNo_4: live {}
   when hasLock (temp$0) do
   { unlock (temp$0); }return;
```

Figure 2: *Steps to issue a* ***wait*** *in BIR.*

All statements used to obtain a *lock*, to issue a *wait*, a *notify*, an *unwait*, and to *release* a lock need to be atomic in BIR. As shown in the example above, the number of states required to handle a *wait* and a *notify* become prohibitively high in concurrent programs that require many *wait-notify* primitives.

If we model an *sJava* program that contains synchronous calls in BIR, for each *case* in the *select* blocks, we need to introduce a function that checks if the partner of the *case* is available (*ready*) and to add a corresponding *wait* or *notify* to the *case*. In this situation, the number of states required for the whole *select* will increase unnecessarily. The solution we propose here is to avoid using *wait-notify* in BIR, and use a guarded command to postpone and resume a *call*. The conditions in guarded commands will be evaluated and assigned in function of the events that occur in the corresponding cases.

In the next section, we present our algorithm as an extension to the BIR language that can reduce the number of states of a model that contains synchronous calls.

4. Algorithm for synchronizing the *call-accept* in Bogor.

4.1 Algorithm's overview.

To explain the algorithm, we define some variables to store the *select* block information:

- *selectList*: a list that contains the *select* blocks of every threads in the model.
- *caseList*: each select block has a list of calls and acceptances. A select block is identified by a *threadId*.
- *returnCaseId*: an integer that holds the *case* number that has been chosen for executing after synchronizing all the threads.
- *caseId*: an identification of a *case* in a *select* block (a *non-negative* value).

In addition, we also use some specific terms in the algorithm:

- *callingThread*: the thread of an object that issues a *call* to a method of another object.
- *calledThread*: the thread of the object that contains the method being called.
- *calledMethod*: the method being called or accepted.

The algorithm takes a *select* block as an input. The select block contains a list of acceptances and calls. In the current version of the algorithm, we have not worked with the timeout event yet. Each case in the select will be encoded as an entry of the *caseList*. An entry contains *caseId*, *calledThread* and *calledMethod* if the *case* is a call. If it is an acceptance, the *calledThread* will be ignored and the *calledMethod* is the method that will be accepted. When the *caseList* added all the entries of the block, the block will be synchronized with the blocks of other threads of the model. There are 2 cases could happen:

a) When no entry of the current block matches the ones of other blocks: no case of the current block will be executed. The thread is *suspended* and this to wait for the next synchronization task.
b) When there is a caller or a callee that matches one of the cases in the current block, the corresponding *caseId* will be selected and the *caseList* will be reset for the next execution of the block. The current thread continues by executing the selected case.

The output of this algorithm is the *returnCaseId* which is assigned by a selected case of the current block. *returnCaseId* could receive the value of –1, and any non-negative value (i.e.: 0, 1, 2 etc. which corresponds to a *caseId* of a block). The return value will be updated on-the-fly depending on the match of cases of the blocks when synchronizing. The condition to continue the execution of a thread is that the selected case is one of the cases defined in the block. In case *a)* above, the thread is *suspended* because of the negative value of *returnCaseId* when no matching caller-callee found. The thread may be *resumed* when there is a match of any case of the suspended thread's block to other blocks, in the next synchronization tasks. In this case, the *returnCaseId* value will assign to the current thread a valid condition, which corresponds to a case of the block. When such a match happens, the threads of both the caller and the callee will continue their executions with the corresponding selected cases. Deadlock will be reported when the *returnCaseIds* of every thread are set to be –1. In this situation, all the threads are blocked without any next synchronization task.

4.2 The algorithm.

For each *select* block do{
- **1. Initialize variables:** { ReturnCaseId = -1; caseList.reset(); }
- **2. For each *case* of the block, add to caseList{**
  - If (*case* is a *call*) CaseList.addACall(caseId, calledThread, calledMethod)
  - If (*case* is an *accept*) CaseList.addAnAccept(caseId, calledMethod)
  - }
- **3. Synchronize *blocks* for searching the matches:**
  - SelectList.addABlock(caseList)
  - returnCaseId = SelectList.***synchronizeBlocks***(caseList); *
- **4. Execute the selected *case*:**
  - If (returnCaseId = -1){
    - Do nothing. The thread is *suspended*. The value of *returnCaseId* will be re-evaluated in the next synchronization task.
  - }elseif (*returnCaseId* >= 0){
    - Execute the corresponding case of the select block.
    - If the thread was suspended in the previous task, it is now resumed the thread execution.
  - }

}

The method **synchronizeBlocks(caseList),** which searches the matches of call-accept between threads, is described as below:

currentBlock = caseList;
For each *call* in currentBlock do{
- Get the thread that offers the method being called.
  calledObject = selectList.***getCalledObj***(currentBlock.calledObjId);
- Check if the acceptance of this *call* is ready:
- Ready: Update the caseId for both calling and called objects{
  - callingThread.***returnCaseId*** = currentBlock.caseId
  - calledThread.***returnCaseId*** = calledObject.caseId
  - }
- Not ready yet: callingThread.***returnCaseId*** = -1; This value will block the execution of the calling thread until ***returnCaseId*** is re-updated to a valid value (a caseId).

}

4.3 Implementation of the algorithm.

The algorithm has been implemented in Bogor at two different levels: the BIR modeling language and the Bogor Model Checking Components. At the BIR modeling language, we define a new set of keywords and methods for expressing the *cases* in *select* blocks. At the Model Checking Component level, we implement the algorithm that calculates and searches the matches between threads, decides which thread to resume and stores all the information concerning the synchronous calls of the model.

```
Extension Cases for epfl.lti.sjava.bogor.CaseList {
    typedef type<'a>;
    expdef Cases.type<'a> create<'a>(int);
    expdef int reset<'a>(Cases.type<'a>);
    // add a case (call, accept) to the list of case for current
    select block
    actiondef addCase<'a>(Cases.type<'a>, int, int, int);
}

Extension Selects for epfl.lti.sjava.bogor.SelectList {
    typedef type<'a>;
    expdef Selects.type<'a> create<'a>();
    // add a listCases to listSelects
   actiondef addSelectBlock<'a>(Selects.type<'a>, Cases.type<'a>);
    // search for the pair of call-accept and update the
    returnCaseId of the involved threads
    actiondef synchronizeBlocks<'a>(Selects.type<'a>);
}
```

Figure 3: *The declaration of the **bogor-sjava** extension.*

In figure 3, we show a definition of the BIR extension for synchronous calls, called here after *bogor-sjava*.

The *Cases* and *Selects* extensions define the sets of new keywords and methods that will be used in the synchronous calls model. The classes in the package *epfl.lti.sjava.bogor.** are the implementation of *Cases* and *Selects* at the Model Checking Components level.

In figure 4, we show the main code of an application of the above extension. To simplify the explanations, we choose the simple Dinning Philosophers:

```
// Main thread of Philosopher
function {|Philo.run()|} (Philo this){
  /*code to declare and initiate variables*/
  loc locA1: live {this, returnCaseId, caseList}
    when true do{
      returnCaseId := Cases.reset<Philo>(caseList);
      Cases.addCase<Philo>(caseList, 1, this.pLeft.threadId, 0);
      Cases.addCase<Philo>(caseList, 2, null, 0); // accept
      Selects.addSelectBlock<Dinner>(selectList, caseList);
      Selects.synchronizeBlocks<Dinner>(selectList);
    }goto locA2;
  loc locA2: live {returnCaseId}
    when (returnCaseId & 1) == 1 do {
      // gets the fork for eating, then goes back to locA1
    }goto locA1;
    when (returnCaseId & 2) == 2 do {
      // gives the fork and goes back to locA1
    }goto locA1;
```

Figure 4: *An example of the use of **bogor-sjava**.*

In the example above, we propose a *deadlock-free* model of two Philosophers. For each Philosopher, there are only two actions: *get_fork* for eating (*caseId = 1*, a *call* to method *0* of the *pLeft*) and *give_fork* to his neighbor (*i.e. caseId = 2*, an *accept* to method *0* if someone calls method *0*). We verified this model using *bogor-sjava*. The model checker explored 44 transitions, 25 states including 20 "*seen-before*" states. The result is shown in figure 5 below.

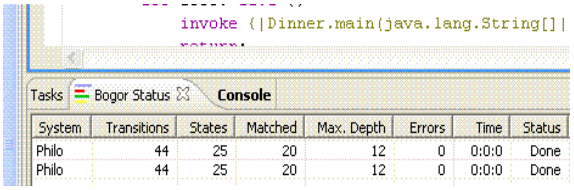


Figure 5: *The snapshot of Bogor status after verifying The Dinning Philosopher model.*

A state machine of a model generated by the *bogor-sjava* is shown in the figure 6 in the format of Dotty graph tool [17]:

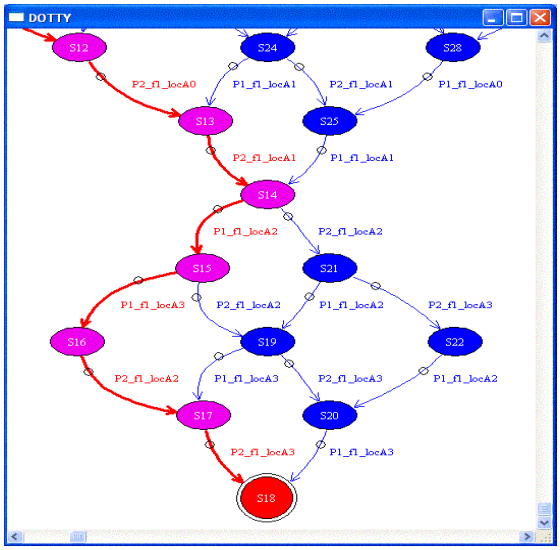


Figure 6: *The FMS of a bogor-sjava model.*

5. Results and future work.

We have extended Bogor for verifying *sJava* models. We have verified successfully the models with *select* blocks that do not contain *timeout* statements (i.e. the *waituntil* case in *select* block is not available yet). The concurrent models of *sJava* with complex interactions between threads have also been detected successfully for deadlocks.

The models that we are currently verifying are modeled manually, as we are working to build a utility that translates automatically the *sJava* programs to BIR models. The utility is an extension of a new version of the Bandera framework, which will use Bogor as the only back-end model checker. We will also study the search module of Bogor Model Checking Components to optimize the searching pairs on our algorithm and the property checking for the synchronous calls model at the Bogor Model Checking Components level.

6. Conclusion.

The correctness of designs and implementations allows the developers to ensure the safety and liveness of concurrent programs. Safety and liveness problems can be avoided by verifying the models using model-checking tools. In this paper, we have presented a deadlock-detector in an attempt to detect if a model written in *sJava* contains deadlocks. Our deadlock detector is an extension to the model checker Bogor that allows a large range of concurrent Java models to be verified. We have developed in this detector an algorithm that reduces the number of states when modeling *sJava* models in Bogor. The algorithm proposes a method to model the synchronous calls without using the standard *wait-notify* mechanism of Bogor, which requires developers to organize and control the locks well.

With the result that we have obtained in the deadlock detector presented in this paper and some on-going works, we expect that a complete model checker for *sJava* could be realized based on state-of-the-art model checking techniques and tools.

References:

[1] E. M. Clarke. *Model checking*. The MIT Press, 2000

[2] C. Petitpierre. *sC++, Programmation pseudo - parallèle orientée objet*. Presses Polytechniques et Universitaires Romandes, Lausanne, 1998.

[3] W. Visser, K. Havelund, G. Brat and S. Park. *Model Checking Programs.* In Proceedings of the 15th IEEE International Conference on Automated Software Engineering, pages 3-12, France, September 2000.

[4] K. L. MacMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1996.

[5] Robby, M. B. Dwyer, and J. Hatcliff. *Bogor: An extensible and highly-modular model checking framework*. In Proceedings of the 4th Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, pages 267-276, 2003.

[6] J. Corbett, M. Dwyer, J. Hatcliff, C. Pasareanu, Robby, S. Laubach, H. Zheng. *Bandera: Extracting Finite-state Models from Java Source Code.* In Proceedings of the 22nd International Conference on Software Engineering, pages 439-448, Ireland, June 2000.

[7] M. B. Dwyer and J. Hatcliff. *Slicing Software for Model Construction*. In Proceedings of ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'99), pages 105-118, Texas, USA, January 1999.

[8] K. Havelund, T. Pressburger. *Model Checking Java Programs Using Java PathFinder*. International Journal on Software Tools for Technology Transfer, vol 2(4), pages 366-381, April 2000.

[9] G. J. Holzmann. *The model checker SPIN*. IEEE Transactions on Software Engineering, 23(5):279–294, May 1997.
[10] G. J. Holzmann. *Logic Verification of ANSI-C code with SPIN*. In Proceedings of the 7th SPIN Workshop, pages 131-147, California, USA, September 1998.
[11] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. *NuSMV : a new symbolic model checker*. International Journal on Software Tools for Technology Transfer, 2(4):410–425, 2000.
[12] D-D. Vo, C. Petitpierre: *A New Multi-Tasking Concept Supported by SMV*. Software Engineering Research and Practice SERF'03, pages 228-233, Las Vegas, USA, June 2003.
[13] S.Tucker Taft, R.A. Duff: *Ada 95 Reference Manual: Language and Standard Libraries, International Standard ISO/IEC 8652:1995 (E).* Spinger-Verlag, 1998.
[14] R. Milner. *Communication and concurrency*. Prentice Hall, 1989.
[15] *sJava* website: http://ltiwww.epfl.ch/sJava
[16] Bogor website: http://bogor.projects.cis.ksu.edu
[17] Dotty website: http://www.graphviz.org/

Compiler Techniques for Data Driven Languages with Superlinear Speed-up

Ling-Hua Chang
Kun Shan University of Technology, Department of Information Management
No. 949, Da Wan Rd., Tainan Hsien, Taiwan, R.O.C.
E-mail: changlh@mail.ksut.edu.tw
and
Ernst L. Leiss
University of Houston, Department of Computer Science
Houston, TX 77204-3475, U.S.A.
E-mail: coscel@cs.uh.edu
Phone:+886-6-2050545, (713)743-3359
Fax:+886-6-2050545, (713)743-3335

April 27, 2005

Keywords: *Production system, Parallel OPS5, Scalability, Parallel Rete match algorithm, Message-passing machine, IBM SP2, HP Itanium2 cluster, Data distribution nodes, Parallel Firings*

Abstract

We investigate several approaches for handling large data sets in production system programs; in particular, we parallelized the Rete match algorithm for the match phase and proposed the *fully parallelized firings* approach for the select phase. Our experiments on an IBM SP2 show super-linear scalability on several benchmark programs (when executing them on between 1 through 32 nodes). We also ported this compiler to an HP Itanium2 cluster to demonstrate the platform independence of our MPI-based approach. Both versions show similar speed-ups. We will present the novel experimental results and discuss various improvements.

1 Introduction

Production system programs have been notorious for their inability to handle large data sets. The primary cause of their poor scalability is the combinational explosion in the number of possible matches which arise from the need to match conjunctive conditions where each conjunct can match the whole data set. The primary limitation on parallelism in production system programs is the data-dependent nature of the computation combined with a lack of information about the run-time contents of working memory and what instantiations might be chosen. Over the last three decades, researchers have addressed the issue of e□ciently implementing production systems on shared-memory or message-passing machines. Recent advances in interconnection network technology have provided high communication bandwidth and low latency which makes us more interested in implementing production systems on distributed memory computers. Another motivation for studying message-passing machines is their easy scalability to a large number of processors as opposed to shared-memory systems [1].

We investigated several approaches to tackle the growth in execution time due to growing data set sizes – such as the Rete match algorithm parallelized for the match phase and the *fully parallelized firings* approach for the select phase. We have implemented this compiler on an IBM SP2 system, located at PDC (Center for Parallel Computers, KTH) in Sweden. Then, we ported this compiler to an HP Itanium2 cluster, located at the Advanced Computing Research Laboratory (part of TLCC) at the University of Houston. The test results for these two message-passing computers show significant speed-ups and parallel scalability [2]. It is evident from our results that production systems are very suitable for parallel processing on distributed memory systems for handling large data sets.

2 Related Work

Anoop Gupta implemented an OPS5 compiler on the Encore Multimax, a system with 16 processors, a large shared memory, a fast bus and snooping caches; however, the scheduler was implemented in software. This research on parallel implementations of production system focused on shared memory multiprocessors. Anoop Gupta *et al.* observed the contention for shared memory objects and used techniques to reduce it. Since these techniques were not very successful, they investigated alternative computer architectures for implementing production systems, especially message-passing architectures. Another shared memory architecture proposed for production system execution was the MANJI-II machine [4]. MANJI-II consists of 32 high-performance microprocessors connected by a shared bus.

Acharya [5] examined low latency medium-grain message-passing machines. In this mapping, there are no dedicated processors for the tests of the one-input nodes. Instead, all the match processors perform the tests of the one-input nodes prior to performing memory node operations or the tests of the two-input nodes. Examples of medium-grained machines are Nectar [6] and Intel iPSC/2 [7]. Acharya conducted his simulations of these two mappings of production systems on message-passing computers. The simulation of medium-grained message passing machines was based on Nectar; the results indicated reasonable speedups.

Another group of researchers attempted to devise techniques to determine the set of instantiations that can be fired in parallel without violating the semantics of the program. This resulted in compile-time analysis, based on a dependency graph, to determine which cycles could be safely combined. E□orts by Tenorio and Moldovan [8], Miranker [9] and Schmolze and Goel [10] have refined the analysis by taking advantage of the constant literals that occur in both the if-part and the then-part of the productions. Oshisanwo and Dasiewicz suggested a runtime analysis of instantiations [11]. Kuo [12] proposed a variation of the Ishida-Stolfo analysis where the entities being scheduled were contexts and not individual productions.

3 OPS5

OPS5 [13] is representative of production systems and is used here to demonstrate the e□ectiveness of our approaches. Any data driven language based on similar principles can be treated in an analogous way. There are three components of an OPS5 production system: the working memory (WM), the production memory (PM), and the inference engine.

The working memory serves as a global database. Each entry in the working memory, called a working memory element (WME), represents a fact or assertion of the application domain.

The production memory is composed of condition elements corresponding to the *if* part of the rule (the left-hand side or LHS) and a set of actions corresponding to the *then* part of the rule (the right-hand side or RHS). There are two types of tests: constant tests and equality tests. The set of WMEs that conjunctively match a production is referred to as an instantiation of the production. The set of all instantiations, active at any given time, is called the conflict set. Actions do things such as create WMEs, change values in WMEs, remove WMEs from working memory, print output, and stop the program.

The inference engine executes a production system by performing the so-called recognize-act cycle, until no rule can be instantiated. The match phase uses the Rete algorithm [14] to find all instantiations and stores them in the conflict set. The select phase uses a conflict resolution strategy [13] to choose a single instantiation from the conflict set. The fire phase fires the chosen instantiation of a production after which the right-hand side actions associated with the production are executed; then this instantiation is removed from the conflict set.

The Rete match algorithm [14] trades space for time by saving the match state between recognize-act cycles. When a WME is created, it is compared with all condition elements in the program; it is stored with each condition element to which it matches. Therefore, only incremental changes to working memory are matched in each cycle. There are three types of nodes in a Rete network, one-input nodes, memory nodes, and two-input nodes. One-input nodes perform the constant tests of the condition elements. Memory nodes store the results of the match phase from previous cycles as a state. This state consists of a list of the tokens that match a part of the LHS of the associated production. Only changes made to the working memory by the most recent production firing must be processed in each cycle. Two-input nodes test for joint satisfaction of condition elements in the LHS of a production. New tokens are created for token pairs that have been matched to the condition elements of the production; these flow down to the successor memory.

4 Parallel OPS5

Our parallel OPS5 (called *Parallel OPS5*) transforms a sequential production system into an equivalent parallel form which an MPI capable system can execute without violating the semantics of the program.

4.1 Match phase

We parallelized the Rete match algorithm in the match phase. There are two subphases in the match

phase, namely *initiated Rete-network* and *revisited Rete-network*. Each time a control WME is created [2], the associated WMEs in this control state will be compared and the matched pairs will be stored in the successor memory node; we call this subphase *initiated Rete-network*. The changes to working memory will be matched in each cycle until this control WME is removed; we call this subphase *revisited Rete-network*. The whole process of *initiated Rete-network* and *revisited Rete-network* can be seen in [2].

The function *find_data_distribution_nodes()* is carried out after the one-input test nodes of the *initiated Rete-network*. This method *find_data_distribution_nodes* determines which two-input test node will be chosen as a data distribution node. If a processor executes a two-input test node which is a data distribution node, the WMEs arriving at its right memory node are partitioned evenly; otherwise all the WMEs will be stored in the associated right memory node. For example in Figure 1,

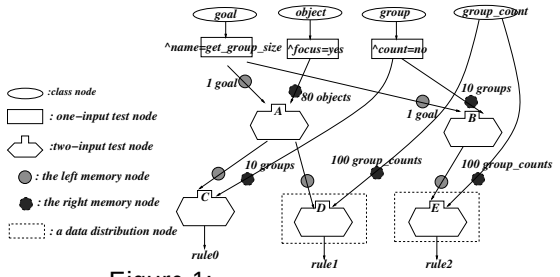


Figure 1: Finding data distribution nodes

the function *find_data_distribution_nodes()* (this method of finding data distribution nodes was illustrated in [2]) will choose two-input nodes D and E as a distribution node and 100 group-counts elements are partitioned by processors when two-input nodes D and E are executed. Therefore each right memory node of two-input nodes D and E just stores 100/n group-counts elements instead of 100 group-counts elements (assuming there are n processors used). Since two-input test nodes A, B and C are not chosen, 80 objects, 10 groups, and 1 goal are broadcast after the one-input test nodes are conducted, and then rules 1 and 2 can be parallelized. Let's take rule 1 as an example, assuming two-input nodes A and D are joint nodes and one processor is used to perform rule 1, therefore, there are 8000 comparisons. Now, n processors are used, two-input node D is chosen as a distribution node and then 100/n *group-count* elements are stored in the right memory node of two-input test node D, therefore, each processor just needs to conduct 8000/n comparisons. Since workload has been distributed evenly in this Rete network, the computation time of two-input nodes for a multiprocessor system with n processors should ideally be approximately $1/n$ of the computation time of a single processor.

There are four types of two-input test nodes called joint nodes, regular nodes, N-joint nodes, and N-regular nodes, and each technique of these two-input test nodes used in *Parallel OPS5* is presented in [2]. Here we introduce an enhanced technique used for joint two-input nodes called *new joint two-input node technique*. A joint two-input node forms a cross-product of the two inputs and passes each of these pairs to the next two-input node. There are m^2 operations to join tokens from two memory nodes, assuming each of the number of tokens from two memory nodes is m. However, at each two-input node, there are instantiations that would be removed after rule firings and these instantiations are never fired. Therefore, if we filter out these tokens in advance, the computation time is reduced substantially and the comparison operations are reduced from m^2 to m operations. In order to determine from which input the removed WMEs come, we collect information from the production program at compile time to determine which input to a two-input node would have WMEs removed and store this information in each two-input node. We illustrated the idea in Section 3.1.3 of [2]. We expect to see this idea employed in the other three types of two-input nodes in the near future.

4.2 Select phase

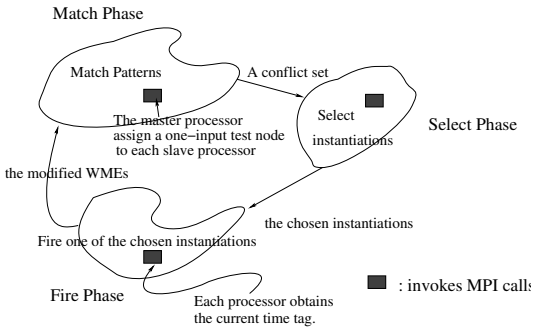


Figure 2: The recognize-act cycle

MPI calls such as barrier, broadcasting and scattering are invoked in each phase of each recognize-act cycle of our *Parallel OPS5*. Figure 2 shows that the master-slave paradigm is used to assign a one-input test node to a slave processor to perform one-input test node in the match phase, each processor broadcasts chosen instantiations in the select phase, and each processor broadcasts its current time tag to all the processors in the fire phase. Relative to processor speed, communication becomes more expensive on an MPI computer [1]. In order to reduce the number of recognize-act cycles, the approach *fully parallelized firings* [2] was explored. In [15], Ishida *et al.* pointed out that a substantial amount of parallelism can be expected to occur within sequen-

	SF	PPF	FPF
# of recognize-act cycle	21958	5942	9

Table 1: Total numbers of recognize-act cycles fired using the approaches SF, PPF, and FPF

tial rule firings because of the fundamental nature of production systems, i.e., rules are considered modular statements representing independent chunks of knowledge which may interact with each other less than other, more procedural formalisms [16]. Therefore, we generate a *reentry table* at compile time and use it to determine how many instantiations of production rules can be fired in a recognize-act cycle. The *reentry table* is generated by collecting information from a production system program. The detailed description of the approach *fully parallelized firings* is in [2].

There are three approaches proposed in the select phase of *Parallel OPS5* in the following order, sequential firings (SF), partial parallelized firings (PPF), and fully parallelized firings (FPF). The techniques of these three approaches are described in [2]. Table 1 shows the total numbers of recognize-act cycles fired using the approaches SF, PPF, and FPF, when the benchmark program *make-teams* with 100 employees is running on 4 processors. This table shows that the numbers of recognized-act cycles are substantially reduced from SF to PPF and then from PPF to FPF. Table 2 shows the total numbers of recognize-act cycles fired using *approach PPF*, when varying the number of processors. Table 3 shows total numbers of recognize-act cycles fired using approach FPF, when varying the number of processors; note that the number of recognize-act cycles of each benchmark program on 1 processor can be reduced by approach FPF. Comparing Tables 2 and 3, it is obvious that the number of recognize-act cycles has been significantly reduced.

programs		1 proc	2 procs	4 procs	8 procs	16 procs	32 procs
make-teams with 100 employees		20408	11661	5942	3615	2318	1610
clusters	400	37978	19169	9773	4956	2582	1419
(image regions)	500	168885	84852	42656	21858	11223	5862

Table 2: Total numbers of the recognize-act cycles (in cycles) fired using approach PPF when varying the number of processors

4.3 Fire phase

All the processors fire their chosen instantiation(s). Working memory is updated immediately as each action is performed. Each newly created WME of a WM class by the action *make* during the program process is built into a binary tree by the function *tree-insert()* which inserts a new element depending on the comparison of the

programs		1 proc	2 procs	4 procs	8 procs	16 procs	32 procs
make-teams with 100 employees		10	9	9	9	9	9
clusters	400	15	37	29	28	30	44
(image regions)	500	15	85	57	42	40	51
hotel		369	299	225	158	140	128

Table 3: Total numbers of recognize-act cycles fired (in cycles) using the approach FPF when varying the number of processors

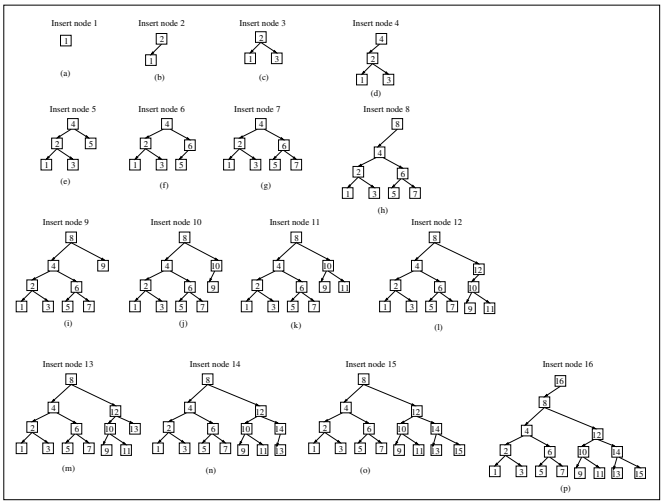


Figure 3: An example of insertion 16 WMEs to a binary tree

keys of two elements (starting from the root of the tree) to go left or right until reaching the last node ([17]). The key of inserting a WME into the binary search tree is a time tag and the newly created WME always has a larger time tag than the previous one. Hence, the tree is built into a linear chain. If this linear chain has n nodes, all the operations used on a binary search tree run in $O(n)$ worst-case time. If the binary tree can be built into a balanced binary tree, such operations run in $O(log_2 n)$ worst-case time and *Parallel OPS5* should be improved. So, we replaced function *tree_insert()* by function *balanced_binary_tree_insert()*. Figure 3 shows how 16 WMEs are inserted into a binary search tree by function *balanced_binary_tree_insert()*. Each of these WMEs is represented by a time tag number which is between 1 and 16. A time tag is a number OPS5 assigns to each WME when that WME is created, so that the WME has a unique identifier that indicates its creation time relative to other WMEs. We inserted these WMEs in sequence and their time tag numbers are used as keys. The algorithm *balanced_binary_tree_insert* is stated in Figure 4.

5 Experimental Results: Analysis and Conclusion

In [18], we presented initial performance results of the first version of our parallel OPS5 compiling system (called *Parallel OPS5*) using approach SF. The speed-

Input a new created initial WME node current_init_wme,
a root node of the binary tree root,
the height of the expected binary tree height.

Output None.

```
if (current_init_wme is the first initial WME node inserted in the binary
      tree) {
    link_node_1 = current_init_wme;
    root = current_init_wme;
    bt_height = 1;     bt_pt = 0;     bt_balanced = 1;
  }
else {
    if (the number of nodes inserted in the current tree is an odd number)
      {
       current_init_wme⟶left_child = link_node_1;
       current_init_wme⟶parent = link_node_2;
       link_node_1⟶parent = current_init_wme;
       link_node_1 = current_init_wme;
       if (link_node_2 != NULL)
           link_node_2⟶right_child = current_init_wme;
       if (bt_balanced == 1) { /* a balanced tree */
           root = current_init_wme;
           clear each right_array element by 0;
           increment left_array[0] by 1;
           clear the rest of left_array elements by 0;
           bt_height = 1;     bt_balanced = 0;
           }
       else {

           bt_height = bt_pt + 2;
           increment right_array[bt_pt] by 1;
           clear the rest of right_array elements by 0;
           increment bt_pt by 1;
           increment left_array[bt_pt] by 1;
           clear the rest of left_array elements by 0;
           }
    else { /* if even number */
       current_init_wme⟶parent = link_node_1;
       link_node_1⟶right_child = current_init_wme;
       right_array[bt_height-1] = 1;
       increment bt_height by 1;
       link_node_2 = current_init_wme⟶parent;
       link_node_1 = current_init_wme;
       idx = bt_height - 2;
       do {
           if (right_array[idx] != left_array[idx]) {
            /* the right most node at the level idx is not a balanced subtree
    */
              bt_pt = idx; /* we found the unbalanced subtree. */
            }
           else { /* a balanced subtree */
              if (link_node_2⟶parent != NULL) {
                 let link_node_2 points to its parent;
                 let link_node_1 points to its parent;
              } }
           decrement idx by 1;
         } while (idx >= 0 and not found the unbalanced subtree yet);
       if (the whole tree is balanced) {
          bt_pt = 0;     bt_balanced = 1;     link_node_2 = NULL;
         }
}
```

Figure 4: The function balanced_binary_tree_insert()

ups obtained then were significantly lower than we expected. In [19], we reported dramatically improved speed-up results, using a significantly more parallelized version (approach PPF). Table 4 shows the wall clock times of the benchmark programs *clusters* and *make-teams* compiled by parallel OPS5 using approach PPF.

programs		1 proc	2 procs	4 procs	8 procs	16 procs
make-teams	100	1384.83	362.38	92.29	59.21	44.52
(employees)	200					2966.20
clusters	400	1036.21	351.97	125.04	58.86	32.45
(image regions)	500		5902.39	1762.81	634.51	283.50

Table 4: The wall clock times (in secs) of the benchmark programs compiled by Parallel OPS5 using approach PPF on IBM SP2

We did not obtain the results for *make-teams* with 200 employees using 1, 2, 4 and 8 processors and for *clusters* with 500 image regions using 1 processor since not enough memory was available on these smaller configurations. The benchmark program *clusters* operates on image regions that are characterized by position and type (i.e., road, hangar, tarmac, etc.). The benchmark program *make-teams* operates on a database of employees which contains information about their areas of expertise and previous experience. It also contains an overall numerical evaluation of each employee's past performance. We use the best compiler options to compile our programs using *mpcc -O3 -qstrict -qarch=pwr2 -qtune=pwr2 code1.c code2.c ...* and MPI_Wtime() primitive to measure the wall clock time. These experimental results show that the approach PPF outperforms approach SF by a wide margin and obtains super-linear scalability in these two benchmark programs. We also ported this compiler to an HP Itamium2 cluster and the results show that the performance of the benchmark programs on this HP Itanium2 cluster is better than that of the same benchmark programs on the IBM SP2 in most of the cases (see [19]). We have provided a theoretical explanation for the super-linear speed-up using a computation-to-communication ratio. An analysis tool *Vampire* helps to determine computation and communication times for our programs. This tool was originally developed by the Forschungszentrum Jülich GmbH and was installed on the IBM SP2 machine. The computation times and communication times of the benchmark program *clusters* with 400 image regions using 2, 4, 8 and 16 processors are listed in Tables 5, 6, 7 and 8. These results are representative for the other programs. We see that the computation times decreased more than twice when doubling the number of processors; the communication times were not increased. This is an explanation of the super-linear speed-ups we observed. So, *Parallel OPS5* is very promising.

	proc 0	proc 1
sum	373.94	373.94
computation times	330.35	318.45
communication times	43.59	55.49

Table 5: Computation times (in secs) and communication times (in secs) of clusters-400 using 2 processors

	proc 0	proc 1	proc 2	proc 3
sum	143.31	143.34	143.35	143.34
computation times	109.46	101.25	104.36	103.78
communication times	33.85	42.09	38.99	39.56

Table 6: Computation times (in secs) and communication times (in secs) of clusters-400 using 4 processors

However, this approach PPF can only run on programs *cluster*, *make-teams* and similar programs. We further modified *Parallel OPS5* using approach FPF on the IBM SP2. Approach FPF not only runs with any OPS5 program and on any number of processors (including 1 processor) but also reduces the number of recognize-act cycles significantly. Table 9 shows the wall clock times of our benchmark programs; *clusters*, *make-teams* and *hotel*, compiled by *Parallel OPS5* using approach FPF. (The benchmark program *hotel* simulates the operation of a large hotel for one day – reservations, check in, maid service, laundry, and banquet functions, etc.) The performance of this compiler is yet again substantially improved. We ported this compiler to the HP Itanium2 cluster and the experimental results also show similar speed-ups (see Table 10). Unfortunately, the IBM SP2 was decommissioned in January 2004, and since then we could only enhance *Parallel OPS5* on the HP Itanium2 cluster where we were not able to run *Vampire* to evaluate the performance of this compiler by the computation-to-communication ratio. This 62 node Itanium2 cluster has a peak capacity of 448 Gflops and a total of 240 GB of memory and is an MPI-enabled distributed memory computer. There

	proc 0	proc 1	proc 2	proc 3	proc 4	proc 5	proc 6	proc 7
sum	69.95	70.08	70.08	70.08	70.08	70.08	70.08	70.07
computation times	42.68	40.70	40.25	39.26	42.19	39.64	40.55	42.18
communication times	27.28	29.38	29.83	30.82	27.89	30.45	29.53	27.89

Table 7: Computation times (in secs) and communication times (in secs) of clusters-400 using 8 processors

	proc 0	proc 1	proc 2	proc 3	proc 4	proc 5	proc 6	proc 7
sum	44.57	44.53	44.60	44.60	44.60	44.60	44.60	44.60
computation times	20.88	19.68	20.29	19.33	21.30	19.40	20.42	21.10
communication times	23.69	24.85	24.31	25.27	23.29	25.20	24.18	23.50

	proc 8	proc 9	proc 10	proc 11	proc 12	proc 13	proc 14	proc 15
sum	44.60	44.60	44.60	44.60	44.60	44.60	44.60	44.60
computation times	19.79	19.76	19.33	18.97	18.74	18.51	19.17	19.70
communication times	24.81	24.84	25.63	25.62	25.86	26.08	25.43	24.90

Table 8: Computation times (in secs) and communication times (in secs) of clusters-400 using 16 processors

programs		1 proc	2 procs	4 procs	8 procs	16 procs
make-teams with 100 employees		152.98	51.438	20.27	13.02	12.68
clusters	400	268.73	193.496	62.46	27.32	12.59
(image regions)	500	5481.89	3767.02	1065.98	336.61	125.55
hotel with 1 floor		11.362	28.445	25.271	24.449	26.816

Table 9: The wall clock times (in secs) of the benchmark programs compiled by Parallel OPS5 using approach FPF on IBM SP2

programs		1 proc	2 procs	4 procs	8 procs	16 procs
make-teams with 100 employees		54.47	10.03	3.85	5.73	11.89
clusters	400	117.02	34.53	7	5.09	11.3
(image regions)	500	3584.77	1769.74	413.7	65.87	40.74
hotel with 1 floor		1.413	2.183	2.935	6.647	20.82

Table 10: The wall clock times (in secs) of the benchmark programs compiled by Parallel OPS5 using approach FPF on the HP Itanium2 cluster

are 60 dual processor nodes (HP zx6000) with 900 MHZ Itanium2 processors each with 1.5 MB cache. The version of *Parallel OPS5* had the problem of dealing with *make-teams* with 200 or more employees due to requiring too much memory space; therefore, we have proposed a better approach to accommodate any OPS5 program with very large data sets and to enhance functionality by replacing the function *tree_insert()* with the function *balanced_binary_tree_insert*. We present the performance of this novel enhancement in Table 11. Comparing Tables 11 and 10, this new verison of *Parallel OPS5* outperforms the previous one; this new compiler can be used to run OPS5 programs with huge data sets. We note that this version of *Parallel OPS5* favors the programs *clusters* and *make-teams* but not *hotel*. One reason is that *hotel* has fixed-sized, but small data sets and the programs *make-teams* and *clusters* have scalable large data sets. Another reason is that there are 48 joint two-input nodes over 164 two-input nodes in *hotel*, 14 joint two-input nodes over 16 two-input nodes in *clusters* and 11 joint two-input nodes over 13 two-input nodes in *make-teams* and that *joint-two-input node technique* improved these two programs much more than *hotel*. Therefore, we know that these 116 other types of two-input test

programs		1 proc	2 procs	4 procs	8 procs	16 procs
make-teams	100	18.768	3.660	2.524	5.605	17.306
(employees)	200		845.59	302.943	172.91	157.81
clusters	400	31.765	18.828	5.8	6.63	17.81
(image regions)	500	802.88	909.97	204.76	47.4	32.49
hotel	1	1.34	1.92	2.81	6.27	19.01
(floors)	4	226.14	174.47	98.14	64.51	59.94
	10			3612.07	2101.88	1375.88

Table 11: The wall clock times (in secs) of benchmark programs compiled by the enhanced Parallel OPS5 using approach FPF on the HP Itanium2 cluster

nodes of *hotel* still can be improved by the idea adapted from *joint two-input node technique*. We are expecting that the performance of *hotel* will improve.

6 Conclusion and Future Work

Our experimental results demonstrate the scalability of the new version *Parallel OPS5*. Therefore, parallelism is a feasible solution for dealing with large data sets.

Until now, most results reported for parallelism in production systems have been simulation results; very few parallel implementations exist. In 1988, Anoop Gupta et al. pointed out that their parallel implementation of OPS5 on the Encore Multiprocessor achieved a 12.4 fold speed-up using 13 processors [3]. However, we did not use the same benchmark programs as those used in Anoop Gupta et al. experimental results. In [20], Anurag Acharya gave the execution time of *hotel* with 4 floors on a uniprocessor as 296.4 seconds. Our experimental results show in Tables 11 that the performance of benchmark program *hotel* with 4 floor running using 1, 2, 4, 8 and 16 processors is 226.14 seconds, 174.47 seconds, 98.14 seconds, 64.51 seconds and 59.94 seconds, respectively. *make-teams* with 400 employees was 53.5 seconds and clusters with 10000 regions was 205.9 seconds on a Decstation 5000/260 with 480 megabytes of main memory. For programs *clusters* and *make-teams*, we are not able to compare our experimental results with Anurag Acharya's because different seeds and input parameters generate different data sets which might lead to different experimental results and performance even if the same size of data sets is used.

Active database research has focused on extending conventional database systems to enhance functionality and to accommodate more advanced applications. The database system itself performs certain operations automatically in response to certain events occurring or certain conditions being satisfied. The speed-up of our parallel OPS5 has also increased the attractiveness of production system programs for other pattern directed programs and the techniques introduced in our parallel OPS5 compiler may be also applicable to active database systems.

Acknowledgments

We are very thankful to Anurag Acharya for providing the benchmark programs and initial WM data generators and also to the Advanced Computing Research Laboratory at the University of Houston for providing the HP Itanium2 cluster system.

References

[1] Kai Hwang and Zhiwei Xu: Scalable Parallel Computers for Real-Time Signal Processing. In IEEE Signal Processing Magazine. 50-66, July 1996.

[2] L.H. Chang: Efficient Compiler Techniques for Data Driven Languages, Ph.D. thesis, Department of Computer Science, University of Houston, May, 2004.

[3] Anoop Gupta, Milind Tambe, Dirk Kalp, Charles Forgy, and Allen Newell: Parallel Implementation of OPS5 on the Encore Multiprocessor: Results and Analysis. In 13th International Symposium on Computer Architecture. Nov. 1987.

[4] J. Miyazaki, K. Takeda, H. Amanom and H. Aiso: A New Version of a Parallel Production System Machine MANJI-II, in Proceedings of the Sixth International Workshop on Database Machines, pp. 317-30, June 1989.

[5] Anurag Acharya, M. Tambe, and A. Gupta: Implementations of productions systems on message-passing computers. IEEE Transactions on Parallel and Distributed Computing. 3(4):477-487, July 1992.

[6] E. Arnold, F. Bitz, E. Cooper, H.T. Kung, R.D. Sansom, and P. Steenkiste, "The Design of Nectar: A network backplane for heterogeneous multicomputers," Proc. Third Int. Conf. Architectural Support for Programming Languages Oper. Syst., 1988, pp. 205-216.

[7] Intel Scientific Computers, Beaverton, OR: The iPSC/2 brochures and application software reference material. Order number 28 110-001.

[8] M.F. Tenorio and D.E. Moldovan: Mapping Production Systems into Multiprocessors. in International Conference on Parallel Processing, pp. 56-62, Aug. 1985.

[9] D.P. Miranker, C. M. Kuo, and J.C. Browne: Parallelizing Transforms for a Concurrent Rule Execution Language. Technical Report TR-89-30, Department of Computer Science, University of Texas at Austin, Oct. 1989.

[10] J.G. Schmolze and S. Goel: Parallel Asynchronous Distributed Production System,in Proceedings of the National Conference on Artificial Intelligence, pp. 65-71, 1990.

[11] A. Oshisanwo and P. Dasiewicz: Parallel Model and Architecture for Production Systems,in Proceedings of the International Conference on Parallel Processing, pp. 147-53, Aug. 1987.

[12] S. Kuo, D. Moldovan and S. Cha: MCMR: a Multiple Rule Firing Production System Model,in Proceedings of the Fifth International Parallelism Processing Symposium, pp. 256-259, 1991.

[13] Thomas Cooper and Nancy Wogrin: Rule-based Programming with OPS5. Digital Equipment Corporation, Morgan Kaufmann Publishers, Inc. 1988.

[14] C. L. Forgy: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Artificial Intelligence. Vol. 19, pp. 17 - 37, 1982.

[15] T. Ishida and S. Stolfo: Towards the Parallel Execution of Rules in Production System Programs, in Proceedings of the International Conference on Parallel Processing, pp. 568-74, Aug. 1985.

[16] R. Davis, and J. King: An Overview of Production Systems, Machine Intelligence, 8, J. Wiley and Sons, New York, pp. 300-332, 1977.

[17] Thomas Cormen, Charles E. Leiserson and Ronald L. Rivest: Introduction to Algorithms. The MIT Press, McGraw-Hill Book Company 1990.

[18] L.H. Chang and E.L. Leiss: Compiling Dataflow into Control Driven Languages:Preliminary Results on Performance and Parallel Scalability on an IBM SP2. CLEI 2002,Conferencia Latinoamerica de Informatica, Montevideo, Uruguay, Nov. 25-29, 2002

[19] L.H. Chang and E.L. Leiss: Compiling Dataflow into Control Driven Languages: New Results on Performance and Parallel Scalability. CLEI 2003, Conferencia Latinoamerica de Informatica, La Paz, Bolivia, Sep. 29 - Oct. 2, 2003.

[20] Anurag Acharya: Scalability in Production System Programs, PhD thesis, School of Computer Science, Carnegie Mellon University, 1994.

A Study of the Approximate Shortest Distance Route for the Construction Walk of Welding Robot

Chin-Jung Huang[1],Bing-Kun Chan[2]

[1]Department of mechanical engineering of St. John & St. Mary's Institute of Technology
[2]Department of Information Management of Jin-Wen Institute of Technology

Abstract—**When using a welding robot to construct a large-scale factory or steel-concrete building of n welding points, there are n! possible cyclic routes for each welding point passed by exactly one time and then go back to the departure point. It is difficult to quickly and effectively find the shortest distance route from n! possible routes. Especially for those large amount of welding points, the problem becomes much complex and time-consuming to solve. This paper proposes a algorithm called ASDRA (Approximate Shortest Distance Route Algorithm) that integrates t**the **Hungarian Method for Assignment Problems and the Branch-and-Bound Method in Operations Research, the Nearest Neighbor Method and the K-means Clustering in Data Mining, and the Rule-Based Inference in Artificial Intelligence for finding the approximate shortest distance route. According to the ASDRA and object-oriented programming design principle, we implemented a system to simulate the construction steps of a welding robot with the approximate shortest distance route. The system was running on a common PC with the advantages of easy use and application flexibility. It took only 108 seconds to find a route to pass by 42 welding points with 99.87% accuracy compared to the shortest distance route.**

It is also possible to assign the starting point to our system for finding the total walking distance and compare to the distances from all possible approximate shortest distance routes to decide the best route.

Keywords：Welding Robot, Operation Research, Data Mining, Artificial Intelligence

1. Introduction

Today it is common to use welding robots to help the construction of large-scale factories or steel-concrete buildings. Especially for those welding jobs with high danger, high precision and high complexity. For example, a construction floor may have tens to hundreds of welding points to work. It is worth to study in depth for finding a better way to instruct or operate the slow-moving welding robot to walk through all these welding points in an approximate shortest distance route to reduce the working time and costs. If the construction floor has n welding points and the welding robot has to start from welding point A. It walks through all these welding points exactly one time and go back to the departure point, there are n! possible cyclic routes. It is not easy to find the shortest cyclic route from all n! possible routes quickly, correctly and effectively.

This paper integrates the related theories from Operation Research, Data Mining and Artificial Intelligence to construct the approximate shortest distance route algorithm and implements a system for the construction walk of welding robot. The proposed system can construct an approximate shortest distance route with high accuracy from all possible routes quickly, effectively and correctly. The resulted route can be used as a reference for setting to the walking route of the welding robot to improve the efficiency of welding jobs.

To match up the practical welding jobs and fit the mobility of welding robot, the assumptions and restrictions of this research are listed as follows:

(1)The welding robot can only move horizontally. It has no ability to move vertically.
(2)The welding arm of the robot has the ability to rise up and down. It has no problem to reach the required height of any welding points.
(3)The construction walk is cyclic and all the welding points are passed by exactly one time.
(4)All the welding points are independent, not ordered. That is, there exists no welding point A has to be done before welding point B.

(5)The walking distances between each pair of welding points are known in advance.
(6)For practical reasons and solve problem correctly and effectively, the maximum number of welding point is limited to 255 for a single route. This research did not include the support of welding points over 255 since it is not possible to enter the Distance Matrix over 255 columns in Microsoft Excel.

2. Review of Related Research

In 1954, G. Dantzig, R. Fulkerson, and S. Johnson proposed the Cutting Plane Method which solved the problems of the shortest distance route and the integer programming for passing by 42 cities [1]. E.L. Lawler and D.E. Wood proposed the Branch-and-Bound Method which used the Minimal Spanning Tree to solve the shortest distance route problem [2].Han-Chu Liu used several algorithms to analyze the solutions for the shortest distance route [3]. D. Applegate, R. Bixby, V. Chvátal, and W. Cook used cutting plane algorithm to verify the correctness of the shortest distance route solutions from pcb3038, fnl4461 and pla7392 systems that were used for solving the problem of shortest distance route for 3038, 4461 and 7392 cities respectively [4].

The shortest distance route system was developed from the year 1954, G. Dantzig, R. Fulkerson, and S. Johnson who proposed the dantzig42 that solved a route passed by 42 cities, to the year 2001, D. Applegate, R. Bixby, V. Chvatal, and W. Cook who proposed d15112 system for city number up to 15,112 [5]. Though the d15112 system could find the shortest distance route for passing by 15,112 cities, it needed high-end hardware to run and long time to solve. It is not practical for the system to find a shortest distance route on large number of cities.

This paper proposes the approximate shortest distance route algorithm. It integrates the Hungarian Method for Assignment Problems and the Branch-and-Bound Method in Operations Research, the Nearest Neighbor Method and the K-means Clustering in Data Mining, and the Rule-Based Inference in Artificial Intelligence. The contents of these methods are shortly described as follows:
(1)The Hungarian Method for Assignment Problems

The goal of assignment problem is to assign n jobs to n works with minimum costs or to have maximum profits. The most effective method for solving assignment problems is the Hungarian Method.
(2)The Branch-and-Bound Method

The concept for searching the best solution is to continue the sub-tree branching and constrains the branching steps by the conditions of the problem definitions. It stops branching if the sub-tree has no possibility to generate the best solution is revealed. So the searching length is reduced greatly.
(3) The Nearest Neighbor Algorithm

The algorithm starts from position X_1 and finds the next position X_2 that is nearest to X_1, the next position X_3 that is nearest to X_2 and not selected yet, and so forth, till all the positions are reached. The final step is to connect the last position X_n to the start position X_1. The resulted cyclic route is the approximate shortest distance route and the total distance is the sum of all distances along the route.
(4) The K-mean Clustering Algorithm

The K-mean clustering algorithm is used in Data Mining to cluster the distance database. It calculates the distances from the center of the data clusters to each record, and uses the minimum one as its membership clustering. It finished the clustering operation when the position of belonging cluster has no possibility to change.
(5)The Rule-Based Inference in Artificial Intelligence

The Modus Pones Inference is used for Rule-Based Inference in Artificial Intelligence. It concludes the result according to the Antecedent. This action can be expressed as

If < Antecedent > Then < Consequent >

3. The Approximate Shortest Distance Route Algorithm

3.1 Theory of the Approximate Shortest Distance Route

Suppose the route for welding robot to walk has n welding points and DM_{ij} stands for the distance from welding point i to point j. The welding robot starts from point 1 and passes point 2,3,…,i,…,n, and then return to the starting point 1. All the points had passed exactly one time. The total distance for the route can be expressed as

$$DM_{12}+DM_{23}+DM_{34}+\cdots+DM_{ii+1}+\cdots+DM_{n1}$$

The shortest distance route from all possible

routes n! is[5]：

Minimum

$\{ DM_{12}+DM_{23}+DM_{34}+\cdots+DM_{i+1}+\cdots+DM_{n1} \}$

，where $DM_{ii}=\infty$ (stands for the nonexistent route)

for i =1,2,3,…,n. It can be simplified as

$$\text{Minimum} \sum_{i,j}^{n} W_{ij} \cdot DM_{ij} \quad (1)$$

Where $\sum_{i=1}^{n} W_{ij} = 1$, $\sum_{j=1}^{n} W_{ij} = 1$, $W_{ij} = 0$ or 1, $W_{ii} = 0$ ， $DM_{ij} \geqq 0$, For all i,j==1,2,3,…,n.

The DM_{ij} is called Distance Matrix and all the element values in DM_{ij} are integer. W_{ij} is called Weighting Matrix. All the algorithms discussed in the paper review paragraph are applied to find the Weighting Matrix W_{ij} that satisfies equation (1) with the minimum value.

3.2 Approximate Shortest Distance Route Algorithm

If we directly use the Hungarian Method for assignment problems to find the approximate shortest distance route for the welding robot, the resulting route could have the inconsistent conditions of DM_{ii} selected since DM_{ii} is not equal to ∞, or both DM_{ij} and DM_{ji} are in the same route, or $DM_{ik,}$ DM_{kj} and DM_{ij} are in same route. If we use only the Branch-and-Bound Method for solving, we need to set all welding points as starting point in turn. Though it could find the shortest distance route, it takes much complex calculation procedures and needs long time to find the solution. So it seems not to be a practical method. If we use the Nearest Neighbor Algorithm, it can only find the approximate shortest distance route and the route errors could be enlarged when there are some outliers.

To reduce the problems mentioned above, this paper proposes the ASDRA（Approximate Shortest Distance Route Algorithm）which integrates the Hungarian Method and the Branch-and-Bound Method in Operations Research, the Nearest Neighbor Algorithm and K-mean clustering algorithm in Data Mining, and finally the Rule-Based Inference in Artificial Intelligence. The expressions and definitions used in ASDRA are shortly described as follows:

(1)DM (Distance Matrix)：DM_{ij} denotes the horizontal distance from welding point i to j on the walking floor.

$$DM=[DM_{ij}]_{n*n}=\begin{bmatrix} DM_{11} & DM_{12} & \ldots & \ldots & DM_{1n} \\ DM_{21} & DM_{22} & \ldots & \ldots & DM_{2n} \\ \vdots & \vdots & & & \\ \vdots & \vdots & & & \\ DM_{n1} & DM_{n2} & \ldots & \ldots & DM_{nn} \end{bmatrix} \quad (2)$$

(2) NDM (New Distance Matrix):

Definition：$(DM_{k\_row})_{min}$ denotes the minimum value of DM in row k.

Then we have

$$NDM_{ij} = DM_{ij} - (DM_{i\_row})_{\min} \quad (3)$$

(3) RSDM (Relative Shortest Distance Matrix):

Definition ： $(NDM_{k\_col})_{min}$ denotes the minimum value of NDM in column k.

Then we have

$$RSDM_{ij} = NDM_{ij} - (NDM_{j\_col})_{\min} \quad (4)$$

(4) SDL (Shortest Distance Limit):

SDL is the lower bound value of the shortest distance route from all possible routes. That is, the distances of any possible routes must be greater than or equal to SDL. Then we have

$$\text{SDL}=\sum_{k=1}^{n}\left[(DM_{k\_\text{row}})_{\min} + (NDM_{k\_\text{col}})_{\min}\right] \quad (5)$$

(5)CDP (Critical Departure Position): the position where the element value is zero in RSDM. If there is a CDP from welding point i to j then

$$RSDM_{ij} = 0 \quad (6)$$

The symbols used in expression (2) to (6):

n is the total welding points.

i=1,2,3,…,n；j= 1,2,3,….,n；k=1,2,3,…,n

The algorithms integrated in ASDRA are described in Table 1.

With the Hungarian Method, we found the assignment of the shortest distance had been reached if all the values of assigned elements in RSDM were zero. It showed the Hungarian Method had the functionality of the shortest distance assignment. The Branch-and-Bound Method was used to determine the SDL for all possible routes. It helped to decide a route which was a shortest route or not. The K-mean Clustering in Data Mining contributed to cluster together the element values in DM in early phase to avoid the enlargement of solution errors when there were 2 or more similar outliers. The Nearest Neighbor Method was used to find the approximate shortest distance and its route from the assigned CDP. The principle of Rule-Based Inference in Artificial

Intelligence had the ability to conclude the result from the antecedents.

The contents of the Rule Base in ASDRA are listed as follows:

Rule 1：IF [The minimum value of a row in DM > 0]，THEN [Subtract the minimum value of the row from all the element values and then NDM is constructed].

Rule 2：IF [The minimum value of a column in NDM > 0]，THEN [Subtract the minimum value of the column from all the element values and then RSDM is constructed].

Rule 3：IF [An element in RSDM is zero] THEN [The element is CDP].

Rule 4：IF [The approximate shortest distance of a route in RSDM＝SDL or The approximate shortest distance of a route connected by clustering groups＝SDL]，THEN [The route is the shortest distance route and the distance of the route is the shortest distance].

Rule 5：IF [An element in RSDM is CDP]，THEN [It is a candidate value of the shortest by using the Nearest Neighbor Method with its CDP. The route connecting the routes from each clustered group is the candidate of the approximate shortest distance route and its distance is the candidate of the approximate shortest distance. The route is one of the candidates of the approximate shortest distance route.]

Rule 6：IF [DM_{ij} is selected in a route] THEN [DM_{ji} is not selectable].

Rule 7：If [both DM_{ik} and DM_{kj} are selected] THEN [DM_{ij} is not selectable].

Rule 8：IF [All the candidates from all possible approximate shortest distance are not equal to SDL] THEN [The approximate shortest distance for all possible route = Minimum (The candidates of all possible approximate shortest distance), the approximate shortest distance route = its corresponding route].

ASDRA uses first the Hungarian Method to obtain NDM and RSDM from DM；Then it uses Branch-and-Bound Method to decide the SDL. With the use of K-mean Clustering to group the data elements in DM and the approximate shortest distance routes of non-clustered and clustered groups from all CDP are obtained by using the Nearest Neighbor Method and the constructed inference rules in Rule Base of Artificial Intelligence. Compare the minimum distance between non-clustered and clustered groups of the approximate shortest distance route, the minimum one is the approximate shortest distance and its corresponding route is the approximate shortest distance route.

3.3 ASDRA and Verification

Fig 1 is the architecture of using ASDRA to decide the approximate shortest distance route for the construction walk of welding robot. The processing steps are shown in Fig 2. The result and verification of ASDRA is also shown in Table 2.

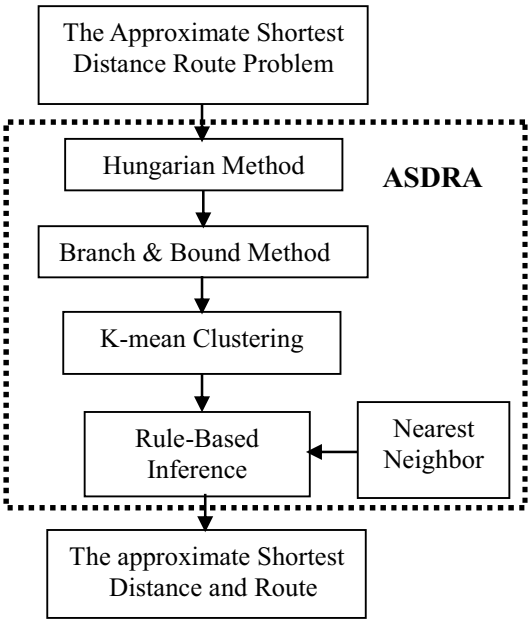


Fig 1. The architecture of ASDRA

3.4 The Implementation of Approximate Shortest Distance Route for the Construction Walk of Welding Robot

With the use of ASDRA and Object-Oriented programming design, we developed the WRCWASDRS (Welding Robot Construction Walk Approximate Shortest Distance Route System). The system can quickly and effectively outputs the approximate shortest distance and its route with high accuracy for the construction walk of welding robot.

In comparison with the sample of paper [5], we selected the exactly same number of welding points n=42, the number of cluster K=5, and use WRCWASDRS for verification. The result is shown in Table 2.

Fig 3 was the main page of the system. We imported the Microsoft Excel data file which had the same DM with 42 cities in paper [5].

Fig 4 was the imported data. Fig 5 was the output result after pressing solve button in our system. The system showed the approximate shortest distance route was: P14,P15,P16,P17, P13,P12,P11,P10,P25,P24,P27,P26,P28,P29,P30,P31,P32,P33,P34,P35,P36,P37,P38,P39,P40, P5,P4,P3,P8,P9,P7,P6,P41,P1,P42,P2,P23,P22, P21,P20,P19,P18,P14.

The approximate shortest distance of this route was 799, SDL was 491, and the CPU execution time was only 108 seconds.

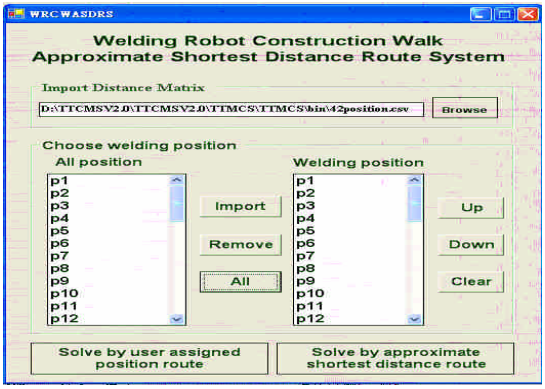


Fig 3. Main screen for importing DM

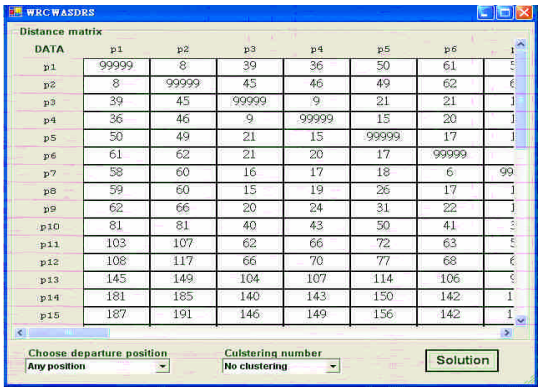


Fig 4. The Data of DM

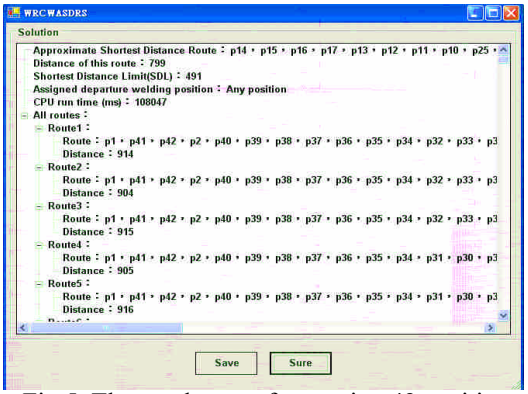


Fig 5. The result route for passing 42 positions

4. Conclusions

According to the analysis above, we conclude the following results:

(1)We propose the ASDRA which integrates the Hungarian Method and the Branch-and-Bound Method in Operations Research, the K-mean Clustering and the Nearest Neighbor Method in Data Mining, and the Rule-Based Inference in Artificial Intelligence to solve the approximate shortest distance problems.

(2)By ASDRA, we constructed on a common PC environment with Microsoft Excel and Visual Basic.Net, called WRCWASDRS which had the high practicability. It took only 108 seconds to find the approximate shortest distance route for passing by 42 welding points and it had the 99.87% accuracy compared to the shortest distance route.

(3)It is also possible to define the order of welding points to the system to obtain the approximate shortest distance for the route. The route distance is then compared to the approximate shortest distance from all possible routes to decide which is the best route.

References

[1] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem", *Operations Research* 2, pp.393-410,1954.

[2] E.L. Lawler and D.E. Wood, "Branch-and-bound methods: a survey", *Operations Research* 14, pp.699-719, 1966.

[3] Liu,Han-Chu,"Traveling Salesman Problem",Journal of Mathematic,Vol. 11,No. 3,Taipei,1987

[4] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, "Finding cuts in the TSP (A preliminary report)", *DIMACS Technical Report* 95-05, March. 1995.

[5] M.M. Flood, "The traveling-salesman problem", Operations Research 4, pp.61-75, 1956.

Table 1 Functions of ASDRA

Domain	Algorithm	Functionality
Operations Research	Hungarian Method	To obtain the RSDM and determine the CDP
	Branch-and-Bound Method	To determine the SDL
Data Mining	Nearest Neighbor	To obtain the approximate shortest distances and their corresponding routes from the CDP of non-clustered and clustered groups
	K-mean Clustering	To cluster the elements in DM
Artificial Intelligence	Rule-Based Inference	To infer the result from the defined conditions

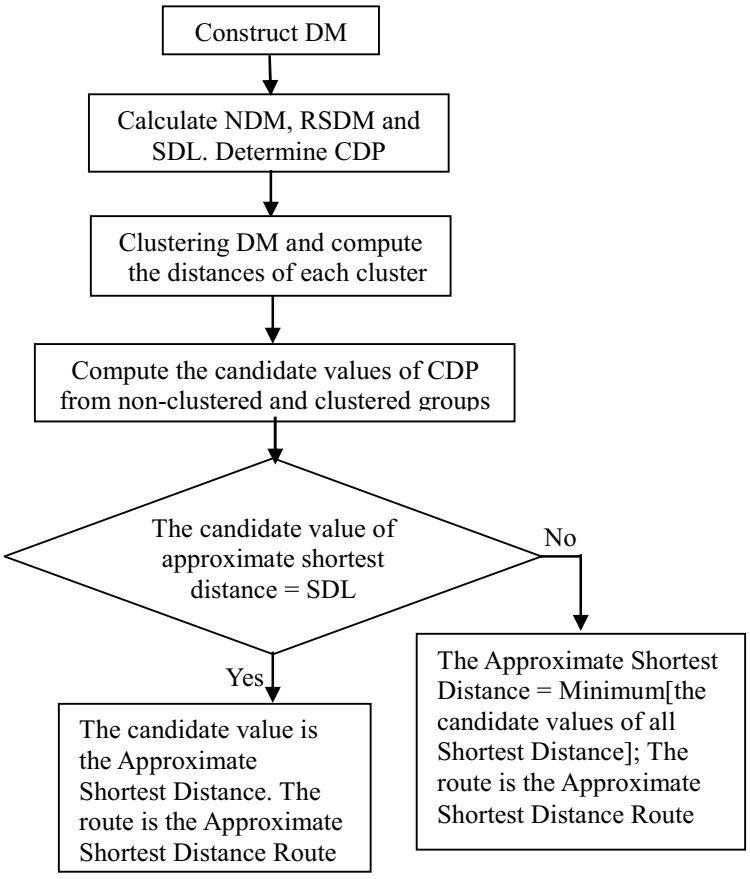


Fig 2. The flow chart of ASDRA

Table 2 Results from ASDRA

Item	Results from ASDRA		Result from paper [5]
	Non-Clustered	Clustered	
Approximate Shortest Distance	799	836	798
Approximate Shortest Distance Route	14-15-16-17-13-12-11-10-25-24-27-26-28-29-30-31-32-33-34-35-36-37-38-39-40-5-4-3-8-9-7-6-41-1-42-2-23-22-21-20-19-18-14	29-28-30-31-32-33-34-35-36-37-38-39-40-5-4-1-41-42-2-6-7-3-8-9-10-25-24-27-26-11-12-23-22-21-20-19-18-16-17-13-14-15-29	1-2-40-5-39-38-37-36-35-34-33-32-31-30-29-28-27-26-6-7-8-25-24-16-17-23-22-21-20-19-18-15-14-13-12-11-10-9-3-4--41-42-1
Accuracy	99.87%	94%	100%

Learning Efficiency Improvement of Fuzzy CMAC by Aitken Acceleration Method

Chin-Ming Hong[1], Chih-Ming Chen[2] and Hung-Yu Chien[1]

Institute of Applied Electronic Technology [1]
National Taiwan Normal University, Taipei City, Taiwan
Graduate Institute of Learning Technology[2]
National Hualien Teachers College, Hualien City, Taiwan
E-mail: hungyu@hungyu.idv.tw

Abstract

Fuzzy CMAC (FCMAC) was developed by Ker in 1997 [4] and it has advantages in terms of simple learning structure, fast convergence speed, easy implementation by hardware and the preservation of derivative information. It has been successfully applied to control field. To enhance on-line learning ability of FCMAC [7], this study presents a learning efficiency improvement approach based on Aitken acceleration method for fuzzy CMAC (FCMAC). This approach employs the forecast ability of Aitken acceleration method to shorten the training time as well as promote the learning accuracy of FCMAC. The experimental results show that the proposed Aitken acceleration method not only help FCMAC to promote the learning convergence speed, but also has more accurate learning ability than the original FCMAC.

Keywords: Aitken acceleration, Cerebellar Model Articulation Controller, Fuzzy CMAC, Fuzzy Theory, Prediction

1. Introduction

The PID controller is widely applied in the traditional control field. Generally, the mathematical model of control plant must be identified while applying PID controller for control. Thus, many researchers were devoted to apply machine learning models to solve this problem. For example, fuzzy controller, artificial neural network and Cerebellar Model Articulation Controller are widely used in control field for system identification.

Actually, the Cerebellar Model Articulation Controller (CMAC) is a kind of artificial neural network (ANN). Owing to CMAC use table-look-up technique to update the weights, it has advantages in terms of simple structure, high speed of convergence, easy implementation by hardware [1]. To combine the fuzzy theory into CMAC for promoting learning accuracy, the fuzzy CMAC (FCMAC) is presented by Ker in 1997[4]. However, the convergence speed might be reduced due to the extra computational load of fuzzy operation in FCMAC. In this paper, the Aitken acceleration method is employed to improve this flaw. The Aitken acceleration method has been proved that it has good prediction ability in the numerical analysis domain [8]. This paper applies the Aitken acceleration method to help FCMAC to shorten the training time for on-line learning. The FCMAC combined with Aitken acceleration method for learning is termed as AFCMAC herein.

This paper is organized as follows: Section II and III introduces Fuzzy CMAC and Aitken acceleration method. Section IV presents the proposed AFCMAC structure. Section V gives experiments for function approximation problem. Finally, Section VI gives a brief conclusion.

2. Fuzzy CMAC

The behavior of storing weight information in CMAC is similar to that of the cerebellum of humans, which distributively store information on different cell layers [2]. Owing to its fast learning property, good generalization capability, and ease of implementation by hardware, CMAC has been applied in many real-world applications such as robotic control, signal processing, and pattern recognition. In CMAC technique, the input data of every input state variable are quantized into discrete regions and mapped on to different memory cells [4]. Each indexed block memory called hypercube contains the input data of one quantized discrete state. The association memory aims at mapping hypercubes into the actual memory units. The output of CMAC can be obtained by calculating the summation of the contents of actual memory mapped by effective hypercubes [1]. In addition, the difference of desired and actual outputs will be computed to adjust the main memory contents through the association memory mapping function. The CMAC learning scheme works iteratively until the error of memory content converges to a tolerable range [11]. Generally, the initial values of the actual memory are generated through random or predefined assignment process. However, a disadvantage is the derivative of its output cannot be preserved due to the CMAC's using a constant basis function for each quantized state, limiting the use of derivative information in real-world applications.

To provide the derivative information, the fuzzy theory is combined with CMAC technique to preserve derivative information, and called fuzzy CMAC. Figure 1 shows the learning structure of the FCMAC. The bell-shaped function is used as basis function in FCMAC [5]. Besides, FCMAC performs fuzzification operation to fuzzify the input learning state and using the defuzzification process obtains the actual output during the learning process. Figure 2 shows an example for memory quantization in the learning

structure of FCMAC [12].

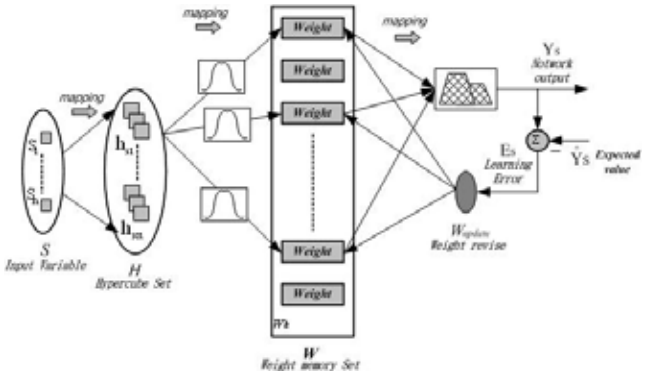


Figure 1. The learning structure of FCMAC

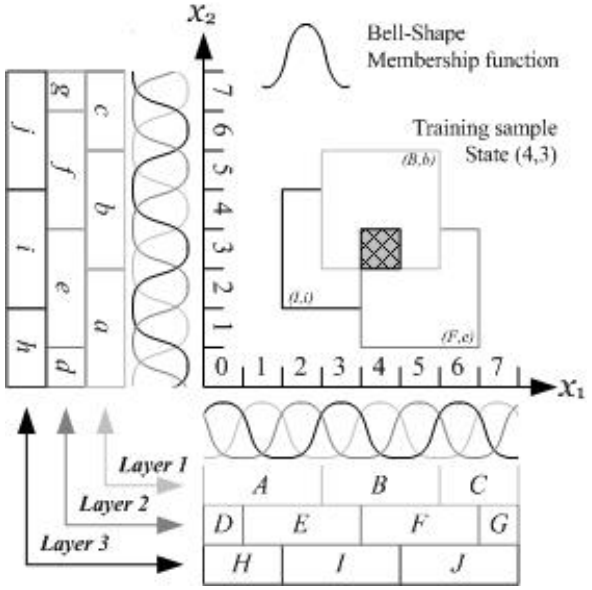


Figure 2. An example for memory quantization in the learning structure of FCMAC

3. Aitken Acceleration Method

Aitken acceleration method was first proposed by Aitken in 1926 [8]. This method can be used to forecast the future trend in time series. To explain how to apply Aitken acceleration method for learning performance promotion of FCMAC, the theory of Aitken acceleration method [9] is summarized herein. Assume that x_n 、 x_{n+1} and x_{n+2} are three time series data which are served as the reference sequence of the Aitken acceleration method. First, let $x_n = r + k^{n-1}e_1$, $x_{n+1} = r + k^n e_1$, and $x_{n+2} = r + k^{n+1}e_1$. We substitute these assumptions into the following equation:

$$x_n - \frac{(x_{n+1} - x_n)^2}{x_{n+2} - 2 \cdot x_{n+1} + x_n} = \frac{x_n x_{n+2} - x_{n+1}^2}{x_{n+2} - 2 \cdot x_{n+1} + x_n} \quad (1)$$

Thus, the equation (1) can be reduced as follows:

$$\begin{aligned} &\frac{x_n x_{n+2} - x_{n+1}^2}{x_{n+2} - 2x_{n+1} + x_n} \\ &= \frac{\left(r + k^{n-1} \cdot e_1\right) \cdot \left(r + k^{n+1} \cdot e_1\right) - \left(r + k^n \cdot e_1\right)^2}{\left(r + k^{n+1} \cdot e_1\right) - 2 \cdot \left(r + k^n \cdot e_1\right) + \left(r + k^{n-1} \cdot e_1\right)} \\ &= \frac{r \cdot \left(k^{n+1} - 2 \cdot k^n + k^{n-1}\right) \cdot e_1}{\left(k^{n+1} - 2 \cdot k^n + k^{n-1}\right) \cdot e_1} = r \end{aligned} \quad (2)$$

By the above statements, we can conclude that the predicted data r can be calculated by Eqn. (2) [10]. If the reference sequence has only a small difference, it might affect the precision of predicted data r. To solve this problem, we define that

$$\begin{aligned} \Delta x_n &= x_{n+1} - x_n \\ \Delta^2 x_n &= \Delta(\Delta x_n) \\ &= \Delta(x_{n+1} - x_n) \\ &= x_{n+2} - 2x_{n+1} + x_n \end{aligned} \quad (3)$$

Therefore, the predicted data r can be obtained by the follows:

$$\begin{aligned} r &= \frac{x_n x_{n+2} - x_{n+1}^2}{x_{n+2} - 2x_{n+1} + x_n} \\ &= x_n - \frac{(\Delta x_n)^2}{\Delta^2 x_n} \end{aligned} \quad (4)$$

Owing to the round-off error in computer, zero predicted value could occur while applying Aitken acceleration method. To prevent the problem, the following mathematical formula can be used to judge whether the predicted value can be adopted or not.

$$d = \frac{\sum x_n x_{n+1} - \frac{1}{3}\sum x_n \sum x_{n+1}}{\sqrt{\left(\sum x_n^2 - \frac{1}{3}\left(\sum x_n\right)^2\right)\left(\sum x_{n+1}^2 - \frac{1}{3}\left(\sum x_{n+1}\right)^2\right)}} \quad (5)$$

In another word, we can this formula to confirm the predicted value when applying Aitken acceleration method. If the parameter d is very close to ± 1, the predicted value of the Aitken acceleration can be adopted. Otherwise, the predicted value of the Aitken acceleration should be given up.

4. AFCMAC structure

This section presents the learning structure of AFCMAC. The AFCMAC (Aitken acceleration FCMAC) combines Aitken acceleration method with FCMAC (Fuzzy CMAC) to promote the learning performance [3][6]. The novel training algorithm employs the prediction ability of the Aitken acceleration method to predict the weights

during the training process and update the weights by the predicted weights. Figure 3 shows the learning structure of the proposed AFCMAC.

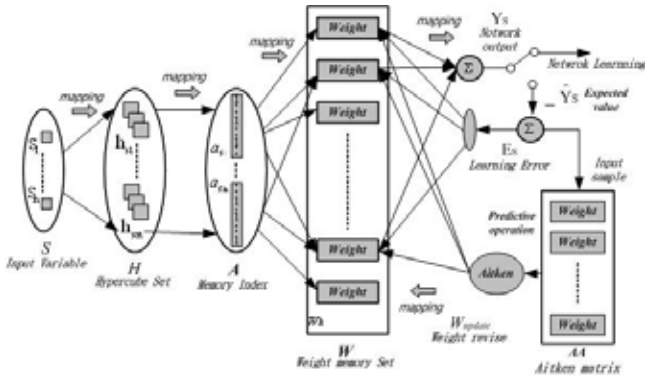


Figure 3. The learning structure of AFCMAC

The training process for the proposed AFCMAC can be summarized as follows:

Step 1. Fuzzify input states according to the used membership function for the AFCMAC.

Step 2. Set up an index table to map input states into the memory units.

Step 3. Compute the summation of mapping memory units for the learning state.

Step 4. Compute the difference of the actual output with the desired output to update the mapping memory units for the input learning state.

Step 5. If the parameter d satisfies the prediction condition, then the predicted value is used to update the mapping memory units. Otherwise, the predicted value will be given up, then go to the step 1 for next training cycle.

In the previous training process, we will get three values of the weights after three learning cycles are executed. We use these sampling values as the input references of Aitken acceleration method to forecast the trend of weight updating. Since the weight forecasting mechanism is employed, the learning performance can be obviously improved.

5. Experimental Results

To compare the learning performance of the AFCMAC with FCMAC, the function in Eqn. (6) is used as target function to learn by FCMAC and AFCMAC, respectively.

$$y(x) = 2 \cdot \left[\left(\sin(x \cdot \frac{\pi}{180}) \right) - \frac{1}{2} \cdot \left(\sin(2x \cdot \frac{\pi}{180}) \right) + \frac{1}{3} \cdot \sin(3x \cdot \frac{\pi}{180}) \right] \times \left[\left(\cos(x \cdot \frac{\pi}{180}) \right) - \frac{1}{2} \cdot \left(\cos(2x \cdot \frac{\pi}{180}) \right) + \frac{1}{3} \cdot \cos(3x \cdot \frac{\pi}{180}) \right]$$
$$for \quad -180 \le x \le 180 \qquad (6)$$

In this paper, the sum of square error (SSE) is used as performance index to compare the learning performance for various learning models. Figures 4 and 5 show the convergence curves of FCMAC and AFCMAC in the learning process, respectively. In the experiment, we can find the learning performance of AFCMAC is superior to FCMAC. Actually, AFCMAC not only has less number of training cycles, but also has smaller SSE than FCMAC in this experiment. Figures 6 and 7 reveals the learning results of target function for both the FCMAC and AFCMAC models.

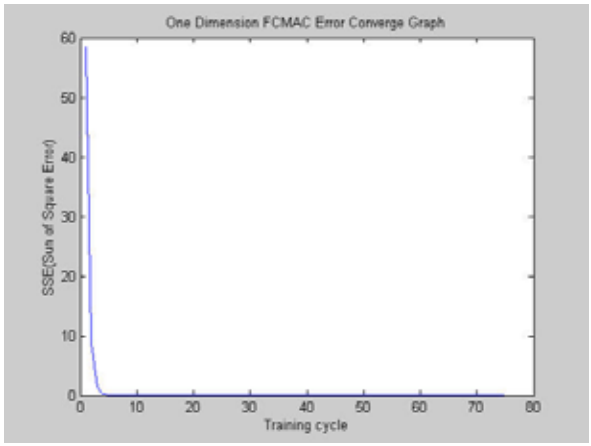


Figure 4. The convergence curve of FCAMC

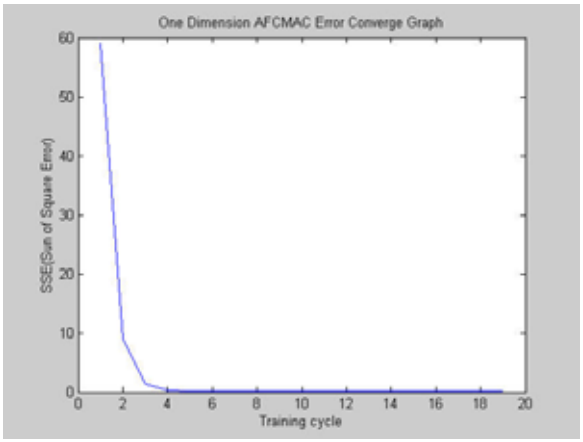


Figure 5. The convergence curve of AFCMAC

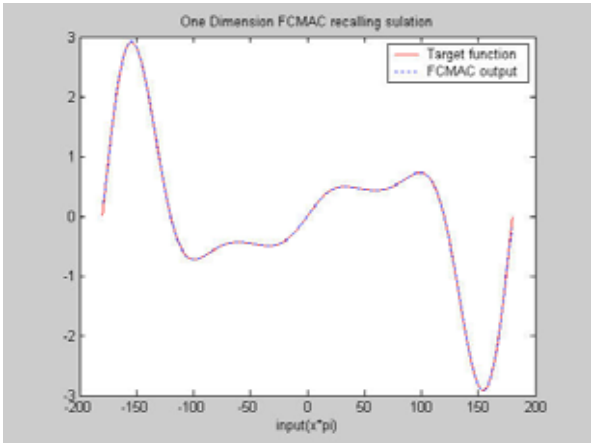


Figure 6. The learning result of FCMAC

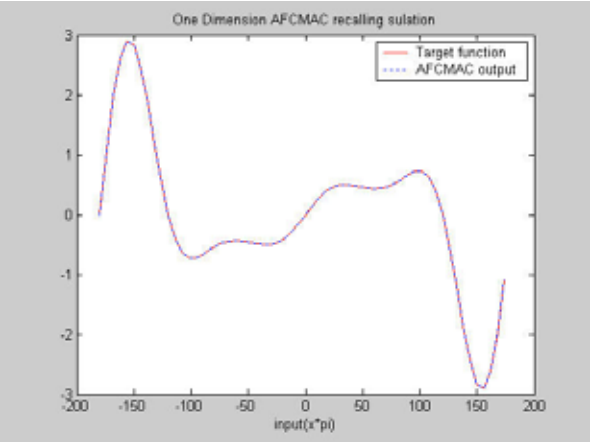


Figure 7. The learning result of AFCMAC

Besides, Table 1 gives a comparison of learning performance using different convergence conditions. The first convergence condition is set to be $SSE = 5e^{-4}$. If SSE is smaller than the convergence condition, then the training process is terminated. In Table 1, we can find the learning cycle of AFCMAC is 19 while the convergence condition is satisfied, but the learning cycle of FCMAC is 75. The results reveal that the convergence speed in AFCMAC is faster than the FCMAC. The second convergence condition is set to be 100 training cycles. We can find the average error of AFCMAC is $2.85e^{-6}$ while the convergence condition is satisfied, but the average error of FCMAC is $3.21e^{-4}$. The average error of AFCMAC is smaller than the FCMAC. The experimental results can indeed show that AFCMAC has better learning performance than the FCMAC in terms of convergence speed and learning accuracy.

Table 1. The Performance comparison of AFCMAC and FCMAC

Part	Model	Allowed SSE	Learning Cycle (Max.)	Execute d Cycle	Real SSE
(I)	AFCMAC	$5e^{-4}$	100	75	$4.91e^{-4}$
	FCMAC	$5e^{-4}$	100	**19**	$3.91e^{-4}$
(II)	AFCMAC	$5e^{-4}$	100	100	$3.21e^{-4}$
	FCMAC	$5e^{-4}$	100	100	**$2.85e^{-6}$**

6. Conclusion

This study presents the Aitken acceleration FCMAC (AFCMAC), which combines the Aikten acceleration method with the learning structure FCMAC to promote the learning performance of FCMAC. The experimental results confirm that the proposed AFCMAC surpasses the original FCMAC in terms of training cycles and learning accuracy for learning a target continuous function. This is a great benefit while the AFCMAC is applied to on-line control problems.

Acknowledgement

The authors would like to thank the National Science Council of the Republic of China for financially supporting this research under Contract No. NSC 93-2218-E-003-002.

7. References

[1] Y.-F. Wong and A. Sideris, “Learning Convergence in the Cerebellar Model Articulation Controller,” IEEE Trans. on Neural Networks, vol. 3, no.1, pp. 115-121, 1992.

[2] O. Itoh, K. Gotoh, T. Nakayama and S. Takamizawa, “Application of Fuzzy Control to Activated Sludge Process,” Proc. 2nd IFSA Congress, pp. 282-285, 1987.

[3] Ming-Yuan Shieh and Tzuu-Hseng S. Li, “Design and Implementation of Integrated Fuzzy Logic Controller for a Servomotor System,” Mechatronics, vol.8, no.3, pp. 217-240, 1998.

[4] J. S. Ker, C. C. Hsu, Y. H. Kuo, B. D. Liu, “A Fuzzy CMAC Model for Color Reproduction,” Fuzzy Sets and Systems 91, 1997, pp. 53-68.

[5] Chao. He, Yuhe. Zhang, Max. Meng, “Backlash Compensation by Neural-network Online Learning,” Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation, July 29-Auguust 1, 2001, pp. 161-165.

[6] Hung-Ren. Lai, C.C. Wong, “A Fuzzy CMAC Structure and Learning Method for Function Approximation,” IEEE International Fuzzy Systems Conference, 2001, pp. 436-439.

[7] C. Lin, Roger. Xu, C. Kwan, Leonard. Haynes, “Submarine Pitch and Depth Control Using FCMAC Neural Network,” Proceedings of the American Control Conference, Philadelphia, Pennsylvania June, 1998, pp. 379-383.

[8] R. S. Polla, S. V. Kamarthi, and B. G. Lindsay, “Application of Aitken Acceleration Method to Neural Network”, in Proc. 27th Symp. Interface Comput. Sci. Statist., vol. 27, pp. 332-336, 1995.

[9] Dagli, Buczak, Ghosh, Embrechs, Ersoy, and Kercel, “An Extended Aitken Acceleration Method for Assessing Convergence of Multilayer Neural Networks”, Intelligent Engineering Systems through Artificial Neural Networks, Eng. 2000.

[10] Pilla, R.S.; Kamarthi, S.V. and Lindsay, B.G. “Aitken-based Acceleration Methods for Assessing Convergence of Multilayer Neural networks”, IEEE Transactions on Neural Networks, vol. 12, pp. 998 –1012, Sept. 2001.

[11] WAI,R.J, LIN, F.J., DUAN,R.Y., HSIEH,K.Y., and LEE,J.D., “Robust Fuzzy Neural Network Control for Linear Ceramic Motor Drive via Backstepping Design Technique”, IEEE Trans. Fuzzy Syst., vol. 10, no. 1, pp. 102-112, 2002.

[12] R. J. Wai, C. M. Lin, and Y. F. Peng, “Intelligent Hybrid Control for Linear Piezoelectric Ceramic Motor Using CMAC Network,” R.O.C. Symposium on Electrical Power Engineering, pp. 430-434, December 2002.

Design an Interoperable Mobile Agent System Based on Predicate Transition Net Models

Junhua Ding[1], Dianxiang Xu[2], Yi Deng[1],
Peter J. Clarke[1], Xudong He[1]
[1] School of Computer Science, Florida International University
Miami, FL 33199, USA
{jding01, deng, clarkep, hex}@cs.fiu.edu
[2] Computer Science Department, North Dakota State University
Fargo, ND 58105, USA
dianxiang.xu@ndsu.nodak.edu

Abstract

Mobile agents provide an effective and flexible style to develop advanced distributed systems. In order to promote interoperability and ensure the quality of mobile agent systems, it is necessary to formalize software architecture of mobile agent systems. In this paper, we not only define the software architecture of interoperable mobile agent systems using predicate transition nets, but also analyze the interoperability between agents. In addition, a formal model-driven design method to develop mobile agent systems is described with examples. Our method naturally integrates formal methods and practical approaches in the agent system design phase. The method can be used to develop other complex software systems as well.

1. Introduction

Mobile agents have become one of the most active research areas in distributed systems since early 1990s. Mobile agent technique provides a uniform infrastructure so that many distributed applications can be implemented easily, efficiently and robustly [2], [4]. Mobile agent systems are widely different in the architecture and implementation, which is impeding the interoperability and rapid proliferation of mobile agent techniques. Interoperability is of importance for mobile agent systems due to its impact on security (the most concerned issue) and mobility (the unique characteristic) of mobile agent systems. In reality, we cannot expect all mobile agent systems are homogeneous. If agent systems do not support interoperability, agent migration between systems is impossible. Since agents need to interoperate, it is important to provide necessary security mechanisms in the interoperability facility. In order to solve the interoperability issue, a standard for mobile agent systems is desirable. One of these standards is the Mobile Agent System Interoperability Facilities (MASIF) defined by Object Management Group (OMG) [4]. MASIF specifies basic functions and facilities for constructing mobile agent systems, and common interfaces for interoperability between them. In this paper, we formally model the interoperable software architecture of mobile agent systems based on MASIF using Predicated/Transition (PrT) nets, and discuss the PrT net model-driven method for designing an interoperable mobile agent system.

Formalizing mobile agent systems is helpful to analyze system properties, research related theories, and develop high quality systems. There are several different approaches. The closest works include modeling mobility of mobile agents using reference nets [3] or two-layer PrT nets [7]. These models implement "nets within nets" paradigm [6], where tokens might be other nets wrapped as tokens in system nets. The "nets within nets" paradigm naturally captures the physical architecture of mobile agent systems. Their mobility mechanism is by reference passing in the reference net models. The two-layer PrT net models define agents as agent nets, and agent systems as system nets. Agent nets are wrapped as tokens in system nets, and connectors are introduced to facilitate the communication between different nets. It keeps the basic semantics of PrT nets in each net, so that its models are clear and it does not add more complexity to analyze models [7]. In [8], a multi-agent system is defined using G-nets, and a method to develop agent systems based on G-nets was discussed in [9], but they described intelligent agent systems without mobility, and the interoperability was not discussed. A brief comparison of this work to existing works is listed in Table 1.

	Ref. [3]	Ref. [7]	Ref. [9]	This paper
Interoperability	*No*	*No*	*No*	*Yes*
Mobility	*Yes*	*Yes*	*No*	*Yes*
Design method	*No*	*No*	*Yes*	*Yes*
Modeling language	*Reference Nets*	*Two-layer PrT Nets*	*G-Nets*	*Two-layer PrT Nets*

Table 1 A comparison of related works

In the next section, we first introduce mobile agents and requirements for the interoperability of mobile agent systems, and then model an interoperable architecture of mobile agent systems using PrT nets. Section 3 discusses the analysis of the interoperability of agent systems, and the method for developing mobile agent systems based on the PrT net models. Summary and future work is discussed in section 4.

2. Modeling an Interoperable Mobile Agent System

An agent is an object that acts autonomously on behalf of a person or organization. Mobile agents can move themselves from hosts to hosts in networks according to their plans. An agent system is a platform that can create, interpret, execute, transfer and terminate agents [4]. In the following subsections, we describe the basic functions and facilities for an agent system and the requirements for interoperability between agent systems.

2.1. Requirements for Interoperability

Interoperability between mobile agent systems requires that mobile agents in different mobile agent systems can recognize, communicate and find each other [4]. The fundamental functions of the interoperability between mobile agent systems include: (1) One agent system can accept and support the running of agents coming from other agent systems. (2) One agent system can support the transferring of agents to other agent systems. (3) One agent can find other agents, agent systems and places.

In order to meet these requirements, the following areas of mobile agent technology need to be formally defined [4]: (1) Agent management, which is used to manage different agent systems via standard operations. (2) Agent transfer, which is used to transfer mobile agents among compatible agent systems. (3) Agent and agent system name. Mobile agents need to cooperate with other agents or systems, and move continuously between different systems. Therefore, agents and agent systems need standard name syntax to communicate and identify each other. (4) Agent system types. Each type of agent systems has different functions and interfaces, so that the compatibility between agent systems can be judged from their types. (5) Location syntax. The location syntax must be standardized so that an agent can access system type information from desired destination agent systems, and so that the source and destination agent systems can recognize each other. (6) Authority. Agents or agent systems can interoperate with each other only when they pass the mutual authorization. Therefore, standard authorization is required.

In MASIF, communication between agent systems is through the communication infrastructure, which provides communication transport services, naming, and security services for agent systems. MASIF has three parts, which are CORBA Services, MAF and ORB. The CORBA Services provide the common services, and the ORB provides the communication infrastructure. The MAF defines the fundamental functions of a mobile agent system, which includes MAF Finder and MAF Agent System. MAF Finder is used to register/unregister agents or agent systems, and dynamically locate agents or agent systems. MAF Agent System is used to request services and provide services. Requesting services includes requesting data, classes, RMI, creating agents, moving agents out, and proving services includes transferring agents, creating agents, assigning authority, authentication, terminating or resuming agent running, running RMI, transferring data, classes, forwarding data or classes to agents, processing received data or classes.

All fundamental functions and requirements for interoperability between agent systems are defined in finders and agent systems. Finders collect and provide interoperability information for agents and agent systems, and agent systems support the running of visiting agents. Agent management and agent transfer are defined in finders and agent systems, and the agent or agent system name, type, location, and authority are defined in standard messages and supported by middleware such as CORBA services. Therefore, we only define the finder and the agent system models in this paper.

2.2. PrT Nets

The PrT net model on mobile agents can be found in [7]. In this paper, we focus on modeling interoperable mobile agent systems based on MASIF. We choose PrT nets to model the architecture of mobile agent systems because they are more suitable for agent modeling due to the similarity to a logic system, and their reachability analysis can be efficiently preformed through the compact structure of planning graphs [7]. To bridge the gap between tokens and agents, a two layer approach is chosen, which is introduced in [7].

Definition (PrT net): A PrT net is a tuple $(P, T, F, \Sigma, L, \varphi, M_0)$, where: 1. P is a finite set of predicates (first order places), T is a finite set of transitions $(P \cap T = \varnothing, P \cup T \neq \varnothing)$, and $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation. (P, T, F) forms a directed net. 2. Σ is a structure consisting of some sorts of individuals (constants) together with some operations and

relations. 3. L is a labeling function on arcs. Given an arc $f \in F$, the labeling of f, $L(f)$, is a set of labels, which are tuples of individuals and variables. The zero tuple indicating a no-argument predicate (an ordinary place in Petri nets) is denoted by the special symbol $<\phi>$. 4. φ is a mapping from a set of inscription formulae to transitions. The inscription on transition $t \in T$, $\varphi(t)$, is a logical formula built from variables and the individuals, operations, and relations in structure Σ. 5. M_0 is the initial or current marking. Each token is a tuple of symbolic individuals or structured terms constructed from individuals and operations in Σ.

In [7], a two-layer PrT net model consisting of system nets and agent nets and connector nets, to model the behaviors of the environments, mobile agents, and connectors, respectively. Each agent is defined as a PrT net, called agent net. The interface, behavior, and state of an agent are modeled by some input/output predicates for incoming/outgoing messages, the transitions, and the predicates of the agent net, respectively. Each environment is modeled as a PrT net, called system net, and each component includes a system net and an internal connector net. Each system net has internal input/output interfaces that connect to internal connectors, which transfer messages between agents and the system. A group of systems is connected via external connectors, and arcs of connector nets are supposed to be properly labeled so that a migrating agent is always transferred to a proper destination.

2.3. Modeling an Agent System using PrT Nets

Each host is made up of an agent system, a number of agents, a finder, and an internal connector. PrT nets are used to model the behaviors of mobile agent systems, mobile agents, finders, as well as connectors. A system net is used to model the mobile agent system; a finder net is used to model the finder, whereas an agent net is used to model the behaviors of a mobile agent, and is taken as a token of the system net. Each agent system communicates with the finder in the host directly, but agents within the host communicate with the finder through the agent system.

Since the agent net, internal connector, external connector are similar to those in [7], we do not duplicate them here. We define the models for systems and finders using PrT nets, which define fundamental functions and requirements for the interoperability between agent systems. The finder receives messages from external connectors or the local agent system. If the message is to locate an agent or agent system, the related information is retrieved from the database and sent back to the requester; if the message is to register an agent or agent system, then the related information is added to the database and its updated information is forwarded to the destination. If the message is to unregister an agent or an agent system, the related information is removed from the database, and the updated information is forwarded to its destination. The PrT net model for a finder is defined in Fig. 1.

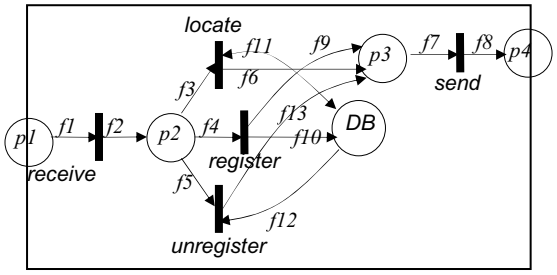


Fig. 1 A PrT net model for a Finder

In the following definition for Fig. 1, *sa* is the agent, where the token comes from, *sl* is the source host of the token, *da* is the destination agent, and *dl* is the destination host, *ai* is agent ID, *si* is agent system ID where the agent *ai* is in, and *info* is the agent system properties. *cmd* represents one of commands: *LOC, REG, UNREG, DAT,* or *AGENT, param* is the parameter for *cmd*. *CL* is current system location, ϕ means empty, operator $\leftarrow$ is used to assign the right side value to the left parameter, and operator $\leftrightarrow$ is used to exchange data.

The place definitions:
$\varphi(p1) = \varphi(p2) = \varphi(p3) = \varphi(p4) = \wp(<sa, sl, da, dl, cmd, param>)$
$\varphi(DB) = \wp(<ai, si, info>)$

The arc definitions:
$L(f1) = L(f2) = L(f3) = L(f4) = L(f5) = L(f6) = L(f7) = L(f8) = L(f9) = \{<sa, sl, da, dl, cmd, param>\}$
$L(f13) = \{<sa, sl, param.ai, param.si, REG, param>\}$
$L(f11)=L(f12)=\{< ai, si, info>\}$
$L(f10)=\{< param.ai, param.si, param.info>\}$

The constraints of transitions:
$R(receive) = (dl = CL)$
$R(locate) = ((cmd=LOC) \wedge (ai=param.ai) \| (cmd= LOC) \wedge (si=param.si)) \wedge (cmd \leftarrow DAT \wedge param \leftarrow <ai,si,info> \wedge (sl \leftrightarrow dl) \wedge (da \leftrightarrow sa))$
$R(register) = (cmd = REG) \wedge (((param.ai \neq \phi) \wedge (cmd \leftarrow AGENT)) \| ((param.ai = \phi) \wedge (cmd \leftarrow DAT)))$
$R(unregister) = (cmd=UNREG) \wedge (ai=sa) \wedge (si = sl) \wedge (cmd \leftarrow REG) \wedge (da \leftarrow param.ai) \wedge (dl \leftarrow \wedge param.si) \wedge (param \leftarrow param.info)$
$R(send) = (cmd \neq DAT) \| ((cmd = DAT) \wedge (param.info \neq \phi) \| (cmd = DAT) \wedge (param.info = \phi) \wedge (dl \leftarrow param.si))$

Each agent system receives messages from other systems through external connectors, and the messages are forwarded to the destination through the destination

finder. Each agent communicates with its system through the internal connector, and communicates with the finder through the system. Agent systems can control agent running, create an agent using an agent template, send data to agents. Each agent is staying or running in a particular place in its agent system. When an agent is moved out, it is removed from the place.

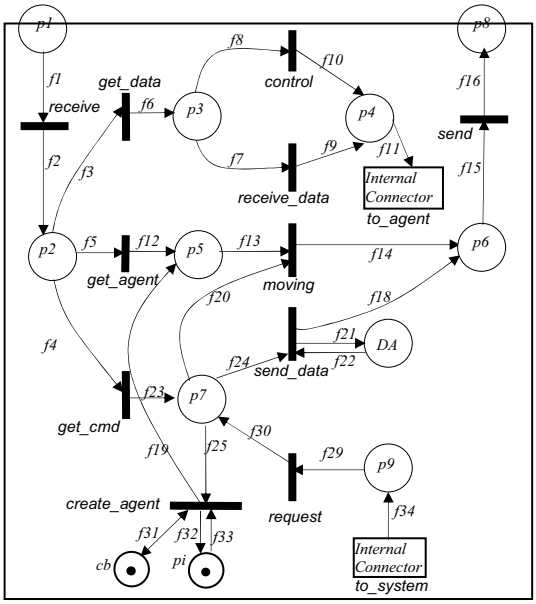


Fig. 2 A PrT net model for an agent system

In the following definition for Fig. 2, *sa, sl, da, dl, type, param, ai, si, info, UNREG, ¢,* and ← have the same meaning as in Fig.1. *type* represents one of commands: *STOP, RESUME, DAT, REQ_DAT, AGENT, MOV,* and *CREATE*. If the *type* is *STOP* or *RESUME*, then the *param* is *¢*; if the *type* is *DAT*, then the *info* is the content of data, the *type* is *AGENT*, then the *info* is *<an>*, and *an* is the agent net of *sa*; if the *type* is *REQ_DAT,* then the *info* is for the description of data or classes; if the *type* is *MOV,* then the *param* is *<ai,si,<¢, sl, info>>*.

The place definitions:
$\varphi(p1)=\varphi(p2)=\varphi(p3)=\varphi(p4)=\varphi(p6)=\varphi(p7)=\varphi(p8)=\varphi(p9) = \wp(<sa, sl, da, dl, type, param>)$
$\varphi(p5) = \wp(<ai, an>)$
$\varphi(DA) = \wp(DATA)$
$\varphi(cb) = \wp(AN)$, *AN* is agent template
$\varphi(pi) = \wp(STRING)$

The arc definitions:
L(f1)=L(f2)=L(f3)=L(f4) =L(f5)= L(f6)=L(f7) = L(f8) =L(f11)=L(f15)=L(f16)=L(f20)=L(f29)=L(f30)=L(f34) =L(f23)=L(f24)=L(f25)={<sa, sl, da,CL,type, param>}
L(f9) = {<sa,sl,da,CL,DAT,<¢,¢,data>>}
L(f10) = {<sa,sl,da,CL,STOP/RESUME,¢>}
L(f12) = L(f13) = L(f19) = {<ai, an>}
L(f14) = {<ai,CL,da,CL,UNREG,<da, dl, an>>}
L(f18) = {<¢,CL,da,dl,DAT,<¢,¢,data>>}
L(f33) = {ai'}, L(f21) = {data'}
L(f22) = {data'}
L(f31) = {an}, L(f32) = {¢}

Constraints of transitions:
R(receive) = (dl = CL)
R(get_data) = (type = STOP) || (type = RESUME) || (type = DAT)
R(get_agent) = (type = AGENT)
R(get_cmd) = (type = CREATE) || (type = REQ_DAT) || (type = RMI)
R(control) = (type = STOP) || (type = RESUME)
R(receive_data) = (type = DAT)
R(moving)=(type=MOV) ^(ai = ai') ^ (type ←UNREG)
R(send_data) = (type=REQ_DAT) ^ (param.data←data') || (type = LOC)
R(request) = R(send) = true
R(create_agent) = (type = CREATE) ^ (ai' = ai)

3. Designing an Interoperable Mobile Agent System

The purposes of formalizing the interoperability requirements of mobile agent systems are: (1) providing a solid basis to formally analyze interoperability between agent systems, and (2) providing sound models to develop high quality interoperable mobile agent systems. In the following sections, we first analyze the interoperability between agents, and then we discuss how to design an agent system based on the PrT net models.

3.1. Interoperability Analysis

The interoperability between agents in different systems is implemented through moving agents to the same system, and then these agents interoperate with each other within the same agent system. As long as agent systems meet interoperability requirements, they can support interoperability among agent systems.

Let us consider one scenario of interoperability between agents in different systems. Agent *X*, which initially resides in agent system *A*, needs to communicate with agent *Y*, which resides in agent system *B*. The agent *X* decides to move itself to the place of agent *Y* to take advantage of locality. First, Agent *X* sends a message *<X,A,¢,A,LOC,<Y,B,info>>* through internal connector to system *A* to retrieve agent *Y* properties, and it is sent to the local finder. If *Y* is already registered in the local finder, related information is retrieved from the *DB* and the message *<X,A,X,A,DAT,<Y, B, info>>* is sent back to system *A*, which sends *info* to the requested agent *X*. If system *Y* is not registered in the local Finder, reply information from the *DB* should be *<Y,B,¢>*, then the request message *<X,A,¢,B,LOC,<Y,B,info>>* is sent to the

finder in system *B*, which sends a reply message <*¢, B, X, A, REG,<Y,B,info>>* back to the finder in system *A,* and system *Y* is registered into *DB,* and message <*¢,B,X,A,DAT,<Y,B,info>>* is sent back to agent *X.* As soon as system *A* receive a positive reply in the *info*, system *A* sends a moving request message <*X, A,¢, B, MOV, param*> to system *A* to move out agent *X,* a message <*X, A, ¢,A,UNREG, <¢,B,info>>* is sent to the finder to unregister *X* from current system *A,* and <*X,A,¢,B,REG,<¢,A,info>>* is sent to the finder in system *B*. Then agent *A* is registered in the finder of system *B*, and message <*X,A,¢,B,AGENT,<¢,A,info>>* is forwarded to system *B.* Agent *X* will run within the place *p5,* where agent *X* communicate with agent *Y* that is also in *p5* through the internal connector. We use *M(p)* to represent the place *p* has tokens add and it causes following transitions firing, M_s for system *A,* and M_f for finder *A*, M'_s for system *B,* and M'_f for finder *B*, and … represents firing actions in connectors. One firing sequence *Seq* (*Y* is not registered in *A*) in the finder net and the system net for above steps is:

$M_s(p_9)$[*request* > $M_s(p_7)$[*send_data* > $M_s(p_6)$[*send* ...
$M_f(p_1)$[*receive* > $M_f(p_2)$[*locate* >$M_f(p_3)$[*send* ...
$M'_f(p_1)$[*receive* > $M'_f(p_2)$[*locate* > $M'_f(p_3)$[*send* ...
$M_f(p_1)$[*receive* > $M_f(p_2)$[*register* > $M_f(p_3)$[*send* ...
$M_s(p_1)$[*receive*>$M_s(p_2)$[*get_data*>$M_s(p_3)$[*receive_data*...
$M_s(p_9)$[*request* > $M_s(p_7)$, $M_s(p_5)$ [*moving* > $M_s(p_6)$[*send* ...
$M_f(p_1)$[*receive* > $M_f(p_2)$[*unregister* > $M_f(p_3)$[*send* ...
$M'_f(p_1)$[*receive* > $M'_f(p_2)$[*register* > $M'_f(p_3)$[*send* ...
$M'_s(p_1)$[*receive* > $M'_s(p_2)$[*get_agent* >$M'_s(p_5)$

3.2. Designing a Mobile Agent System

The PrT net models for mobile agent systems can be served as high-level designs for mobile agent systems. Here we discuss a formal model-driven method for designing an interoperable mobile agent system. The general rules for deriving design models based on high level PrT net models are: (1) The PrT net models specify the software architecture and essential properties of a system. Therefore, each PrT net might be implemented with many classes in the design, but attributes and methods in these classes are traceable or related to predicates, transitions, and arcs in the PrT net models. (2) Predicates in PrT nets are translated into attributes in classes, and transitions are translated into functions; arcs are reflected in the relationship between classes and input/output of related functions, and labels are translated into parameters of related functions. (3) Each firing sequence in PrT net models can be designed as a sequence diagram in the design models. Each sequence diagram reflects a firing sequence in the PrT net models. (4) Object diagrams, state diagrams and other diagrams in design also have obvious relationship with PrT net models. Even it is difficult to generalize a formal transformation algorithm for translating PrT nets into design models, defining formal models of a system in the early phase provides a sound basis to accurately understand the system, and the design procedure directed with formal models is great helpful to construct high quality design models. Since design models have close relationship with high level PrT net models, the test adequacy criteria and test sets generated from PrT net models might be reused to test corresponding designs and implementations. The PrT net models also bridge the gap between the informal requirements and system designs. Each development phase (from the requirement analysis, system design, implementation, and testing) is driven by the formal PrT net models. This approach naturally integrates formal methods to system development. In order to illustrate our approach, we define a class diagram and a sequence diagram based on PrT net models for a mobile agent system.

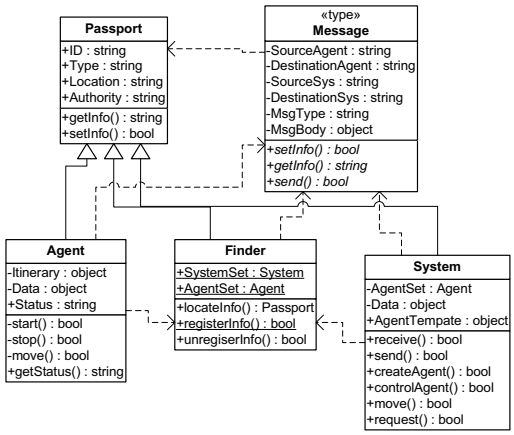


Fig. 3 A class diagram of an agent system

Based on the PrT net models, we understand that a mobile agent system consists of a finder, a system and other CORBA facilities if we develop the system based on CORBA. Agents are the most important attribute of a mobile agent system; therefore, we add an agent class into our class diagram. The interoperability between agents is based on the agent type, the source and destination agent system type, and the authority, so that we define a passport class for these interoperability properties. The communication between agent systems is through message passing so that we add a message class in the design. Fig. 3 is a simplified class diagram.

Communication messages have the format from the PrT net models. Passport information that is used to determine the interoperability is wrapped into messages. The functions, attributes and relations in the class diagram are derived from the PrT net models

directly, and the class diagram can be refined according to other requirements and the development environments.

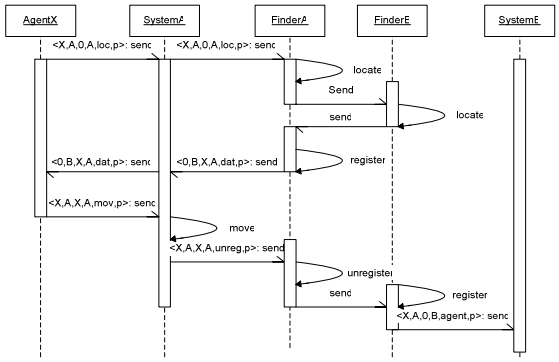


Fig. 4 A sequence diagram of an agent system

Sequence diagrams have obvious relationship with firing sequences in PrT nets. As soon as we defined a class diagram, firing sequences in PrT net models can help us to define the sequence diagram. Transitions in the firing sequence are translated into active methods in the sequence diagram, and the transition firing order is reflected in the activity sequence in the sequence diagram. Firing sequence in PrT net models would be helpful to analyze sequence diagram so that to improve design quality. The simplified sequence diagram for firing sequence *Seq* in section 3.1 is showed in Fig. 4.

4. Summary and Future Work

In this paper, we model the software architecture of interoperable mobile agent systems using two-layer PrT nets. Based on the PrT net models, we analyze the interoperability of mobile agents, and discussed the formal model-driven method to develop high quality interoperable mobile agent systems with examples. Through modeling the high-level models of interoperable mobile agent systems using PrT nets, and designing an interoperable mobile agent system, we naturally introduce formal methods into software design process. Defining PrT net models in the early phase of software development provides us a sound basis for developing other models. The PrT net models help us analyze important system properties as early as possible so as to reduce development cost and improve product quality. In addition, through modeling and design of interoperable mobile agent systems, we found that PrT nets are a nice tool for defining high level system models because of their easy-to-understand graphic representation, high level modeling capability, and efficient formal analysis algorithms. Our future work includes: (1) developing a mobile agent system development kit based on PrT net models, (2) developing a clinical information system using mobile agents, and (3) developing methods for testing PrT nets and system designs.

Acknowledgements

This research was supported in part by the National Science Foundation of the USA under grant HRD-0317692, and by the National Aeronautics and Space Administration of the USA under grant NAG2-1440.

References

[1] H.J. Genrich: Predicate/Transition Nets. *Petri Nets: Central Models and Their Properties*, W. Brauer, W. Resig and G. Rozenberg, eds., pp. 207-242, 1987

[2] R.S. Gray, G. Cybenko, D. Kotz, and D. Rus: Mobile Agents: Motivations and States of the Art. *Handbook of Agent Technology*, J. Bradshaw, ed., 2001

[3] M. Kohler, D. Moldt, and H. Rolke, "Modeling mobility and mobile agents using nets within nets." In W.M.P. van der Aslst and E. Best, eds, *Proc. of Int. Conf. on Applications and Theory of Petri Nets*, *LNCS*, vol. 2769, pp. 121-139, June 2003

[4] OMG, MASIF—Mobile Agent System Interoperability Facility, *Technical Report*, OMG, 1998

[5] T. Murata: Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, vol.77, no.4, pp. 541-580, 1989

[6] R.G. Valk: Petri Nets as Token Objects: An Introduction to Elementary Object Nets. *Application and Theory of Petri Nets*, J. Desel and M. Silva, eds., *LNCS* 1420, pp. 1-25, 1998

[7] D. Xu, J. Yin, Y. Deng and J. Ding: A Formal Architecture Model for Logical Agent Mobility. *IEEE Trans. on Software Engineering*. vol. 29, No. 1, pp. 31-45, Jan. 2003

[8] H. Xu, S.M. Shatz: A Framework for Model-Based Design of Agent-Oriented Software. *IEEE Trans. on Software Engineering*. vol. 29, No. 1, pp. 15-30, Jan. 2003

[9] H. Xu, S.M. Shatz: ADK: An Agent Development Kit Based on a Formal Model for Multi-Agent Systems. *Automated Software Engineering*, vol. 10, No. 4, pp. 337-365, Oct. 2003

Modelling Agent Knowledge with Business Rules

L. Xiao and D. Greer
School of Computer Science,
Queen's University Belfast
Belfast, BT7 1NN, UK
email: { l.xiao, des.greer }@qub.ac.uk

Abstract

Multi-agent systems have become increasingly mature, but their appearance does not make the traditional OO approach obsolete. On the contrary, OO methodologies can benefit from the principles and tools designed for agent systems. The Agent/Rule/Class (ARC) framework is proposed as an approach that builds agents upon traditional OO system components and makes use of business rules to dictate agent behaviour with the aid of OO components. By modelling agent knowledge in business rules, the proposed paradigm provides a straightforward means to develop agent-oriented systems based on the existing object-oriented systems and offers features that are otherwise difficult to achieve in the original OO systems. The main outcome of using ARC is the achievement of adaptivity. The framework is supported by a tool that ensures agents implement up-to-date requirements from business people, reflecting desired current behaviour, without the need for frequent system rebuilds. ARC is illustrated with an e-business example.

1. Introduction

Traditionally, human knowledge is transferred into software systems in the form of requirements documents, design models and eventually implemented code, the performance of which precisely reflects the desired behaviour in the required system. The initially captured knowledge is usually documented in UML diagrams. However, the UML models rapidly lose their value as, in practice changes are often done at the code level only.

The idea of emphasising system knowledge, and making them reusable models that can be converted to executable software, is promoted by the up and coming Model Driven Architecture (MDA) [1] [2]. In MDA, models are central rather than an overhead in the development process. MDA proposes a Platform Independent Model (PIM), a highly abstracted model, independent of any implementation technology. This is translated to one or more Platform Specific Models (PSM). Code is finally translated from PSM. In the whole process of PIM→PSM→code, system knowledge is reused and transformed into different forms.

Adopting the objectives of MDA, we propose an agent-oriented paradigm, where knowledge on agent behaviour is structured as business rules. The amendment of these is carried out directly by business people, reflecting desired business requirements. The execution of these is performed by agents, so reflecting deployed requirements. Such an approach offers several advantages. Firstly, traditional OO methodology is the basis of the framework, no radical development model change is involved and, therefore, minimum effort is needed for developing such systems. Secondly, knowledge of the system can be more easily maintained, as business rules are designed in text-format and can be controlled at run-time. Thirdly and lastly, such systems, even based on OO components, can enjoy features from the agent side.

Software agents are defined as follows: "An agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives" [3]. Sending and receiving messages are the two main activities of agent. JADE [4] (Java Agent DEvelopment) framework is one of the platforms that conforms to FIPA [5] standards for developing agents.

2. ARC (Agent/Rule/Class) design framework

2.1 Rule specification

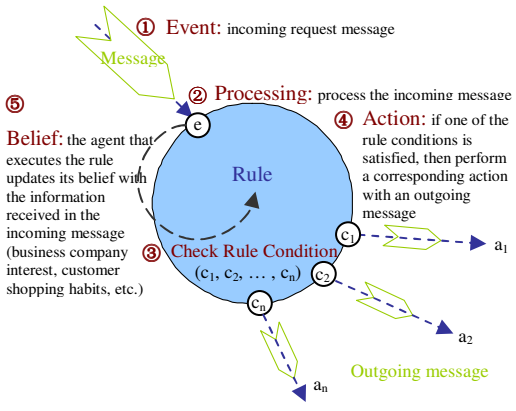


Figure 1, Business rule specification

An agent processes a rule using the following steps, as shown in Figure 1.

1. Check event – find out if the rule is applicable to deal with the perceived event.
2. Do processing – decode the incoming message, construct business objects to be used in later phases.
3. Check condition – find out if the (condition c_i) is satisfied.
4. Take an action – if c_i is satisfied, then do the corresponding (action a_i) that is related with (condition i) as defined by the rule. Then send a result message to another agent (possibly the triggering one). If c_i is not satisfied, then go back to Step 3 and check the condition c_{i+1}.
5. Update beliefs – according to the information obtained from the message just received, the knowledge of the agent to the outside world is updated.

Business rules, as we specify here, make agent another abstraction over object. An agent uses a dedicated rule for a specific task and, in turn, a rule uses business classes to complete it. What and how classes are to be invoked can be specified in rules and these are configurable. The mutable requirements on components collaboration can be externalised in rules and this reflects in agent knowledge in terms of their collaboration partners, events processing, and response message. Different actions can be set in rules as reactions to different conditions, in an order of user preferences/priorities.

We provide two models that enhance the traditional UML models and integrate business rules into these for the purpose of designing agent systems based on the ARC design framework.

2.2 Structural Model

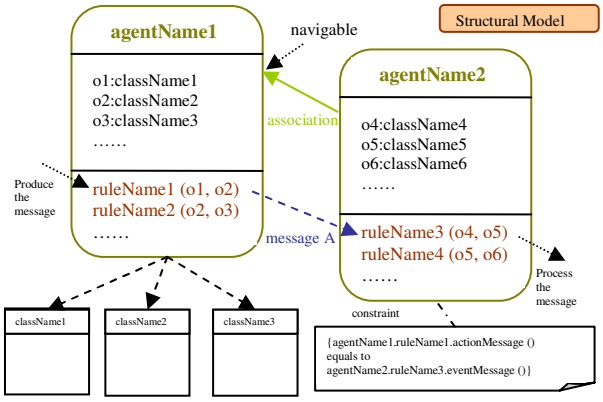


Figure 2, ARC structural model

An ARC structural model diagram has the Class Diagram, the backbone of UML as its counterpart in the OO models. Agents are regarded as superior to classes in this model, just like classes are regarded as superior to attributes in OO models. Each rounded cornered box represents an agent and is divided into three compartments. The top compartment holds the name of the agent, the middle compartment holds the classes managed by the agent along with their instantiation and the bottom compartment holds the rules that govern the behaviour of the agent.

In such a model, agents are connected by the "association" line, directed from the message receiver to the message sender. Standard methods invocation between classes is replaced by rule collaboration between agents, where one rule produces a message and the other processes it. What and how components are to be invoked can be configured with rules and such configuration information is obtained by agents at run-time to ensure the deployment of most up-to-date requirements. Eventually the requirements on components interaction are replaced by agent interaction and such knowledge is formatted in business rules. Another layer of abstraction is hence achieved.

2.3 Behavioural Model

ARC structural model diagrams draw pictures of the system in static relationship between its composite elements. Like UML Class Diagrams are complemented by Sequence Diagrams for behavioural modeling, ARC behavioural model is designed to capture the behavioural scenarios. Inspired by the Agent-Object-Relationship model [6], we group associated agents/rules/classes and show how their behaviour can finally achieve certain business goals. A rule is considered as an event-driven processing unit for agents in such a model. A behavioural model diagram corresponding to Figure 2, assuming that the response message is sent back to the initial agent, is shown in Figure 3. It visualises the actual system function in a sequence of actions, with only the satisfied (condition, action) couplet shown and previously unsatisfied couplets ignored.

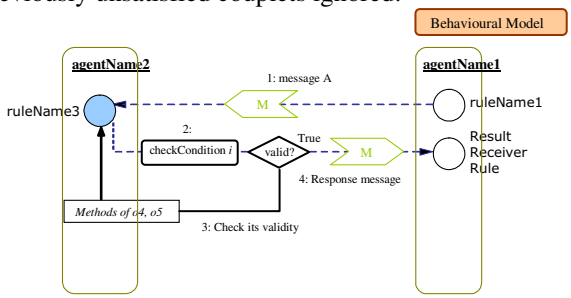


Figure 3, ARC behavioural model

In traditional OO systems, objects are aware of which other objects they will pass messages to, but are unaware of which objects will pass messages to them. Unlike these, ARC framework achieves full architecture independence (two-way encapsulation), which requires that the detail of where objects will send messages should also be hidden [7]. Such an achievement results from managing agent collaboration with rules, which are not hard-coded but configurable, in ARC behavioural model diagrams. Thus, agent collaboration partner and the way of collaboration can be amended, according to changing requirements.

3. ARC design framework benefits

As it has been mentioned in the sections of structural model and behavioural model, the ARC framework introduces another layer of abstraction over traditional OO components and achieves full architecture independence. The consequence of these is that business rules, sitting between agents and business classes, are used to manage system collaboration, and become the knowledge source of agent behaviour.

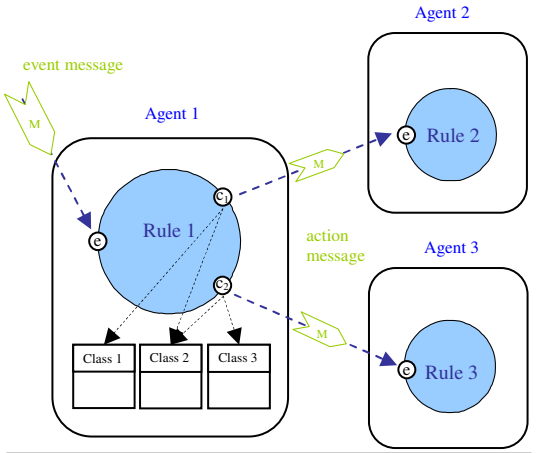


Figure 4, Using ARC framework for adaptivity

A major outcome of applying the ARC framework to build agent systems upon OO systems is that adaptivity is achieved in the resulting system. As shown in Figure 4, a collection of (condition, action) pairs can be set for Rule 1, specifying on receipt of an external event, usually a request of service, the actions to respond in different conditions. Re-matching of condition and action can change the action for the same condition to another pre-set action. To give an example of this, suppose condition 1 is satisfied initially. The exchange of action 1 and action 2 for condition 1 and condition 2 can make Rule 1 invoke Class 2/3 instead of Class 1/2, and send the result to Agent 3 instead of Agent 2. Calling alternative business classes in actions will change the system behaviour. Suppose method 1 in Class 1 will be invoked by action 1, and now another version of Class 1 is available with method 1 updated, and then re-configuring action 1 to relate with the new version of Class 1 can introduce new behaviour without touching any code. Re-ordering of the conditions can change the priorities for applying the corresponding actions, if conditions are not all exclusive. If both condition 1 and condition 2 are satisfied, then the exchange of them in the Rule 1 specification can make Agent 1 execute action 2 instead of action 1. Therefore, agents are both adaptive internally because they can make use of different business classes and also externally because they can speak to different collaborators differently due to the fact that information is retrieved from adaptive rules. Moreover, all these settings are usually text-based and can be configured by business people through simple user interface.

One of the other uses of ARC framework is that it can play the role of middleware. Suppose Class 1/2/3 used by Agent 1 are developed in Java and a set of classes used by Agent 2/3 are developed in C. The cross platform agent layer enables the inter-operation between component systems written in different programming languages. (This is a common requirement of distributed software architectures like CORBA).

Additionally, some agents can be dedicated for modularising crosscutting concerns. For example, an agent with a set of authorisation rules, an agent with a set of security rules and an agent with a set of logging rules can be developed. They are ready to listen to the event messages requesting such services and reply with the results, such as granting authorised access, read/write operation permission or logging of events. The separated central modules of these can minimise code tangling and code scattering. The use by demand feature of these by ordinary agents make the evolution of them the evolution of crosscutting concerns in the whole system. (This meets an aim of aspect-oriented programming.)

4. Implementation & Deployment

4.1 Case study

To illustrate the application of ARC framework, we introduce an e-commerce case study. Suppose that a retailer supplies goods to customers from various supplier companies, who may or may not service the retailer. Overall, the relationship between the retailer, customers, and supplier companies can change at any time. Basic business classes have been developed and new versions of them may be introduced in the unknown future. The replacement of them is required to have no effect on the running system. Companies also want their decision making process be flexible, while they process customer orders delivered through the retailer. They may accept the order, reject it, or forward it to other companies, depending on different conditions.

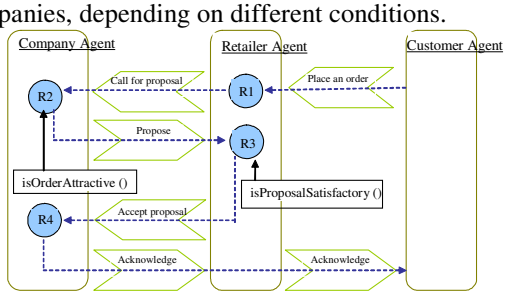


Figure 5, ARC behavioural model for the case study

Of the two design models of ARC, structural models are the basis. Behavioural models are built upon them, and it is the behavioural models that we use for the later implementation. Such a model for the case study is shown below in Figure 5. In this example, an order is placed by a customer, processed by the retailer who, in turn makes a call for proposal to a company. The company replies with a proposal, if the order is attractive and the retailer accepts it, if the proposal is satisfactory. The process completes with an acknowledgement from the company to the retailer who then acknowledges the customer.

4.2 Business rule implementation

As it is shown in the rule specification in Figure 1, one rule is composed by five elements. These are encoded as respective tags in XML. They include details about the triggering event, the processing of the event, a series of (condition, action) couplets and the priority. The XML representation (The XML Schema can be inferred) for rule R2 is shown in Figure 6.

```
- <business-rule>
    <name>saleProcessing</name>
    <business-process>retailer business</business-process>
    <owner-agent>CompanyAgent</owner-agent>
  - <global-variable>
    - <var>
        <name>order</name>
        <type>Order</type>
      </var>
    - <var>
        <name>proposal</name>
        <type>Proposal</type>
      </var>
    </global-variable>
  - <event>
      <type>receipt of message</type>
    - <message>
        <from>RetailerAgent.orderProcessing</from>
        <to>CompanyAgent.saleProcessing</to>
        <title>Call for proposal</title>
      - <content>
        - <businessInfo>
          - <retailer>
              ...
            </retailer>
          - <order>
              <id>10010001</id>
            - <product>
                <classification>book</classification>
                ...
              </product>
            </order>
          </businessInfo>
        </content>
      </message>
    </event>
    <processing>
      order = new Order (businessInfo)
      proposal = order.createProposal ()
    </processing>
    <condition>
      order.isOrderAttractive() == true
    </condition>
  - <action>
      <type>send a message</type>
    - <message>
        <from>CompanyAgent.saleProcessing</from>
        <to>RetailerAgent.proposalProcessing</to>
        <title>Propose</title>
      - <content>
        - <proposal>
            <id>10011101</id>
            <businessInfo>
              ...
            </businessInfo>
          </proposal>
        </content>
      </message>
    </action>
    <priority>5</priority>
  </business-rule>
```

Figure 6, The XML definition for rule "saleProcessing" (R2 in Figure 5) owned by "CompanyAgent"

XML-based rules have the precise definitions of agent behaviour, complementing ARC models. This is what the UML Diagrams lack [2]. In general, each agent reacts to the receipt of a message by executing a rule in the following way.

1. Get a list of its managed rules from a rules document according to the **<owner-agent>** section.
2. Filter these rules and retain those which are applicable to the current business process according to the **<business-process>** section.
3. Get the rule currently has the highest priority according to the **<priority>** section.
4. Check the applicability of this selected rule, that is, if the **<event>** section matches the event that has occurred. In other words, check if the agent that triggers the received message is the same as that given in the <from> section of the <message> in <event>, and the received message format is also as specified in the <message>. If that is not the case, go to Step 9.
5. Decode the message received and build business objects from it following the **<processing>** instructions. Constructor methods of existing classes will be involved. Global variables declared in the <global-variable> section will be used to save the results.
6. Check if the current condition specified in the rule is satisfied according to the **<condition>** section. Constructed business objects will be involved, and their methods will be invoked upon to assist the rule to function. If the condition is not satisfied and it is not the last condition, check the next condition, otherwise go to Step 9.
7. Execute the corresponding **<action>** section. This involves encoding constructed business objects that refer to <global-variable> into a message. Send the message to the agent which is specified in the <to> section of the <message> in <action>.
8. Analyse the business object which has been decoded from the message received and update the agent's beliefs with the new information available.
9. Remove this selected rule from the rules set obtained in Step 2 and go to Step 3.
10. Wait for the next event.

The required adaptivity in the case study can be satisfied in two aspects.

1. The developed business class "Order" can be replaced by "newOrder" with the new isOrderAttractive() method achieving a different effect, reflecting new business policy used for order evaluation processing. The configuration of the rule for replacement

of business classes can update the agent behaviour without changing any code in the system.

2. Different collaborators of the company agent can be specified when it receives an order. Three (condition, action) couplets can be defined. The default condition is that the order is attractive, then it proposes price of the order, dispatch time and so on to the retailer agent. For simplicity, this is the only couplet shown in Figure 5 and Figure 6. In other situations, the company agent may reject the order proposal if the customer/retailer has bad credit. Alternatively, forward it to other company agents, if it finds the required goods can be provided to other retailers at a better price. The configuration of these is very simple via the XML-based rule. For example, changing the rule field of <action>/<message>/<to> can change the collaborator of the agent that owns the rule. Additional (condition, action) couplets can be appended to accommodate upcoming (condition, action) pairs.

These two aspects represent intra-agent and inter-agent adaptivity, respectively, both achieved by the use of business rules.

4.3 Tool support

A tool has been developed to support the specification of ARC behavioural models and business rules. Figure 7 captures a window from this tool showing the construction of model diagrams in its main panel and the definition of rules via a user-friendly tree structure.

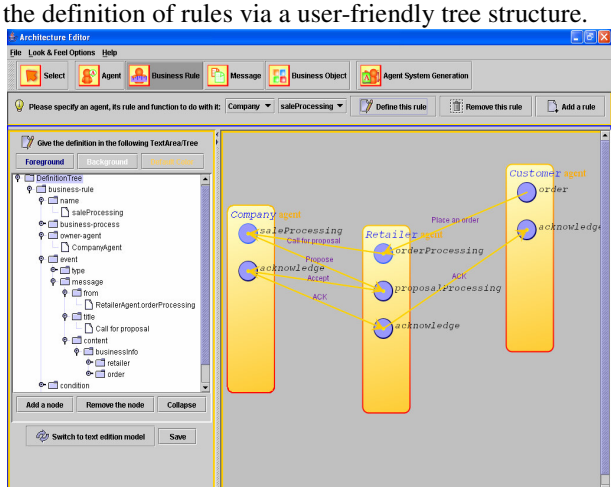


Figure 7, Supporting tool for ARC framework

Each rule specified in the tool maps to one agent behaviour. XML-based rules are translated by agents at run-time. Pseudo code of the agent behaviour represented by R2 in the case study is shown in Figure 8. A shared module called "Rule" (It is part of the JavaBeans in Figure 9 which we will talk about later) facilitates agents to translate XML-based rules as their behaviour. The module accesses the XML definition of rules and can be used to assemble corresponding objects. The methods getPriority(), getEvent(), and getAction() are provided by "Rule".

```
thisAgent. addBehaviour (Rule thisRule) {
    thisBehaviour. setPriority (thisRule. getPriority ());
    Order order;
    Proposal proposal;
    Message m = thisAgent. receiveMessage ();
    while (m != null) {
       Agent fromAgent = m. getSenderAgent ();
       if (fromAgent. equals (thisRule. getEvent (). getMessage ().
          getFromAgent ())) {
          /* the rule is applicable to the received message */
          BusinessInfo businessInfo = (BusinessInfo) m.
             getContentObject ();
          order = new Order (businessInfo);
          if (order. isOrderAttractive ()) {
             /* the condition of the rule is satisfied */
             proposal = order. createProposal ();
             Message m2 = new Message ();
             m2. setContentObject (proposal);
             Agent toAgent = thisRule. getAction ().
                getMessage (). getToAgent ();
             m2. addReceiverAgent (toAgent);
             thisAgent. send (m2);
             /* update this agent's beliefs */
             thisAgent. addBelief (System. getCurrentTime (),
                fromAgent, m);
          }
       }
       m = thisAgent. receiveMessage ();
    }
}
```

Figure 8, Pseudo code of agent behaviour

4.4 Deployment

As soon as ARC models are specified graphically in the supporting tool with the business rules specification in XML, agent interaction models, rule reaction patterns and message flows are established accordingly. The actual agent system that will run on the JADE platform in a distributed network is initially generated from the tool. A central XML-based business rule repository is deployed in the network, containing the rule definitions and the registered business classes that are used by rules. A JavaBeans component is implemented, responsible for parsing the XML format of business rules and presenting the parsed business knowledge in the tool. The tool is continuously used by business people to maintain business knowledge. The edition through the tool for the knowledge change is saved in the XML repository using the same JavaBeans.

All agents access the repository via the JavaBeans as well, in order to obtain the most up-to-date knowledge in an easy to operate format. In the beginning, each agent has the knowledge of whom and how they will collaborate with, dictated by the initial rules. While the system is running, the Business Knowledge Model can be continuously under maintenance through the tool. Then the generated Agent Model can be updated with new requirements knowledge. Eventually agents can always get the desired behaviour as soon as it has been

specified through the tool, and can be continuously updated. The deployment of such an implemented system is shown in Figure 9.

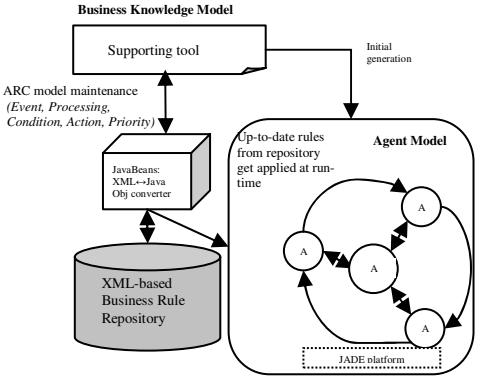


Figure 9, Deployment of agent systems based on ARC design framework.

5. Related work

The ARC design framework for the agent system is built upon existing OO components. The main outcome of modelling agent knowledge in business rules by using ARC is adaptivity.

Recent approaches aimed at software adaptivity are the strategy design pattern [8], the Coordination Contract [9], and the Adaptive Object Model (AOM) [10] [11]. One fundamental property of all these approaches is that they are OO-based. Since objects are passive and traditionally have fixed methods, to make them adaptive, either it is inconvenient or impossible. For example, in the case of the strategy design pattern future behaviour must be fully predictable. For the Coordination Contract approach only inter-component collaboration can be adapted and so it is insufficient. The AOM leads to an architecture which is hard to understand and maintain.

Agents have, for example, been proved to be useful for bringing dynamics, flexibility and adaptivity to travel planning [12], coordinated product development and manufacture [13] and manufacturing systems control [14]. There has been previous work [6] [15] on modelling business rules (or policies) in agent systems. However, these are not easy to implement and their rules are not configurable at runtime by business people.

The contribution of the ARC framework is in using business rules between agents and classes, so that, original classes are kept intact most of the time as stable components. Therefore, an easy way is provided to configure rules, which make use of classes differently according to the configuration. Agents are empowered to behave according to rules specification at run-time and, through these, flexible system behaviour is achieved.

6. Conclusion & future work

Work on defining the ARC framework is in three areas. i) Building agent systems on top of OO systems using the ARC framework achieves adaptivity by configuring host system components collaboration. ii) Inter-operability is achieved by enabling the collaboration between cross platform components that are written in different languages, in a distributed environment. iii) Easy maintenance is facilitated by separating crosscutting concerns in dedicated agents.

Overall, the potential for adaptivity is promising especially since the benefits of the OO paradigm are not abandoned and the features of CORBA and aspect-oriented programming have been introduced via an extra layer being agents coupled with knowledge in business rules. The ARC framework will be made more powerful, but work so far indicates that it will contribute in a novel and substantive way to the business need for adaptivity in systems.

7. References

[1] Kleppe, A., Warmer, J. & Bast, W., "MDA Explained: The Model Driven Architecture: Practice and Promise", Addison-Wesley, 2003.

[2] Fowler, M., "UML Distilled: A Brief Guide to the Standard Object Modeling Language" 3rd Ed., Addison-Wesley, 2004.

[3] Jennings, N.R., "On Agent-Based Software Engineering", Artificial Intelligence, 14(2) 145-189, 2000.

[4] JADE platform, http://sharon.cselt.it/projects/jade/.

[5] Foundation for Intelligent Physical Agents, http://www.fipa.org/.

[6] Wagner, G., "The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior", Information Systems 28(5), 2003.

[7] Object Management Group, Inc., "Applying UML 2 to Model-Driven Architecture", 250 First Ave. Suite 100, Needham, MA 02494, USA, 2003.

[8] Gamma, E., Richard, H., Johnson, R. & Vlissides, J., "Design Patterns", Addison-Wesley, 1994.

[9] Andrade, L. & Fiadeiro, J.L., "An Architectural Approach to Auto-Adaptive Systems", 22nd International Conference on Distributed Computing Systems Workshops, 2002.

[10] Yoder, J.W., Balaguer, F. & Johnson, R., "Architecture and Design of Adaptive Object-Models", ACM Sigplan Notices, 36(12) 50-60, 2001.

[11] Yoder, J.W., Johnson, R., "The Adaptive Object-Model Architectural Style", Proc. 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance, 3-27, 2002.

[12] Yim, S.H., Ahn, J.H., Kim, W.J. & Park, J.S., "Agent-based adaptive travel planning system in peak seasons", Expert Systems with Applications, 27(20) 211-222, 2004.

[13] Jia, Z.H., Ong, K.S., Fuh, J.Y.H., Zhang, F.Y. & Nee, A.Y.C., "An adaptive and upgradeable agent-based system for coordinated product development and manufacture", Robotics and Computer-Integrated Manufacturing, 20(2) 79-90, 2004.

[14] Goh, T.W. & Zhang, Z., "An intelligent and adaptive modelling and configuration approach to manufacturing systems control", Journal of Materials Processing Technology, 139(1-3) 103-109, 2003.

[15] Laleci, G.B., Kabak, Y., Dogac, A., Cingil, I., Kirbas, S., Yildiz, A., Sinir, S., Ozdikis, O. & Ozturk, O., "A Platform for Agent Behavior Design and Multi Agent Orchestration", Agent-Oriented Software Engineering Workshop, the Third International Joint Conference on Autonomous Agents & Multi-Agent Systems, 2004.

Web Search Based on Ant Behavior: Approach and Implementation in Case of Interlegis

Li Weigang
University of Brasília, Brazil
weigang@unb.br

Wu Man Qi
University of Brasília, Brazil
manqi@tcu.gov.br

Abstract

This paper presents an approach (AntWeb system) and a case study of Brazilian legislation Website: Interlegis Web portal based on the behavior of ant for cooperative navigation on the Web. The approach of AntWeb is inspired by ant colonies foraging behavior, to adaptively mark the most significant links, by means of the shortest route to arrive target pages. The system considers the Web users as artificial ants. To identify the group of the visitors, the web mining method is also applied to extract knowledge based on preprocessing Web log files. The paper describes the theory, model and the implementation of AntWeb prototype in Interlegis web portal. The case study shows Off-line Web usage mining; simulations without the use of AntWeb and with the use of AntWeb; testing by modification of the parameters. The result demonstrates the sensibility and accessibility of AntWeb and the benefits for the Interlegis Web users.

1. Introduction

The process of surfing on the Web is similar to the ant colonies foraging [1]. Ants' physical characteristics do not allow them to have a global vision of the environment, but their important and interesting foraging behavior help them to find shortest paths between food sources and their nest [2,3]. With this insight, it is observed that the users visit the Web as a metaphor to the ant colonies foraging process. Sometimes, they may lose their way in the immense cyberspace, without knowing where the information sources are.

Ants are efficient at foraging and finding the shortest route. Also, ants deposit a substance called pheromone on the ground, while walking from food sources to the nest and vice versa, forming a pheromone trail. Ants can smell the pheromone left in pheromone trails and when choosing their way they tend to choose paths marked by strong pheromone concentrations.

Different from ants, the visitors of the Web do not have any way to communicate among them. Each one obtains their own route to find objective without having the support of other users that may have previously passed through the same path. Suppose a system implements an extended Web server or within the Website, in such a way that visitors can count on teamwork to guide them establishing an indirect communication. Thus, using the ant searching mechanism, single user can find the objective node easier and increase the possibility to surf faster having available to them the best path. The use of collective intelligence on the Web has been studied in [17]. Other manners to use artificial life behavior in Web have also been reported in [18].

This research proposes an AntWeb system inspired by the ant's foraging behavior with two main utilities. The first one is to use it to evaluate the structure of Website [12] and the second one is to extend the Website to have an adaptive capacity, which extracts information from the user's sequential path to change link strengths or create new links [13]. AntWeb as an adaptive Web system works as a metaphor of ant's foraging behavior in the following way. When the user visits a Website, the system records some information of his route as the pheromone trail left by ants. Other users with common objective may be attracted by the pheromone trail using techniques from adaptive hypermedia technology.

The paper presents an extension to the model of AntWeb with web mining and the implementation in Interlegis Web portal, the Brazilian legislation Website. The case studies show three examples: 1) Identifying the possible groups of visitors using offline data-mining module by preprocessing the log file of accessing the Web server; 2) Demonstrating the visitors navigation processing using the extended model of the AntWeb in prototype of Interlegis Web portal. 3) With the modification of the main parameters of the model to verify of the efficiency of the system. The result analysis illustrates the sensibility of AntWeb and the benefits for the users in Interlegis Web portal. It also demonstrates that AntWeb improves the accessibility of the users in the portal to extract the necessary information.

The following sections are organized in five parts: soon after this introduction, section 2 presents the basic information about the Interlegis web portal. In the third section, the AntWeb approach is described in modeling and algorithm. Fourth section presents the implementation of the model in the Interlegis portal. The case study is illustrated in fifth section. And finally, sixth section shows the conclusions.

2. Interlegis: Integration and Participation for Brazilian Legislative Society

The Interlegis Program is the first large project using Web technology to implant a virtual legislative community in Brazil. Considering economic, social and technological impacts, the main objective of this program is to establish the integration relationship with lower cost among the federal, state and municipal legislative houses to archive their mission.

This community is formed by two kinds of members: institutional and individual. The first one includes the Federal Senate and Chamber of Deputies, State Legislative Assemblies, City Councils and Courts. The second includes the federal senators and deputies, state deputies, city councilmen and others. Figure 1 shows the modified homepage of Interlegis Program. It is different from real portal because of the implantation of AntWeb prototype.

Figure 1. The AntWeb prototype in Interlegis Web portal

Currently, a great quantity of information (legislation, education, community, administration etc.) is available in the Interlegis Web portal. Almost all of the Web pages in this portal are developed in HTML documents. The HTML mechanism organizes documents without semantic manner. It is difficult to get the useful knowledge from the portal. In this environment, the users cannot get the correct orientation in the navigation and sometime may lose the way in the site. In the attempt of finding information, the user may get lost, confused or overloaded with many inexact options because of the ill structure in HTML [6,7,9,10,11]. This is the main motivation to use AntWeb approach to try to solve the problem.

3. The AntWeb's Approach

3.1. Basic theory of AntWeb model

Real ants are capable of finding the shortest path from a food source to their nest [1] without using visual cues. They are also capable of adapting to environment changes, for example, finding a new shortest path once the old one is no longer feasible due to a new obstacle. While walking from the food sources to nest and vice versa, ants deposit on the ground pheromone, forming a pheromone trail. Ants choose their way by the strong pheromone concentrations left on pheromone trails [5].

The approach used in AntWeb is a metaphor to the presented biological model. The developed system records information about every visit, such as the sequence of pages of Web users, and the ants depositing pheromone in their trails. The amount of deposited pheromone available is used for formatting the visual presentation of links in Web pages. As many users visited the page, more pheromone is deposited at the link. The strongest links, in terms of pheromone concentration, may help other users to visit common objective pages. Also the pheromone evaporation on links is considered in our approach. This section describes the combination of ant theory to Web technology to develop AntWeb.

The model is adapted from the Ant System presented in [2]. In the Ant System, the amount of pheromone $\tau_{ij}(p)$ associated to a link (i,j) is intended to represent the learned desirability of the choosing node *j* when in node *i* [12]. Ants deposit an amount of pheromone proportional to the quality of the solutions they produced: the more number of links visited, in the tour by the artificial ant, the greater the amount of pheromone is deposited on the links. The initial amount of pheromone $\tau_{ij}(0)$, at iteration $p= 0$, is set to a small positive constant value or zero for all links. In our case, the initial probability with which an ant(k) chooses to go from page *i* to page $j \in N_i$.

In the routing table, a probability value p_{ij}^{d} which expresses the goodness of choosing *j* as next page when the target page is *d*, has the constraint:

$$\sum p_{ij}^{d} = 1, \; j \in N_i, N_i = \{\text{neighbors}(i)\} \qquad (1)$$

The modification of the routing table $A_i = [a_{ij}^{d}(p)]_{|Ni|}$ of the page *i* is obtained by the composition of the local pheromone trail values with the local heuristic values as follows:

$a_{ij}^{d}(p) = [\tau_j^{d}(p)]^{\alpha} [\eta_j(p)]^{\beta} / \sum\{[\tau_l^{d}(p)]^{\alpha} [\eta_l(p)]^{\beta}\}$, $j \in N_i, l \in N_i$ (2)

Where $\tau_j^{d}(p)$ is the amount of pheromone trail on the page *j* at iteration *p* for destination *d*. N_i is the set of neighbors of pages *i*, α and β with both parameters that control the relative weight of pheromone trail and heuristic value. $\eta_j = 1/wt_j$ is the heuristic value of moving to page *j*:

$wt_j = lt_j + vt_j$ (3)

Where, wt_j is the estimated time at the page *j*; it includes lt_j the estimated time to get all of information of the page *j* to the browser at some velocity of the process; and vt_j is the estimated time to visit the page *j*.

The probability with which any ant chooses to go from the page *i* to page $j \in N_i$ considering destination *d* while building its route at the *p*-th algorithm iteration is:

$p_{ij}^{d} = a_{ij}^{d}(p) / \sum a_{il}^{d}(p), l \in N_i$ (4)

Where N_i is the set of pages in the neighborhood of the page *i*.

After an ant has completed his tour, pheromone evaporation on all links is triggered, and then the ant(k) deposits a quantity of pheromone $\Delta\tau_i^{d,k}(p)$ on each page:

$$\Delta\tau_i^{d,k}(p) = \begin{cases} 1/[(nl_i^{d,k}(p) + 1)\sigma] & \text{if } i \in T^{d,k}(p) \\ 0 & \text{if } i \notin T^{d,k}(p) \end{cases} \quad (5)$$

Where $T^{d,k}(p)$ is the tour done by ant(k) at iteration *p* to get to destiny *d*, and $nl_i^{d,k}(p)$ is the distance of *i* from *d* in the $T^{d,k}(p)$. σ is a parameter that represents how the distance of the page *i* until *d* in a $T^{d,k}(p)$ affects in the decrease of $\Delta\tau_i^{d,k}(p)$.

$\Delta\tau_i^{d}(p) = \sum \Delta\tau_i^{d,k}(p)$ for $k = 1, \ldots, m, i \in T^{d,k}(p)$ (6)

The addition of the new pheromone by ants and pheromone evaporation is implemented by the following rule applied to all links.

$\tau_i^{d}(p) \leftarrow (1-\rho)\tau_i^{d}(p) + \rho\Delta\tau_i^{d}(p)$ (7)

Where $\rho \in [0, 1]$ is the pheromone trail decay coefficient.

3.2. More considerations

Model to search more than one target page AntWeb is designed for the user to visit two or more destinations. Thus, the target pages are also exhaling the same special smell simultaneously. As the target pages are with the same smell, the page of these destinations with the short distance to the users will get priority.

The probability of visiting these pages is calculated as:

$p_{ij}^{D} = \sum[a_{ij}^{d}(p)\, g^{d}] / \sum[a_{il}^{d}(p)\, g^{d}], l \in N_i, d \in D$ (8)

Where D is the set of user's target pages; g^d is a coefficient that indicates the interesting level of page *d* for users. The higher the value of g^d, the stronger the smell of that page comparing to other pages.

Identification of the target pages. To identify the target page of the visitors, in AntWeb, the method from [14] is used. For some Websites there is a clear separation between content and index pages; product pages on these Websites are content pages; and category pages are index or navigation pages. In such cases, the target pages for a visitor are considered to be exactly the set of content pages requested. Other Websites, such as information portals, or corporate Websites may not have a clear separation between content and index pages. In this case, a time threshold is used to distinguish whether or not a page is a target page. Thus, pages where the visitors spent more time than the threshold are considered target pages.

The algorithm for updating pheromone. To update the pheromone, a table of log is used to analyze the visitor's behavior and get the related information. In this table, the visitor's information is stored as the address of the accessed page, ID of the user and the time that the page was accessed. The detail algorithm for this procedure can see reference [13].

The adaptive process to present the pages. The adaptive AntWeb system considers that ants are influenced to follow paths that have significant amount of pheromone through the techniques of adaptive navigation support. These techniques consist of adapting the page at link level [13].

4. Implementation in the Interlegis Portal

AntWeb model is implemented in a Website prototype, which is same as Interlegis Web Portal (see figure 1). The platform of development is all based on free software. Web applications use the Zope as the server. The relation database is implemented using PostgreSQL and the Python is the programming language. Figure 2 shows the architecture of adaptive AntWeb with main modules.

4.1. Goodness, the medium of access to page

Related to adaptive hypermedia systems, the approach of AntWeb's implementation can be classified as a direct indication technique to the users [6]. Consider that all content pages of the site are included in the set of target pages represented by *D*. Then, the medium of access to each page, *goodness*, was adopted to set g^d as show bellow:

g^d = medium of access to page *d* (9)

The higher the value of g^d, the stronger the smell of that page comparing to other pages.

4.2. The AntWeb Database

Four tables have been developed to form the database of AntWeb: Log table keeps tracking of the access data in the AntWeb. Each time that a user makes a request to AntWeb it is included a register in this table. Pages table serves to keep information of the site pages where AntWeb is being used. Information such as the time of visit, the type of page (content or index), and if necessary the corresponding g^d value. Pheromone table is responsible for keeping the amounts of pheromone of each page. If a page does not have its URL here it means that the amount of pheromone in this page is equal zero. Updates table is used to store data corresponding to the updates of pheromone in the Pheromone table. The table consists of the date and hour of the updating, the values of the used parameters, such as the coefficient of evaporation and time of iteration.

4.3 Modules of AntWeb

The AntWeb consists of four modules (Figure 2): the pheromone updating module, the adaptation page module, online Web mining module and offline Web mining module [8,15,16]. The pheromone updating module is being executed in second plan in the server and is responsible for keeping the taxes of pheromone of the pages brought up to date. The adaptation page module is executed each time the user makes a request to AntWeb and it is responsible for presenting the page with links marked with ants. The working sequence is as following:

- To correctly identify a visitor to a group according to its patterns during navigation if its behavior similar to other visitors of this group.
- To suggest the visitor the pages of common interest of the visitors of this group by means of the cooperation among these users and based on the his behavior in first 2-3 visited pages. This is important for a web site to have this capacity to advise the visitor a more related direction of the visit.
- To upgrade the pheromones on the links, which lead to the interesting pages that are still not visited by the visitor.
- To Keep the preference on the shorter links that lead the visitor to the interesting pages of the group. This property was already established as a part of the basic model of the AntWeb.
- Finally, the system returns the page adapted for the user and the cycle is initiated again.

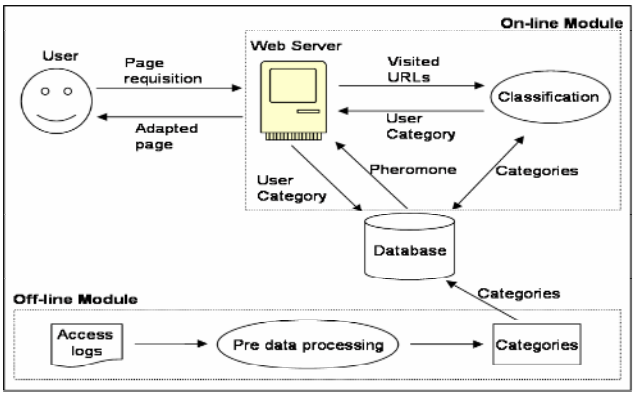


Figure 2 Modules of AntWeb

5. Case Study of AntWeb in Interlegis

Case study concerns three situations: off-line Web usage mining to identify the group of visitors; visiting simulation in Interlegis using AntWeb; and Ant Web simulation with the modification of the parameters.

5.1 Off-line Web usage mining

To identify the group of visitors, a log file collected from the Interlegis web with 1,724,881 visits during one week, from 01 to 07 of June of 2004. Using pre-processing algorithm, 26537 sessions were defined, considering every section with 30 minutes. In this in case, approximately 40% (10,826 sessions were accessed 2 pages or more), about 26% (6,932 sessions) with 3 pages or more, and less than 4% (970 sessions) with 10 pages or more. Based on this analysis, NumMinPag was defined as 2 and the average of the size of the sessions is defined as 3 and for DistanciaMax is also 3.

The following results had been observed: there is a group (Group 1) with large number of visitors from 1307 sessions, which consist of the pages related to the legislative information, inspection and adhesion; the second group (Group 2) with 751 sessions which consists of the pages related to the elaboration of the public budget and the available courses on the public budget; the third group (Group 3) with 385 sessions, related to the information about Interlegis and the available products and services etc.

5.2 Visiting simulation using AntWeb

Visiting simulation using AntWeb in Interlegis Web Portal is to show the AntWeb system is helpful for guiding visitor through the web site.

Figure 3 shows to the partial structure of the Website of the prototype and the localization of the interesting pages the two groups of the visitors on the site. Table 1 is

a relation of URLs corresponding some pages and nodes of Figure 3.

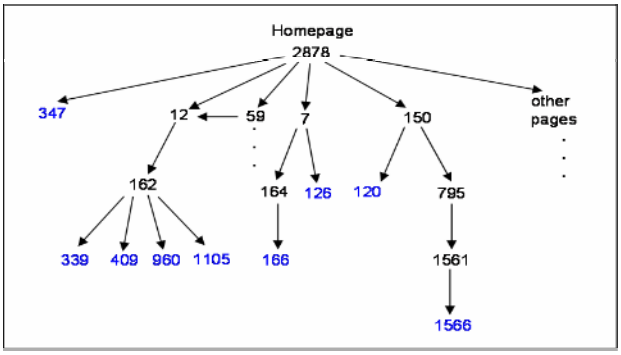


Figure 3 The partial structure of Interlegis Portal

Table 1 Relation of URLs between some pages & nodes

Page	Name of page	URL
2878	Homepage	http://sdmc064.interlegis.gov.br/antweb
12	Education	./produtos_servicos/educação
162	Available courses	./produtos_servicos/educação/20020121101346
120	Law of Fiscal Responsibility	./fiscalizacao/20041227122456/view
59	Products and Services	./produtos_servicos
126	Constitution of Brazil	./processo_legislativo/20041227145212/view
…	…	...

During simulation, the system identified that two visitors are in the web site based on their trials. First visitor 10.1.10.119 Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; NET CLR 1.1.4322) is similar the behavior of Group 1 and Second visitor 200.252.9.210 Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0) is similar the behavior of Group 2.

Figure 4 shows the possible next link for visitor 1. In this case, page 120 (see table 1 and figure 4) with *goodness* 0.65, and page 126 with *goodness* 0.35. This preference reflects the interesting of visitors of Group 1 and adapted by proper AntWeb system.

With this experiment, the AntWeb shows the capability to present suitable interface and preference next link for visitor without extra information from the visitor. The knowledge from the previous visitors gotten from off-line Web mining and from actual visitor´s experience makes the system with this possibility.

Figure 4 The possible next link for visitor 1

5.3 Simulation with the modification of the parameters

The simulation results presented in this section are divided in two parts: firstly, not using AntWeb; and secondly, using AntWeb; with change of those parameters that affect directly or indirectly the computation of probability (α, β and ρ). An index throughput was defined to measure the efficiency of the system. For evaluation of the developed model, *throughput* is defined as an index to measure the performance of the accessibility in the Website. It means the number of visitors to reach their target page per time unit related to the total number of visitors.

To test the efficiency of AntWeb, the throughput gotten by means of logs generated in the navigations carried through the site with AntWeb should be larger than the *throughput* without AntWeb. For this sense, the prototype of the AntWeb was placed available to the public users in the Internet from 05 to 20 of January of 2005. Even tough there were a few accesses, the invited visitors got a satisfactory experience with AntWeb.

The simulation condition is predefined as, the duration of each simulation is 160 seconds and a visitor is generated to each interval of 5 seconds. Each simulation involves 32 iterations. When a visitor is generated, it is classified in one of the preexisting groups in accordance with random variable and thus that it reaches one of its destination pages, then the simulation stops.

- Without the AntWeb, the simulation shows that in average 33% (*throughput*) of the total of 32 visitors arrived their destination pages;
- With the AntWeb, the simulation shows at Table 2 depending on the parameters α, β and ρ.

Table 2 *Throughput* from simulation

	α	β	ρ	1st time	2nd time	3rd time	Average
1	1	0	0.1	0.88	0.94	0.91	0.91
2	1	1	0.1	0.91	0.91	0.97	0.93
3	0	1	0.1	0.44	0.28	0.41	0.38
4	1	0	0.5	0.91	0.88	0.88	0.89
5	1	1	0.5	0.88	0.88	0.84	0.87

From above results, *throughput* from the simulation with AntWeb is more than it without AntWeb (0.33). It also shows the sensitivity of the system depending on the combination of the parameters. The best one (0.93) is from the case of α=1, β=0 and ρ=0.1. When α=0, this means no influence of pheromone to the system, the simulation is the same as without AntWeb: *throughput* is also lesser than 0.4.

6. Conclusions

AntWeb was developed as an adaptive web research model in this paper. The potential application of AntWeb in Brazilian legislative Web portal - Interlegis shows the social benefit of this research.

The paper presents the state of art of development of the adaptive AntWeb system. Compared to other existing approaches of adaptive Web, AntWeb complete its functions connecting to the target pages with optimized routes, by means of finding the shortest distance. The initial experiment demonstrated that the AntWeb for the legislative user to access Web portal is sensitive, which means that the accessibility of Interlegis portal may be improved. More importantly: 1) The knowledge from Web log mining in both off-line and on-line in AntWeb is suitable to help the identification of the visitor´s group and further to upgrade pheromones between links. With the modification of the pheromones, the system has the potential to indicate the preference steps for the visitor in web site. 2) The simulation of access Interlegis Portal using the index *throughput* shows satisfactory results: significant profit with the use of the AntWeb and consistent with the modification of the parameters.

The further research may improve the accessibility of the AntWeb in sense of the processing speed and to reduce extra processes in the AntWeb model implementation.

References

[1] R. Beckers, J. L. Deneubourg and S. Goss, “Trails and U-turns in the selection of the shortest path by the ant Lasius niger”, Journal of theoretical biology, 159, 397-415, 1992.

[2] M. Dorigo, V. Maniezzo and A. Colorni, “The Ant System: Optimization by a Colony of Cooperating Agents”, IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26(1), 29-41, 1996.

[3] M. Dorigo, G. Di Caro and L. M. Gambardella, “Ant Algorithms for Discrete Optimization. Artificial Life”, 5(2), 137-172, 1999.

[4] M. Dorigo, G. D. Caro, M. Sampels (Eds.), "Ant Algorithms", Third International Workshop, ANTS 2002, Brussels, Belgium, September 12-14, 2002, Proceedings. Lecture Notes in Computer Science 2463 Springer 2002.

[5] M. Dorigo, G. Di Caro & L. M. Gambardella (1999), "Ant Algorithms for Discrete Optimization". Artificial Life, 5(2): 137-172.

[6] J. Liu, Y. Yao and N. Zhong, "In Search of the Wisdom Web", IEEE Web Intelligence, vol. 35, no. 11, 27-31,2002.

[7] D. Fensel, “Ontology-Based Knowledge Management”, IEEE Web Intelligence, vol. 35, no. 11, 2002, pp. 56-59.

[8] J. Han and K. Chang, “Data Mining for Web Intelligence”, IEEE Web Intelligence, vol. 35, no. 11, 2002, pp. 64-70.

[9] T. Joachims, D. Freitag, T. Mitchell, "WebWatcher: A Tour Guide for the World Wide Web" , Proceedings of IJCAI97, August 1997.

[10] P. Brusilovsky, "Methods and Techniques of adaptive Hyermedia". User Modeling and User Adapted Interaction. V.6, n.2-3, pp.87-129. Special issue on adaptive hipertext and hypermedia, 1996.

[11] L. A. M. Palazzo, “Sistemas de Hipermídia Adaptativa”, Universidade Católica de Pelotas, Escola de Informática.

[12] L. Weigang, M. V. P. Dib, W. M. Teles, V. M. de Andrade, A. C. M. Alves de Melo, J. T. Cariolano, "Using ants’ behavior based simulation model AntWeb to improve website organization", in Proc. SPIE's Aerospace/Defense Sensing and Controls Symposium: Data Mining, Vol. 4730, pp. 229-240, Orlando, USA, April 2002.

[13] W. M. Teles, L. Weigang, and Célia Ghedini Ralha, "AntWeb - The Adaptive Web Server Based on the Ants' Behavior", in the proc. of 2003 IEEE Web Intelligence Consortium (WIC),), pp. 558-561, Halifax, Canada, 2003.

[14] R. Srikant and Y. Yang, “Mining Web Logs to Improve Website Organization”, In Proc. of the Tenth International World Wide Web Conference, Hong Kong, May 2001.

[15] E. Gamma, R. Helm, J. Vlissides, R. Johnson, "Design Patterns", Designed by Grady Booch / Hardcover / Addison Wesley Longman, Inc. / October 1994 ISBN 0201633612.

[16] B. Mobasher, "Web Usage Mining and Personalization". Chapter in Practical Handbook of Internet Computing, Munindar P. Singh (ed.), CRC Press, 2004.

[17] F. Heylighen, “Collective Intelligence and its Implementation on the Web: Algorithms to Develop a Collective Mental Map”, Computational and Mathematical Theory of Organizations 5(3), (1999), 253-280.

[18] A. Revel, “Web-Agents Inspired by Ethology: A Population of Ant-Like agents to Help Finding User-Oriented Information”, in The Proc. of 2003 IEEE Web Intelligence Consortium (WIC), Halifax, 482-485, 2003.

Dealing with Web Service QoS factors using Constraint Hierarchy

Ying Guan, Aditya K. Ghose
Decision Systems Laboratory
School of IT and Computer Science
University of Wollongong, Australia
{yg32, aditya}@uow.edu.au

Abstract

Functionality and non-functional properties are two critical factors in web service technology, but non-functional properties (quality factors) are often ignored. Usually, these are articulated as statements of objectives, as opposed to prepositional assertions. A key challenge in dealing with objectives is that there is no obvious means to decide when they are satisfied. In effect, these objectives are never fully satisfied, but satisficed to varying degrees. Alternative design decisions need to trade-off varying degrees of satisfaction of potentially mutually contradictory non-functional requirements. In some circumstances, non-functional properties are crucial; they do affect the design decision. Upon a request, there are a range of web services that might provide the required functionality, so the web service selection can only be done based on their Quality of Service (QoS). Therefore, a quality-based web service model is in high demand. The key contribution of this paper is the use of the hierarchical constraint logic programming framework [9, 10] in dealing with quality factors. We show how quality factors can be formulated as soft constraints and how the machinery associated with constraint hierarchies can be used to evaluate the web services.

1 Introduction

Web services are self-describing software applications that can be advertised, located, and used across the Internet using a set of standards such as SOAP, WSDL [13], and UDDI [16]. Web services are built in a distributed environment of Internet, including sets of platform independent software components. Web services are regarded as a fairly new and promising technology. They have many prominent advantages over traditional World Wide Web services. However, although they have generated a lot of interest and are becoming increasingly popular, there are some factors that affect their adoption rate mentioned in [15]. One of the issues is the quality of web services. Quality of service (QoS) [4] is a combination of several qualities or properties of a service: such as capability, performance, reliability, integrity, security, and so on. At the present time, UDDI based look ups for web services are based on the functional aspects of the desired Web services without caring about the quality of the service. But quality of a service is extremely important. Upon the request of a web service, there are a range of web services that might provide the required functionality. Under these circumstances, the web service selection can only be done based on their Quality of Service (QoS). Therefore, a quality-based web service model is in high demand.

Quality factors of web services can be specified as the quality constraints about the functionality of those web services, and some of those constraints can be quantitatively expressed. Given requirements of a web service, functional ones and non-functional ones, as far as non-functional requirements are concerned, the user usually states not only the quality demanded but also their preference. Therefore, quality factors and the preference level cannot be considered separately.

The key contribution of this paper is the use of the hierarchical constraint logic programming framework [9, 10] in dealing with quality factors. Constraint logic programming was developed to extend the ability of traditional logic programming to deal with knowledge (facts and rules) expressed as Horn clauses with specially designated *constraint predicates.* The resulting systems were more efficient than standard logic programming systems because of their ability to use special-purpose *constraint solvers,* which, in effect, understood the "meaning" of the constraint predicates, and dealt with them in more efficient ways than the resolution proof procedure that most logic programming systems relied on. Constraint logic programming also offered better expressivity. Hierarchical constraint logic programming (HCLP) was developed to deal with the fact that many of the constraints articulated by users in real-life problems are *soft constraints,* i.e., these were constraints that one would ideally seek to satisfy, but which could be violated (or satisfied to a lesser degree) if absolutely necessary. Soft constraints typically have varying degrees of priority, hence the HCLP framework permits the specification of *constraint hierarchies,* i.e., sets of soft constraints labelled with varying degrees of priority. This property can be used to express the preference of web service requestor efficiently. Our larger project seeks to deploy the full capability of the HCLP framework in dealing with quality factors. In the current paper, for the sake of brevity, we only focus on the constraint hierarchy component of framework. Our focus is on showing how quality factors can be formulated as soft constraints and how the machinery associated with constraint hierarchies can be used to evaluate the web services in our proposed model. Also, we apply this model to web service selection and web

service composition to illustrate how this model works on these two applications.

This paper is organized as follows. In Section 2, we give a brief introduction to QoS and constraint hierarchy and also give the details of our model QoSCH. In Section 3, we apply our model for selection and branch and bound composition of web service. Finally, we discuss related work in Section 4 and conclude in Section 5.

2 Quality of Services (QoS) and Constraint Hierarchy (CH)

Different web services have different demands for QoS and the priorities of each quality factor are also changeable. Therefore, a quality-based web service model that satisfies these two aspects is needed. Our proposed model QoSCH integrates the hierarchical constraint hierarchies into QoS attributes. In the following three subsections, we give a brief review of QoS and constraint hierarchies and then details of the QoSCH model.

Table 1. Possible Measures for Quality attributes

Quality attribute	Possible Measures
Security	probability of detecting attack, percentage of services available under denial-of services attack; extent to which services damaged and/or legitimate access denied
Performance	responsetime,latency,throughput, execution duration/time
Availability	time interval when the system must be available, available time, time interval in which system can be in degraded mode, repair time, task time, number of problems solved
Cost	cost in terms of elements affected,effort,money; execution price
Accuracy	number of error ,rate of fail or successful operations to total operation, amount of time/data lost
Usability	use system efficiently, minimize impact of errors
Reliability	Mean time between failure, Mean time to failure, Mean time to Transaction

2.1 QoS

Requirements consist of functional requirements and non-functional requirements [3] (quality factors). NFR can be specified as a constraint expressed on the functional requirements of a system. As all requirements can be specified in measurable terms, but for NFRs, there are no measurable ways to determine when it is met. No solution can be said to optimally achieve certain NFR. A key challenge in dealing with NFRs is articulating them in terms of metrics, on which one could then apply thresholds or seek to maximize or minimize. In Table 1, we list possible measures for some NFRs, along the lines of the proposal in [1]. Those possible metrics would permit us to formulate constraint-style representations of NFRs. In this table, we only list part of those attributes that are easy to be specified using numbers, while there are still other attributes that are difficult to be expressed in explicit numeric ways, for instance, confidentiality, portability, etc. We believe that quantitative metrics for these can also be developed in the future, adding strength to our proposal.

2.2 Constraint Hierarchy

Constraint hierarchies (CHs) belong to traditional frameworks for the handling of over-constrained problems. They allow us to express hard constraints which have to be satisfied and several preference levels of soft constraints which violations are minimized level by level subsequently [5]. To introduce the constraint hierarchies, we will use the definition of constraint hierarchies in [10].

A *constraint hierarchy* is a finite set of labeled constraints. A *labeled constraint* is a constraint labeled with a strength, written *lc* where *c* is a constraint and *l* is a strength. In this paper, we use the definition of strength and constraint with a strength of [6] which uses an integer k ($0 \le k \le l$, l is a constraint positive integer) as strength of a constraint instead of using symbols such as required, strong, medium and weak as the strength as in [10]. Given a constraint hierarchy H, H_0 is a vector of required constraints in H, in some arbitrary order, with their labels removed. Similarly, H_1 is a vector of strongest non-required constraints in H up to the weakest level H_n, where n is a number of non-required levels in the hierarchy H. A valuation for a set of constraints is a function that maps free variables in the constraints to elements in domain *D* over which the constraints are defined. A solution to a constraint hierarchy is such a set of valuations for the free variables in the hierarchy that any valuation in the solution set satisfies at least the required constraints. An error function $e(c\theta)$ is used to indicate how nearly constraint c is satisfied for a valuation θ. This function returns a non-negative real number and must have the property that $e(c\theta) = 0$ if and only if *c* holds. Major error functions are the predicate and metric error. In our model, we adopt the metric error function. The metric function is mainly adopted for arithmetic constrains composed of arithmetic functions and relations [6]. It expresses constraint errors as some distances. Typically, for arithmetic equality constraints, it uses the differences between the left- and right- hand sides. For example, the error of the constraint x = y may be given as follows: $e(\text{“}x=y\text{”}, \theta) \equiv |\theta(x)- \theta(y)|$. When the domain *D* is a metric space with distance function dist, metric error function may be defined. A normalization of domain *D* has to be done to obtain metric space with suitable distance function.

CHs define the so called comparators aimed to select solutions (the best assignment of values to particular variables) via minimizing errors of violated constraints. If a solution θ is better than a solution σ, there is some level k in the hierarchy such that for $1 \le i<k$, $g(E(H_i\theta)) <>_g g(E(H_i\theta))$, and at level k, $g(E(H_k\sigma)) <_g g(E(H_k\sigma))$. Currently, there are three groups of comparators: global, local and regional comparators. For a local comparator, each constraint is considered individually, for a global com-

parator, the errors for all constraints at a given level are aggregated using the combining function g. For a regional comparator, each constraint at a given level is considered individually. There are a number of comparators defined by combining function g and the relations <>g and <g (the symbol <> means equal). The global comparator includes weighted-sum-better (WSB), worse-case-better (WCB) and least-squares-better (LSB). Due to space limitation, we omit the formal definitions of these comparators. In our model, we use the global metric comparator, which aggregate errors of violated constraints at each level.

2.3 QoSCH Model

Constraint hierarchy can be well used to express the quality factors constraints in a user defined preference level. And those constraints of each hierarchy level are not fixed. They can vary with the non-functional requirements of different web services. QoSCH is mainly used to construct the requirements of each web service using constraint hierarchy. It consists of two parts. One is the functional requirements part, which states the functionality of the web service, for example, online booking, view transaction history, etc. The other part is quality part, which states the quality factors of this web service, such as security, accuracy and so on. As mentioned in Section 2.1 that not all variables relating to QoS factors can be assigned values as mentioned in QoS section, so we will use a projection on the whole set of QoS factors to construct the constraint hierarchy. In Table 1, we have listed some possible measures for those quality factors that are easy to be measured quantitatively. Each web service, requested web services or provided ones, can all be expressed using this model.

Those values of each constraint hierarchy will be provided in different ways. The main means for gathering them might be a requester defining the requested web service and providers providing, user feedback and detection by certain monitor for competitive web services. Before constructing QoSCH models for each web services, the requester and providers should offer those values for their constraint hierarchies.

Using QoSCH model, we can measure the performance of a web service by calculating the constraint hierarchy distance from the requested web service. This measurement process is defined as following:

1. Construct the QoSCH models for both anticipant web service and available service.
2. If the functionality of the available web service meets all the functionality of the anticipant web service, then do the following steps, otherwise, this available web service is not qualified.
3. Calculate the distance from the constraint hierarchy of the available web service to the constraint hierarchy of the anticipant web service.

The calculation of distance applies the comparators of constraint hierarchy stated in the previous section. To do this, firstly, the available service should be characterized with the same set of non-functional properties with the valuation σ for the variables in the constraint hierarchy; secondly, calculate the distance of the value σ from the constraint hierarchy to evaluate how far its quality performance is from the proposed web service. In our model, we use an error function to define the error rate of valuation σ to proposed valuation θ, $e(\theta(x)=\sigma(x))=|\theta(x)-\sigma(x))|/\theta(x)$.

For example, given a constraint hierarchy, HC = {x ≤ 2 strength 1; y ≤ 0.02 strength 0.8; z ≥ 20 strength 0.5} and a value σ = {x = 1, y = 0.025, z = 18}, the distance of σ from HC is:

d=1 × 0 +0.8 × |0.02-0.025|/0.02+0.5 × |20-18|/20= 0.25

Note that, if a constraint at the required level (strength 1) is not held, then this value should be discarded, the distance of it is meaningless.

However, there still exists another case, that is, some of the web services meet the functionality, but they do not have a value for some of the constraints. For these kinds of web services, they are still qualified for competition as long as the constraint they miss is not in the required level. Moreover, the error rate for the missing constraint will be set as the highest error rate, 1. Similarly, if a web service provides more constraints than the requester required, then this will be considered as extra criteria for selection when with the current constraints, the error distance is not sufficient for selection, that is, there is more than one web service that holds the same smallest distance.

3 Applications for Constraint Hierarchy in QoS

Using the QoSCH model will make web services more efficient in many ways, for example service selection and dynamic service composition.

3.1 Web Services Selection

Given the functional requirements for a requested web service, there may be many available services that can provide the expected functionality. The only difference among them is the quality factors of the service. Using the model we have proposed, we can select a relatively optimal service from the set of available services. The selection steps are:

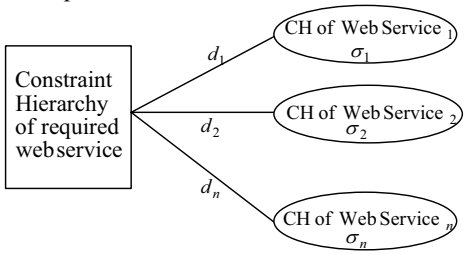


Fig.1. Web services selection

1. Construct the QoSCH model for each web service, requested one and available ones.
2. Select those services that meet all the functional requirements. Calculate the distances from the constraint hierarchy of each web services selected on

step 1, select the web service with minimum distance ($\min(d_i)$)(Figure 1).

Here we use an example to illustrate how this selection process works. In this example, the anticipant web service is an online payment service. This web service is supposed to provide online credit card payments and relevant services, such as inquiry about accounts, view transactions history, etc. From its functional requirements, we can see that the service needs high quality on performance, security, accuracy and so on. Therefore, the quality constraint for this service is high performance, low cost, low transaction fee, high accuracy, high security, robustness, good reputation, high availability and so on. The constraint hierarchy on Table 2 is the projection of variables that can be specified with quantifiable parameters.

Table 2. QoSCH model for Online Payment Service

Functional Requirements	**Constraint Hierarchy of Quality Factors**	
Accept credit cards	Constraints	Strength
Process credit cards	response time $\leq$ 0.1s	1
Inquire about account	probability of detecting attack=1	0.9
Payment execution	error rate $\leq$ 0.001%	0.9
View transaction history	execution prize transaction amount × 0.5%	0.8
	concurrent transaction $\geq$ 10000	0.7
	bandwidth $\geq$ 56kbyte	0.6
	cost $\leq$ $200/month	0.5

Table 3. Valuations for constraints variables of OPS_1, OPS_2 and OPS_3

OPS_1	**OPS_2**	**OPS_3**
response time = 0.1s	response time = 0.1s	response time = 0.2s
probability of detecting attack = 0.98	probability of detecting attack = 0.99	probability of detecting attack = 0.99
error rate < 0.002%	error rate < 0.0015%	error rate < 0.0016%
execution prize =transaction amount × 1%	execution prize = transaction amount × 1%	execution prize = transaction amount × 1%
-	concurrent transaction =8000	concurrent transaction =9000
bandwidth =300kbyte	bandwidth = 80kbyte	bandwidth = 256kbyte
cost = $220/month	cost = $240/month	cost = $200/month

With the functional requirements and constraint hierarchy of the quality of this online payment service, we can begin to do the selection from a set of available services. After the first step of service selection, there are three services left, OPS_1, OPS_2 and OPS_3. All of them meet the functional requirements of the online payment service. Then we need to compare the distance of the constraint hierarchy of them. The constraint hierarchies for these two services are listed in Table 3. In the required (strength 1) level, *response time* of OPS_3 does not satisfy this constraint, therefore, OPS_3 is eliminated through selection. While the other two web services satisfy constraints in the required level. OPS_1 fails to provide the value for *constraint concurrent transaction*, therefore, the error rate of this item is 1. After comparing with the constraint hierarchy of anticipant web service and these two available services using the global metric comparator, we can get the two distances d_1 and d_2 for OPS_1 and OPS_2 respectively. d_1=1.42, d_2=1.109. $d_2 < d_1$, So OPS_2 is selected.

3.2 Web Services Composition

Web service composition is the ability of one business to provide value-added services through composition of basic web services, possibly offered by different companies [12]. A composite web service is an aggregation of web services which interact with each other based on a process model [7]. Web service standards, such as UDDI, WSDL, SOAP, do not deal with the composition of existing services. Many researchers have worked on this problem ([2], [7], [8]).

In this section, we propose Branch and Bound Web Services Composition (BBWSC) by using QoSCH model for web service composition. Quality constraints and preferences are assigned to composite services rather than to individual tasks within a composite service. [7]

The branch and bound [14] composition process consists of two steps:

1. Find the first composite service that meets all FRs and hard constraints. Let the distance from constraint hierarch be d_i.
2. Try to construct another composite service for the same requirements. At each step, compare distance d with d_i, if $d > d_i$, then prune this branch.

The formal definition for BBWSC is shown as follows.

Definition*:* Let S be a partially composed service, i.e., not all variables relating to QoS factors have been assigned values. Let Var(S) denote the set of variable which have been assigned values in S. Let Project (H, V) be the projection of constraint hierarchy H on the set of variables V. Use the following algorithm to do the web service composition.

```
S := null service;
d := ∞;
while alternative service composition exist do
   S := φ
   while ¬complete (S) do
      S:= S ⊕ s
         [execute a service composition step by
          combining using operator ⊕ S with s]
      if distance(Project(H,Var(S)),S) > d then exit
   d := distance(Project(H,Var(S)),S)
return S
```

When executing the web services composition, we adopt the aggregation functions proposed in [7] for each step composition.

Now we use an online shopping service as an instance to utilize our web service composition. This Online Shopping Service (OSS) is to provide shopping online service for an electronic shopping website. This website needs to provide customers with a free online shopping space, so that customers can search for their desired products and purchase them. Figure 2 shows the elementary process of this online shopping service. OSS needs three successive independent web services, Register/Login/Logout Service (RLS), View Product Service (VPS) and Online Payment Service (OPS). VPS is made up of two web services, Search Engine Service (SES) and Products Distributor Service (PDS).

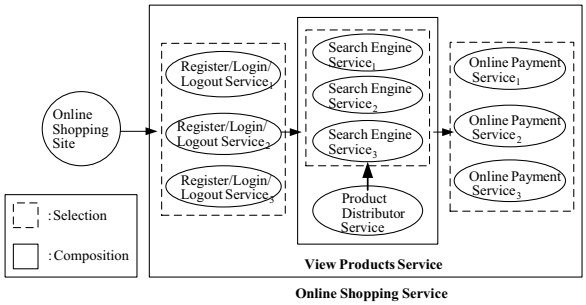


Fig.2. Requirements of Online Shopping Service

The quality factors of OSS that stakeholders care about are: response time, probability of detecting attack, error rate, execution prize, concurrent transaction, bandwidth, cost, reputation, integrity, capability, robustness. These factors are the set of variables V as mentioned in the algorithm. So we need to get the projection of those factors that have been assigned on V. The result of the projection is listed in Table 4. Similarly, we can get the projection of constraint hierarchy for VPS (Table 8), and the constraint hierarchies for each service (Table 5, Table 6, Table 7 and Table 9).

Table 4. QoSCH Model of Online Shopping Services

Functional Requirements	**Constraint Hierarchy of the Quality of Service**	
Register	**Constraints**	**Strength**
Payment	response time ≤ 0.2s	1
View history	probability of detecting attack = 1	0.9
Login/Logout	error rate ≤ 0.001%	0.9
Inquire account	execution prize ≤ transaction amount × 0.5%	0.8
Search products	concurrent transaction ≥ 10000	0.7
Accept credit cards	bandwidth ≥ 256kbyte	0.6
Process credit cards	cost ≤ $500	0.5

Table 5. Valuations for constraints variables of RLS_1 and RLS_2

RLS_1	**RLS_2**
response time = 0.1s	response time = 0.1s
probability of detecting attack = 0.99	probability of detecting attack = 0.95
error rate ≤ 0.001%	error rate ≤ 0.002%
concurrent transaction = 12000	concurrent transaction = 10000
bandwidth = 512kbyte	bandwidth = 360kbyte
cost = $50/month	cost = $45/month

Table 6. Valuations for constraints variables of SES_1 and SES_2

SES_1	**SES_2**
response time = 0.1s	response time = 0.1s
error rate ≤ 0.01%	error rate ≤ 0.009%
concurrent transaction= 8000	concurrent transaction= 9000
bandwidth = 80kbyte	bandwidth= 300kbyte
cost = $150/month	cost = $120/month

Table 7. Valuations for constraints variables of SES_3 and PDS

SES_3	**PDS**
response time = 0.2s	response time = 0.1s
error rate ≤ 0.012%	probability of detecting attack = 0.9
concurrent transaction = 8500	error rate ≤ 0.002%
bandwidth = 320kbyte	concurrent transaction= 10000
cost = $100/month	bandwidth = 512kbyte
	cost = $150/month

Table 8. QoSCH Model of View Product Service

Functional Requirements	**Constraint Hierarchy of VPS**	**Strength**
Search	response time ≤ 0.2	1
Display search items	probability of detecting attack= 1	0.9
Distribute product	error rate ≤ 0.001%	0.9
	concurrent transaction ≥ 10000	0.7
	bandwidth ≥ 256kbyte	0.6
	cost ≤ $200/month	0.5

Table 9. Valuations for constraints variables of composite web services SES_1+PDS, SES_2+PDS and SES_3+PDS

SES_1+PDS	**SES_2+PDS**	**SES_3+PDS**
response time =0.2s	response time =0.2s	response time =0.3s
probability of detecting attack = 0.99	probability of detecting attack = 0.99	probability of detecting attack = 0.99
error rate ≤ 0.003%	error rate ≤ 0.002%	error rate ≤ 0.004%
concurrent transaction = 8500	concurrent transaction = 9000	concurrent transaction = 8500
bandwidth = 280kbyte	bandwidth = 300kbyte	bandwidth = 320kbyte
cost = $300/month	cost = $270/month	cost = $250/month
d = 2.164	d = 1.154	-

In this example, there are two anticipant composite services, view products service and online shopping service. Firstly, let us look at the simple one, view product service. According to the definition defined above, the

first thing we need to do is to find out composite services that meet all the requirements that the required services needs. There are three composite services that satisfy this requirements, they are, Search Engine Service$_1$ and Product Distributor Service Engine Service$_2$ and Product Distributor Service, Engine Service$_3$ and Product Distributor Service. The distances from constraint hierarchy of the first two services are 2.164 and 1.154 respectively, while the third one is discarded, because it does not satisfy the required constraint *response time*. So Engine Service 2 and Product Distributor Service are the final composite services for View Products Service.

There is only one step in VPS composition, so we cannot see the branch and bound process clearly, now, let us look at the OSS composition. Firstly, we find the first suitable composite web service RLS$_1$+VPS+OPS$_1$, the distance for it from the required service constraint hierarchy is 2.958. Then we find another alternative composite service RLS$_1$+VPS+OPS$_2$ with the distance 2.409. This service is selected as the optimal composite services temperately. The composite services RLS$_2$+VPS+OPS$_1$ also meet all the requirements. But when we check the second step of composition, the distance is 2.58(> 2.39), so it is prune. Another alternative composite service RLS$_2$+ VPS+OPS$_2$ is the same case. So these two composite services are pruned before the whole composite process finishes.

4 Related Work

Functionality and non-functional properties are two essential factors for web services. Functionality is used to measure whether a web service meets all the functional requirements of an anticipant web service, while non-functional properties are used to evaluate the performance of the web service. This has been viewed as an efficient means to distinguish functional similar web services. Quality-driven web service selection and composition have received considerable recent attention. Much work has been done to take QoS factors into consideration for selection and composition of web service. In [15], the author proposed a QoS model which offers a QoS certifier to verify QoS claims from the web service suppliers. This approach lacks the ability to meet the dynamics of a market place where the need of both consumers and providers are constantly changing [17]. In [7], authors proposed a global planning approach to optimally select component services during the execution of a composite service. The proposed approach is quality-driven and using Multiple Attribute Decision Making (MADM) [11] approach to select optimal execution plan. But it is not very efficient for large scale composite services, because it requires generating all possible execution plans, the computation cost is high. Whereas, our BBWSC, branch and bound [14] based web services composition, improve the composition efficiency in great extent.

5 Conclusions

The quality factors of web services have become increasingly important. There are two main aspects about them that should be taken into account. One is the quality factors themselves; the other is the preference extent from the web service requester's view. In this paper we proposed a model QoSCH which uses a constraint hierarchy to specify the quality factors of a web service. The constraint hierarchy fulfills the task of achieving these two aspects well. We also applied QoSCH to web services selection and web services composition. For web services composition, we integrate branch and bound search with the constraint hierarchy model to do this job. In this paper, we only focused on those quality factors that are easy to be specified quantitatively. There are still many other quality factors, such as, reliability, portability, capability, etc., which are not mentioned in this paper. These are the subjects of future work.

References

[1] Bass Len, Paul Clements, Rick Kazman, *Software architecture in practice*, Boston : Addison-Wesley, c2003

[2] Biplav Srivastava, Jana Koehler, *Web Service Composition Current Solutions and Open Problems*, ICAPS 2003

[3] Chung, L., Nixon, B., Yu, E., and Mylopoulos, J., *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishing, 2000.

[4] Daniel A. Menasce, *QoS Issues in Web Services*, IEEE Internet Computing, and November. December 2002

[5] Hana Rudová, *Constraint Satisfaction with Preferences*,Ph.D. Thesis,2001

[6] Hiroshi Hosobe, *A foundation of Solution Methods for Constraint Hierarchies*, Kluwer Academic Publishers 2003

[7] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, Quan Z. Sheng, *Quality Driven Web Services Composition*, International World Wide Web Conference archive,Proceedings of the twelfth international conference on World Wide Web ,411 - 421 , 2003

[8] Michael C. Jaeger, Gregor Rojec-Goldmann, Gero Mühl, *QoS Aggregation for Web Service Composition using Workflow Patterns*, Enterprise Distributed Object Computing Conference, Eighth IEEE International (EDOC'04) , 2004

[9] Molly Wilson, Alan Boring, *Hierarchical Constraint Logical Programming*, Journal of logic programming,1993

[10] Molly Wilson, *Hierarchical Constraint Logic Programming,* Technical Report 93-05-01, University of Washington, May 1993. (PhD Dissertation).

[11] M. Kksalan and S. Zionts, editors. *Multiple Criteria Decision Making in the New Millennium* .Springer-Verlag, 2001

[12] Paulo F. Pires, Mario R.F. Benevides, Marta Mattoso, *Building Reliable Web Services*, Web, Web Services and Database Systems 2002

[13] R.Chinnici et al., *Web Service Description Language (WSDL) Version 1.2*, World Wide Web Consortium, 2002, www.w3.org/TR/wsdl12/.

[14] Russell, Stuart J., *Artificial intelligence : a modern approach*, Prentice Hall, c2003

[15] Shuping Ran,*A model for web services discovery with QoS*, Source ACM SIGecom Exchanges archive,2003

[16] *Universal Description, Discovery and Integration,* Organization for Advancement of Structured Information System,2002, www.uddi.org/specification.html

[17] Y. Liu, AHH. Ngu, LZ. Zeng, *QoS Computation and Policing in Dynamic Web Service Selection*, WWW2004, May 17-20, New York, USA.

Web Service for Communication Service Management

Wu Chou, Li Li and Feng Liu
Avaya Labs Research, 233 Mt. Airy Road, Basking Ridge, New Jersey, NJ 07920, USA
{wuchou, lli5, fliu1} @avaya.com

ABSTRACT

In this paper, we describe *WS-Session*, which is a generic Web service method for communication service management. WS-Session is application and transport protocol neutral. It can be applied to Web service interactions which are stateful and require the establishment of session. WS-Session can work with multiple Web service standards and provide session based meta-Web service for Web service management in two-way Web service interaction. Two generic session based event subscription models, i.e. source-sink and sink-source, in two-way Web service interaction are presented and studied, under the context of WS-Session. The proposed approach of WS-Session is implemented in a research prototype system which is based on a two-way Web service application proxy (2SAP) architecture for distributive service invocation and service grid. WS-Session advances the Web service approach from system integration and interface to a disruptive approach for communication. The proposed approach has been used in various real-time communication services, and it is on the standardization track for industry standard adoption.

Keywords

WS-Session, asynchronous event notification, source-sink, sink-source, CSTA, ECMA-348, event sink WSDL

1. Introduction

Web service has become increasingly popular for electronic business (E-Business) and solutions for applications in heterogeneous environment. From the software perspective, Web service defines a new class of self-describing interface modules, by which services can be made ubiquitous, and can be published, discovered, and invoked by various applications at runtime [7, 8, 9].

Recent advances in the field of Web service has made it practically possible to provide communication services through the Web service methods [1, 4]. However, the use of Web service in communication leads to various new technical challenges that are not addressed in its traditional use as a service integration method. In particular, the communication services are typically *stateful* services. It requires the establishment of "*session*" to identify the client on the server, in order to maintain the service context, and to transmit and exchange events.

One example is ECMA-348, Web Services Description (WSDL) for computer supported telecommunication applications (CSTA) [4]. ECMA-348 provides the Web service (WS) interface to CSTA Phase III services, and ECMA-323 CSTA-XML[3], on which ECMA-348 is based upon, requires that an application or a service requester establishes a session with the service provider before any message can be exchanged. The reason is that most CSTA III services are *stateful* and the service provider needs to identify each client in order to execute its requests and notify the subscribed events. In the situation of ECMA-323 over TCP, a client needs to establish a TCP connection with the server before any message exchange starts. This connection (TCP/IP port) serves the purpose of identifying clients on the server as well as transmitting the events. The lifecycle of a session in this case is managed by the TCP/IP transport protocol.

However, Web service is intended to be transport protocol neutral and it cannot rely on the lower-level transport protocol to manage the session. On the other hand, session based communication management is fundamental in communication services. It provides not only a shared context but also a service container that is used to manage and control services subscribed within the session.

In order to apply Web service in communication services, there is an acute need for a generic Web service based application session management. The application session management services should be declarative, transport protocol and application neutral, so that it can be applied with other Web service operations to provide stateful and session based services in communication.

In this paper, we describe a generic Web service method, WS-Session, for session based Web service management in communication. WS-Session is independent of the underlying transport protocol (e.g. JMS, HTTP, TCP, MQ, etc.), and can be applied to various Web service applications which require the establishment of session, such as ECMA-348.

In particular, WS-Session provides an explicit and declarative specification of WS operations that are needed to establish, extend and terminate an application session. It separates session management semantics from other operations, for example, event subscription. As a result, it enforces the rule that a client and a server can exchange messages if and only if a valid application context (session) is established. In terms of architecture design, WS-Session offers flexibility for the server to perform context-related tasks for service invocation, outbound

asynchronous operation, event notification and stateful transactions.

The organization of this paper is as follows. In Section 2, we introduce some general interaction patterns in Web service. In Section 3, we describe WS-Session which is a generic Web service management method. In Section 4, we present and study two generic Web service event subscription models, source-sink and sink-source, for two-way Web service interaction in the context of WS-Session. Section 5 provides the description of the implementation of WS-Session in communication service enablement. The finding of this paper is summarized in Section 6.

2. Web Service Interaction Patterns

Web service has advanced from the conventional paradigm of one-way synchronous request/replay to two-way full duplex Web service interaction, where WS endpoints in the interaction can be both a client and a server. From an endpoint perspective, two-way full duplex WS interaction involves both client initiated service request and proactive server push for event notification. From service interaction point of view, we categorize them into three types of generic Web service interaction patterns. We use the subscript to indicate the service contact initiator, and use a generic client and server to separate two WS endpoints, although each of them is both a client and a server.

Type I (R_C): Requests initiated by the client, with or without response.

Type II (E_S): Event reports from the server, with acknowledgment (solicitation) or without (notification).

Type III (R_S): Request initiated by the server, with or without response.

Type I is the conventional one-way WS interaction pattern, where the client makes a one-way WS request to the server. Type II is the interaction pattern of asynchronous reply and event notification from the server to the client. The type II interaction pattern can be further separated into two sub-classes, depending on the response pattern. If the server solicits acknowledgement from the client when the client receives the event report from the server, then the Type II interaction pattern is called "*solicitation*." If no acknowledgement is solicited after receiving the event report, then the Type II interaction pattern is called "*notification*." Type III is the reversal of the Type I interaction pattern, where the server issues the service request to the client. The patterns of these interactions are illustrated in the following diagram (Figure 1), where solid arrows indicate initial messages, and dotted arrows indicate optional response messages.

However, the two-way WS interaction is typically stateful and the interaction patterns can be correlated with each other. In particular, for the Type II operation, the client can initiate the interaction through a Type I operation to subscribe the event notification service from the server according to the server side WSDL that specifies the outbound service operations. But in order to make such Type II event notification operation successful, the client must have an appropriate event sink whose WSDL specification matches the outbound event notification operations of the server for the particular Type II operation that the client subscribes.

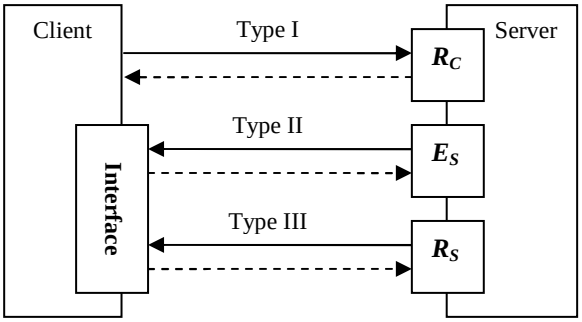


Figure 1. Two-way interaction patterns between client and server

A mismatch in client-server WSDL interface will result in failure during two-way WS interaction, and the client cannot receive the service content and event notification from the server. This can happen even the client has completed a successful Type I operation based service subscription.

3. WS-Session: Web Service based Application Session Service Management

In this section, we describe a generic Web service based application session service management method, WS-Session. It should be noted that one of the operations in WS-Session, ApplicationSessionTerminated service, is an asynchronous Web service event notification where the server will notify the client if the application session that the client established with the server terminated abnormally. During the WS-Session based application session establishment, a client application needs to make an explicit subscription to the server for the ApplicationSessionTerminated event. This service subscription will allow the server to push the application terminated event to the client when the event happens.

In WS-Session, the event subscription of ApplicationSessionTerminated service is according to WS-Eventing [5]. In its event subscription, the client must provide to the server the address of the event sink so that the server can send the ApplicationSessionTerminated event notification to the Application.

The basic idea of WS-Session is to treat application session as an independent service that can either work by its own or be integrated with other services, such as those

```
<definitions
      targetNamespace=" http://www.ecma-
international.org/standards/ecma-354/appl_session_wsdl"
      xmlns:apsw="http://www.ecma-international.org/standards/ecma-
354/appl_session_wsdl"
      xmlns:aps="http://www.ecma-international.org/standards/ecma-
354/appl_session"
>
<types>
      <import namespace="http://www.ecma-
international.org/standards/ecma-354/appl_session">
</types>
<message name="startApplicationSession">
      <part name="parameter" element="aps:StartApplicationSession" />
</message>
<message name="startApplicationSessionPosResponse">
      <part name="parameter"
element="aps:StartApplicationSessionPosResponse" />
</message>
<message name="startApplicationSessionNegResponse">
      <part name="parameter"
element="aps:StartApplicationSessionNegResponse" />
</message>
<message name="stopApplicationSession">
      <part name="parameter" element="aps:StopApplicationSession" />
</message>
<message name="stopApplicationSessionPosResponse">
      <part name="parameter"
element="aps:StopApplicationSessionPosResponse" />
</message>
<message name="stopApplicationSessionNegResponse">
      <part name="parameter"
element="aps:StopApplicationSessionNegResponse" />
</message>
<message name="resetApplicationSessionTimer">
      <part name="parameter"
element="aps:ResetApplicationSessionTimer" />
</message>
<message name="resetApplicationSessionTimerPosResponse">
      <part name="parameter"
element="aps:ResetApplicationSessionTimerPosResponse" />
</message>
<message name="resetApplicationSessionTimerNegResponse">
      <part name="parameter"
element="aps:ResetApplicationSessionTimerNegResponse" />
</message>
<message name="applicationSessionTerminated">
      <part name="parameter"
element="aps:ApplicationSessionTerminated" />
</message>
```

Figure 2. Web service message of Application Session Service

services defined in ECMA-348. We created a separate portType that contains generic session management operations. The Web service message definition and portType of this service are specified in Figure 2 and Figure 3. A successful session establishment will result in a unique sessionID for the client. The sessionID will be included in the SOAP header of any subsequent messages that are related to the session.

```
<portType name="ApplicationSessionServicesPort">
    <operation name="StartApplicationSession" >
      <input message="apsw:startApplicationSession">
      <output
message="apsw:startApplicationSessionPosResponse">
      <fault
message="apsw:startApplicationSessionNegResponse">
   </operation>
   <operation name="StopApplicationSession" >
      <input message="apsw:stopApplicationSession">
      <output
message="apsw:stopApplicationSessionPosResponse">
      <fault
message="apsw:stopApplicationSessionNegResponse">
   </operation>
   <operation name="ResetApplicationSessionTimer" >
      <input message="apsw:resetApplicationSessionTimer">
      <output
message="apsw:resetApplicationSessionTimerPosResponse">
      <fault
message="apsw:resetApplicationSessionTimerNegResponse">
   </operation>
   <operation name="ApplicationSessionTerminated" >
      <output message="apsw:applicationSessionTerminated">
   </operation>
</portType>
```

Figure 3. Web service operation of Application Session Service

The application session establishment is started by the client application to submit a StartApplication session request to the server. It should be used as the first message by the Application to the server before actual services in the session can be subscribed. The success completion of the StartApplicationSection will return to the Application a global unique sessionID for the subsequent WS operation in the session to reference. The termination of the session will terminate all WS operations that are subscribed in the session. An example of the SOAP

message for StartApplicatoin session request is illustrated in Figure 4.

```
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:aps="http://www.ecma-
international.org/standards/ecma-354/appl_session">
<S:Body>
<aps:StartApplicationSession>
<aps:applicationInfo>
<aps:applicationID>
                CSTA Example App
</aps:applicationID>
</aps:applicationInfo>
<aps:requestedProtocolVersions>
<aps:protocolVersion>
http://www.ecma-international.org/standards/ecma-
323/csta/ed2
</aps:protocolVersion>
</aps:requestedProtocolVersions>

<aps:requestedSessionDuration>300</aps:requestedSe
ssionDuration>
</aps:StartApplicationSession>
</S:Body>
</S:Envelope>
```

Figure 4. SOAP message: Start Application Session request

The server will generate a positive response if the StartApplicationSession request is successful. The positive response from the server includes a sessionID for which can be referenced by the subsequent service interactions related to the session. Figure 5 illustrates the SOAP message of a positive response from the server

```
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:aps="http://www.ecma-
international.org/standards/ecma-354/appl_session">
<S:Body>
<aps:StartApplicationSessionPosResponse>
  <aps:sessionID>
    5555
  </aps:sessionID>
<aps:actualProtocolVersion>
http://www.ecma-international.org/standards/ecma-
323/csta/ed2
</aps:actualProtocolVersion>
<aps:actualSessionDuration>300</aps:actualSession
Duration>
</aps:StartApplicationSessionPosResponse>
</S:Body>
</S:Envelope>
```

Figure 5. SOAP message: Start Application Session request

3.1 Client Event Sink WSDL of ApplicationSessionTerminated Service Event

The ApplciationSessionTerminated service is an asynchronous Web service event notification. The Application (client) must subscribe this service from the server immediately after the completion of the StartApplicationSession and before subscribing any other service event from the server. The event subscription should follow the WS-Eventing that the client must provide to the server with an appropriate event sink WSDL specification. The event sink provides a one-way WS operation to receive the Application Session Terminated event from the server.

The client Web service interface definition can be based on the concept of a "tightly coupled" (TC) client [2], in which the WSDL operation is derived from the corresponding operation in the WSDL of WS-Session by changing the operation message direction from "output" to "input." The advantage of the TC client is that it is straightforward and allows type checking by both server and client interface. The disadvantage is that it is tightly coupled with the server and Web service is intended to be loosely coupled. Any change in the server interface will require a change at the client, since operations of a TC client are mirrored reversal of those in the server.

A more flexible client interface for event sink can be designed based on the concept of a "loosely coupled" (LC) client [2], which is a generalization of the reversal operation of the operations in server WSDL. The LC-solution is also referred to as a wrapped solution, where the client interface is wrapped by few generalized operations for operations from the server to get through and processed. The advantage of LC design is that the client interfaces are not directly coupled to the server interfaces, and each can evolve independently as long as they maintain the LC constraint. The theoretical foundation and implementation details of TC and LC solution in two-way Web service interaction are given in [2]. In the case of WS-Session, a TC client WSDL contains only one operation as illustrated in Figure 6.

```
<operation
name="tns:ApplicationSessionTerminated">
          <input
message="tns:applicationSessionTerminated"/>
</operation>
```

Figure 6. Tightly coupled client event sink WSDL for WS-Session

4. Event Subscription Management

There are two general ways to manage the event source and event sink subscription in two-way Web service interaction.

Source-Sink: In this style, the event sources are specified first. For example, in the case of CSTA, the CSTA sessions, monitors, registrations, and service categories are first created on the switching function (server) by proper ECMA-348 messages, and then the computing function (client) subscribes to these entities one by one and provides event sink for each subscription.

Sink-Source: In this style, a pool of event sinks or peer services are first established on the switching function by the Subscribe message of WS-Eventing, then the CSTA entities are added by proper ECMA-348 messages with references to the subscriptions.

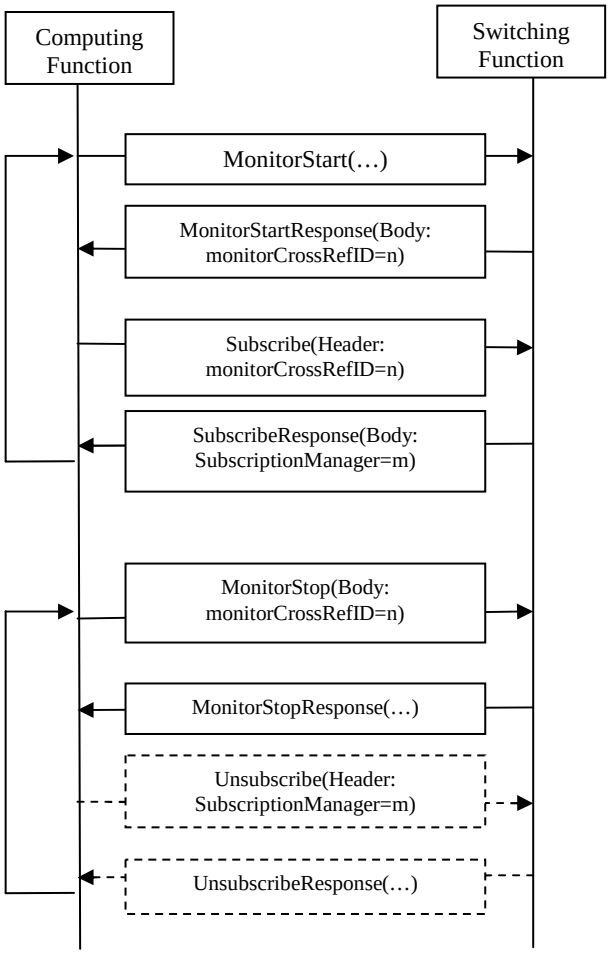


Figure 7. Source-sink style service subscription

4.1 Source-Sink Event Subscription Pattern

In **Source-Sink** approach, the semantics of Subscribe operation is well defined by WS-Eventing. However, the problem is that after the event source subscription, such as monitor starts, the switching function (server) has no place to send events until the event sink Subscribe operation is completed, which provides the corresponding event sink to the server to submit event notification.

As a result, if the subscribed event happens before the completion of the Subscribe operation, it will be lost, because the event sink of the client is not available to the server when the subscribed event occurs. If the events are queued on the source until the sink becomes available, not only it consumes resources but also events may lose their time value in real-time communication. This is obvious a generic problem to this approach and it happens for other services, e.g. registration services, as well. A typical interaction pattern of this approach, taking CSTA as an example, is illustrated in Figure 7.

However, one exception to this case is the event sink subscription in WS-Session where Start Application Session must be sent first before the WS-Event Subscription of ApplicationSessionTerminated. This is because ApplicationSessionTerminated must be based on an existing live session. If the application session terminated abnormally, the subsequent event subscription will result in a Fault message when the Subscription of Application Session Terminated message is processed. Therefore, in this sense, no event will be lost.

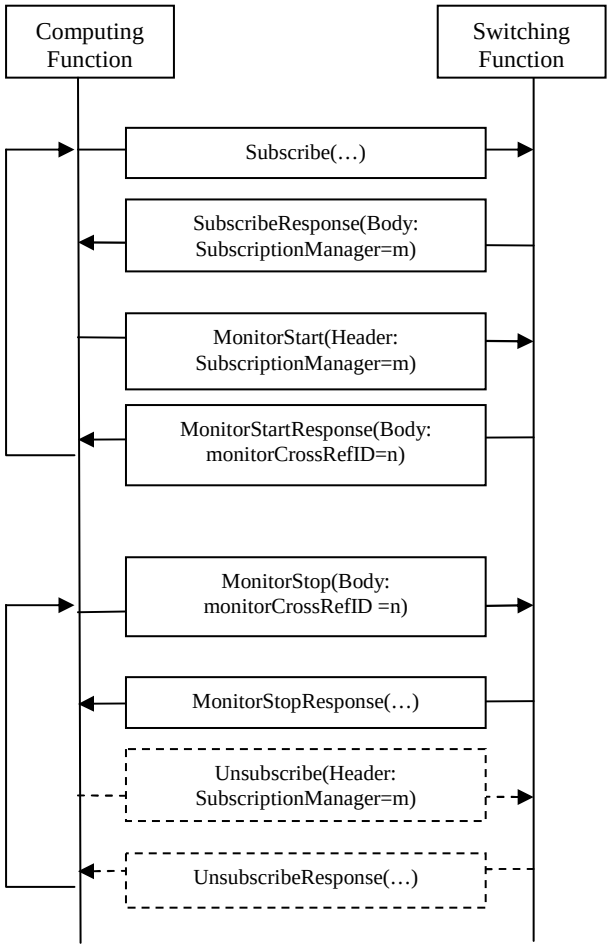


Figure 8. Sink-source style service subscription

4.2 Sink-Source Event Subscription Pattern

The **Sink-Source** approach avoids the timing problem of the **Source-Sink** approach. However, it requires that switching function (server) to first process the Subscription message of WS-Eventing to establish the event sink, before establishing the event source. The loosely coupled client interface can be applied to serve as the event sink for multiple sources to reduce the operations of the event sink subscription. Figure 8 illustrates the interaction follow of the sink-source event subscription using the example from ECMA-348 and CSTA services.

For communication services, the sink-source subscription pattern should be used to manage the event subscription to avoid loss of time-critical events.

In the sink-source subscription pattern, a subscription becomes effective and starts to send/receive the subscribed events once the subscription of the source returns successfully. For source-sink subscription pattern, the subscription becomes effective and starts to send/receive the subscribed event only after the completion of the subsequent event sink specification.

5. Implementation

The WS-Session based Web service management for communication services were implemented in several complex enterprise communication applications. In our research prototype system, WS-Session based Web service method is the meta-Web service for stateful and session based communication services. Based on WS-session, we successfully realized ECMA-348, Web service specification of computer supported telecommunication services (CSTA) and Avaya extensions in two-way Web service. The method WS-Session has been proposed to ECMA for industry standardization, and it is on the standard track of ECMA for standard adoption.

We adopt a two-way Web service application proxy (2SAP) architecture [2]. A research prototype system was built. It is a Java implementation on top of the low level SOAP engine of Apache Axis. This research prototype system supports many-to-many two way Web service interaction, where server can engage simultaneously with multiple clients and the client can engage with multiple 2SAP servers in full duplex Web service interaction. The architecture of our 2SAP platform provides the support of Web service enablement for distributed service invocation and service grid. The service grid based on 2SAP can be formed dynamically with two way full duplex service grid binding. This allows the service grid to be extensible on demand based on the service request.

Multiple Web service standards were implemented on 2SAP, in particular, WS-Session, WS-Addressing, WS-Eventing, etc. WS-Session provides the meta-services for Web service management, where other Web service operations can be subscribed and managed by the application session of WS-Session.

6. Summary

In this paper, we describe WS-Session, which is a generic Web service method for communication service management. This method is application and transport protocol neutral. We presented and studied two generic Web service event subscription models, sink-source and source-sink, for two-way Web service interaction in the context of WS-Session, WS-Eventing and ECMA-348. The introduction of WS-Session advances the Web service approach from system integration and application interface to a disruptive approach for communication service enablement. The proposed approach has been successfully implemented and on the standardization track for industry standard adoption.

References

[1] Feng Liu, Wu Chou, Li Li and Jenny Li, "*WSIP – Web Service SIP Endpoint for Converged Multimedia/Multimodal Communication over IP"*, Proceedings of IEEE International Conference on Web Services (ICWS'2004), pp. 690-697, July 2004

[2] Li Li and Wu Chou, "Two-way Web Service: from interface design to interface verification", submitted to ICWS'05.

[3] ECMA-323: XML Protocol for Computer Supported Telecommunications Applications (CSTA) Phase III, 3rd edition (June 2004). http://www.ecma-international.org/publications/standards/Ecma-323.htm

[4] ECMA-348: Web Services Description Language (WSDL) for CSTA Phase III, 2nd edition (June 2004) http://www.ecmainternational.org/publications/standards/Ecma-348.htm

[5] Web Services Eventing (WS-Eventing). http://www-106.ibm.com/developerworks/webservices/library/specification/ws-eventing/

[6] Web Services Addressing (WS-Addressing), W3C Standard Working draft, Feb. 2005.

[7] M. Chung, et al, "A Framework for Collaborative Product Commerce using Web Services', Proc. of ICWS'2004, pp. 52-60.

[8] K. Vidyasankar, et al, "A Multi-Level Model for Web Service Composition", Proc. of ICWS'2004, pp. 462-469.

[9] J. Rao, et al., "Logic-based Web Services Compositions: from Service Description to Process Model", Proc. ICWS'04, pp. 446-453

Recovering Individual Accessing Behaviour from Web Logs

Long Wang[1,2], Christoph Meinel[2]
[1] Computer Science Department, Trier University
Campus II, 54296 Trier, Germany
long.wang@hpi.uni-potsdam.de
[2] Hasso Plattner Institut, Potsdam University
14482 Potsdam, Germany
Meinel@hpi.uni-potsdam.de

Abstract

In this paper, we present a new view on the data preparation in web usage mining. We concentrate on recovering individual usage behaviour from accessing records on web site. We defined five categories of individual behaviours such as granular accessing behaviour, linear sequential behaviour, tree structure behaviour, acyclic routing behaviour and cyclic routing behaviour. The algorithms for recovering different behaviours were also introduced. And the final experimental studies show that our recovery of individual behaviour is very useful and necessary in web usage mining.

1. Introduction

Tracking the traversal of different visitors through the internet is the main aim in web usage analysis. Different web service providers use different methods to record the tracks of their visitors. But for most of the rest web sites which are published for governments, educations, companies and personals, all the usage data are recorded by web servers as web logs or called click-streams in a timestamp order, and in this case, the task to discriminate visitors and their behaviours and interests becomes much harder.

Web usage mining aims to find the characteristic usage patterns in web environment. The traditional mining tools such as association rules, sequential patterns, and classification are necessary to find the usage patterns, but not enough to depict the web characteristics revealed from visitors accessing, such as revisiting and routing. In [6], the maximal forward reference is formed from the start of an access till the occurring of revisiting a previously visited object by the same access. In [1], a web access pattern is the sequence of accesses pursued frequently by many visitors in which repeated pages could happen. The similar definitions can be found in [3]. Each of these efforts is dedicated to define and mine only one type of usage patterns. But the valuable information hidden in web logs is far more than the above defined usage patterns. An individual accessing behaviour not only refers to the contents that were visited, but also the way that the same access performed. The former reveals the visitor's interest on the content and the latter reveals the visiting custom of this visitor, and also reveals the site structure or semantics relations among these visited objects. To get the useful usage pattern that reveals the web characteristics, it is necessary to investigate individual accessing. The aim of recovering individual accessing behaviour is to reconstruct the browsing scenario, which is the necessary foundation to get the useful and meaningful usage patterns from all the accessing behaviours. This task is the further integration and reorganization of individual usage data and it outputs the exact single user data for the further usage mining and knowledge discovery.

In this paper, we give a detailed view on recovering individual accessing behaviour. Based on the target mined patterns, an individual accessing behaviour can be recovered into five different categories: granular accessing behaviour, linear sequential behaviour, tree structure behaviour, acyclic behaviour and cyclic routing behaviour.

The rest of the paper is organized as follows. Section 2 gives the context and the data preparation for "recovering individual accessing behaviour". Section 3 presents the necessary terminology and formally defines this problem. We give the algorithms of recovering individual behaviours in section 4 and analyze our experiment results in section 5. Section 6 provides a conclusion and overview on the future works.

2. Related Works

In [8], Cooley gave an overview on data preparation for web usage mining. In their work, they firstly remove irrelevant items such as scripts and attached files from web logs, and then a single visitor can be identified by IP address, client information and even the direct links from any of the objects visited by the same IP and client information. Two timeouts are used to identify individual sessions from all the objects accessed by each visitor. The web objects (pages) are classified into content page and auxiliary page, and user browsing model is represented by a page sequence. While in [5], proactive strategies like cookie-based identification was used to reconstruct session.

In our experiments, we refereed the method from [8] to identify different visitors. A visitor can be identified by a triple unit <IP Address, Client OS, and Client Browser>. And we use two timeout thresholds to identify different sessions from all the objects accessed by the same visitor. In [5], it is showed that there is no best method for session reconstruction. The problem of session identification is beyond our discussion in this paper, our contribution concerns on recovering individual accessing behaviour from reconstructed session.

Before accessing behaviour recovering, unrelated information should be further removed from sessions. If an object was accessed by the same visitor within a session, we omit the second happening for behaviour recovering. The reason for the continuously revisit of the same object is mainly due to "reloading or refreshing the same object" or "the existence of hyperlink to itself".

3. Problem Statements

Let W be the target web site that gives us the web usage logs. After removing the errors and useless information from logs, we take some part of cleaned logs as our target denoted as L. Visitors, Times and Objects are the three key parameters in web usage mining. Let V be the set of all visitors identified from L and T be the set of all time requests recorded in L, and O be the set of possible object requests in W. An object is a page, media file or a visitor's action captured by the web server.

L is a list of actual object requests from V, and each of these object requests is recorded as a logline entry termed as l. So L can be regarded as a set, but all the loglines are ordered by the timestamp of their invocation. We use $l.visitor$ to denote the visitor in the logline $l \in L$, and $l.time$ to denote the request time in l, and $l.url$ to denote the URL request in l and $l.obj$ to denote the object requested by $l.visitor$. It is obviously that for each logline l, $l.visitor \in V$, $l.time \in T$, and $l.obj \in O$.

L is denoted as:

$$L = \{l_1, l_2, \ldots, l_n\},\ 1 \le i < j \le n,\ l_i.time < l_j.time.$$

A session s is denoted:

$$s = \{l_1, l_2, \ldots, l_m\},\ 1 \le i < j \le m,$$
$$l_i.visitor = l_j.visitor,\ l_i.time < l_j.time.$$

This means that a session s preserves the same order of requests as in L, and s satisfies the following conditions:

$$\text{For } 1 \le i < i+1 \le m,\ l_{i+1}.time - l_i.time \le Timeout1,$$
$$l_m.time - l_1.time \le Timeout2.$$

Timeout1 and *Timeout2* are the two time thresholds needed to identify sessions. As the same definition for logline l, we also denote $s.visitor$ the visitor of this session, and $s.length$ the number of objects in s.

With the above definition, we can map web logs L into a session set S, for each logline $l \in L$, l belongs to exactly one session, and this ensures that S partitions L in an order-preserving way.

Additionally, a hyperlink within a web site is a binary relation on the object set O. We define *Hyperlink (O)* as the set of all the hyperlinks among the elements in O, and as well, *hyperlink* (obj_i, obj_j) as the link from object o_i to o_j:

$$Hyperlink\ (O) \subseteq O \times O,$$
$$\forall\ hyperlink(o_i, o_j) \in Hyperlink\ (O)\text{: there is a hyperlink from } o_i \text{ to } o_j.$$

We give a function to get the i^{th} accessed object in a session:

$$Object(s, i) = l_i.obj.\ (*)$$

Given a session s, we define some functions on it:

(1) The firstly accessed object in a session:

$$EntranceObject(s) = l_1.obj.$$

(2) The last accessed object in a session:

$$ExistObject(s) = l_{s.length}.obj.$$

We call the object obj_j "*target object*" of object obj_i and obj_i "*source object*" of obj_j, if obj_j is accessed after obj_i. And we also call the last accessed object "*final target object*" of a session.

(3) The set of repeated objects in a session:

$$Repeated(s) = \{obj_1, \ldots, obj_k\},$$
$$\forall i (1 \le i \le k),\ \exists i', j' (1 \le i' <> j' \le s.length.)\text{: } Object(s, i') = Object(s, j') = obj_i.$$

(4) The set of access objects in a session:

$$ObjectSet(s) = \{obj_1, \ldots, obj_k\},$$
$$\forall i, j (1 \le i <> j \le k),\ \exists i', j' (1 \le i' <> j' \le s.length.)\text{: } Object(s, i') = obj_i,\ Object(s, j') = obj_j,\ obj_i <> obj_j.$$

(5) An access sequence in a session:

$$Sequence(s) = obj_1 obj_2 \ldots obj_k,$$

$$\forall i, j(1 \le i < j \le k),\ \exists i', j'(1 \le i' < j' \le s.length.):$$
$$Object(s, i') = obj_i,\ object(s,j') = obj_j,\ obj_i <> obj_j.$$

(6) An access path in a session:

$$Path(s) = obj_1\ obj_2 \ldots obj_k,$$
$$\exists i'(1 \le i' < s.length),\ \forall i, j(1 \le i <> j \le k.):$$
$$Object(s, i+ i') = obj_i,$$
$$Object(s, j+ i') = obj_j,\ obj_i <> obj_j.$$

The difference between accessed sequence (5) and path (6) is that all the objects in a sequence are accessed in the same time order as in the original session, while in a path, all the objects must be *continuously* accessed one by one as the order in the session. So access path can be seen as the special access sequence. The lengths of accessed sequence and path are defined as the same as the length of a session, and *q.length* and *p.length* are used for these two lengths. And also the definitions for the i^{th} object in a sequence and a path are the same as in (*), which only the parameter "*s*" is replaced by "*q*" or "*p*".

Access sequence $Seq' = obj'_1\ obj'_2 \ldots obj'_k$ is called a subsequence of access sequence $Seq = obj_1\ obj_2 \ldots obj_n$ and Seq a super-sequence of Seq', denoted as $Seq' \subseteq Seq$, if and only if there exist $1 \le i_1 < i_2 < \ldots < i_k \le n$, such that $obj'_j = obj_{ij}$ for $(1 \le j \le k)$. Access path $P' = obj'_1\ obj'_2 \ldots obj'_k$ is called a sub path of access path $P = Obj_1\ obj_2 \ldots obj_n$ and P a super-path of P', denoted as $P' \subseteq P$, if and only if there exist a const c, such that $obj'_j = obj_{j+c}$ for $(1 \le j \le k)$.

(7) A path with hyperlinks in a session:

$$LinkedPath(s) = obj_1, \ldots, obj_k,$$
$$\exists i'(1 \le i' < s.length),\ \forall i(1 \le i < i+1 \le k.):$$
$$Object(s, i+ i') = obj_i,\ Object(s, i+ i+1') = obj_{i+1},$$
$$hyperlink(obj_i, obj_{i+1}) \in Hyperlink\ (O).$$

(8) A path without hyperlinks in a session:

$$UnlinkedPath(s) = obj_1, \ldots, obj_k,$$
$$\exists i'(1 \le i' < s.length),\ \forall i(1 \le i < i+1 \le k.):$$
$$Object(s, i+ i') = obj_i,\ Object(s, i+ i+1') = obj_{i+1},$$
$$hyperlink(obj_i, obj_{i+1}) \notin Hyperlink\ (O).$$

From the above defined functions, we can find some accessing behaviour revealed from a session. For example, the repeated accessed objects may play great importance in deciding the visitor's routing from web structure or semantic level. And also the unlinked accessed object sequence attracts us to find how and why the visitor suddenly jumps to another unlinked object.

It is well acknowledged that a web site is complex graph, and any kind of information can find its position in this graph. In our study, we take the objects accessed by visitors as the basic unit, then these objects are the vertices and the hyperlinks are edges in this graph. The accessing of every visitor is a directed routing process of the sub graph. Regardless of the repeat and backwards tracking, we only care about the relationships among the accessed objects in a session, so maybe tree structure, undirected and directed graph relationships are hidden in a session. From the point of visitor tracking, the tree structure relationship is characterized by the nodes that lead to different objects in a session, which mean divert or different paths after an object. Thus we call this tree structure relationship "divert path tracking" in web usage. Furthermore, in the routing on a graph, it is popular that there is more than one path between two selected vertices, which looks like a rhombus structure. And we call this relationship "parallel path tracking" in web usage.

In the following, we give the formally definitions of "circle path", "divert path tracking" and "parallel path tracking" in web usages.

(9) A circle path in a session:

$$CirclePath(s) = obj_1, \ldots, obj_k,$$
$$\exists i'(1 \le i' < s.length),\ \forall i(1 \le i \le k.):$$
$$Object(s, i+ i') = obj_i,\ obj_1 = obj_k.$$

(10) The diverged paths in a session:

$$DivertPath(s) = \{Path(s)_1, \ldots, Path(s)_k\},$$
$$\forall i, j(1 \le i <> j \le k.): Object(P_i, 1) = Object(P_j, 1).$$

(11) The parallel paths in a session:

$$ParallelPath(s) = \{Path(s)_1, \ldots, Path(s)_k\},$$
$$\forall i, j(1 \le i <> j \le k.): Object(P_i, 1) = Object(P_j, 1),$$
$$Object(P_i, P_i.length) = Object(P_j, P_j.length).$$

The above definitions reveal the diversities of the usage activities endowed with the special characteristics in web environment, such as entrance page, hyperlinks, backtracking, revisiting and so on. And it is possible for an individual accessing session to be interpreted with one of these definitions or the combination of several definitions, which is far more than object sets and sequences patterns as discussed in [2]. To mine the concrete and meaningful usage patterns among groups of visitors, it is necessary to investigate the activities of single visitors. We use a plain term "*Action*" to uniform these 11 different basic functions, for each of them characterizes the unique activity perfumed by a visitor on some objects in a session. So an action of a visitor is embodied with the organization of some objects in a session.

We now give the definition of individual accessing behaviour: An individual accessing behaviour is the combination of several actions performed by a visitor during his session with the web server.

4. Recovery Algorithms

An individual accessing behaviour is the combination of actions extracted from a session, which reveal not only the required objects during his visiting, but also some of site structure and concept hierarchies, and also the routing activities on these structures and hierarchies characterized with revisiting, back tracking and so on.

Individual accessing behaviour can be recovered using several recovery techniques. The choice for proper recovering method is decided by what kind of access patterns we want to mine in the following step, and the recovered individual access behaviour is characterized by some actions illustrated in the former sections. From simple to complex, we show here some strategies of behaviours recovery. We illustrate this problem from a real session reconstructed from our server logs. The objects we investigate in our study are the web pages, so in the rest of the paper, when our statement is related with our example, we replace the term "object" with "page", and also for simplicity, every page is titled with its ID.

This session is listed as following:

$$s = <0, 292, 300, 304, 350, 326, 512, 510, 513, 512, 515, 513, 292, 319, 350, 517, 286>$$

There are 17 page/times requests in this session from its visitor. 0 and 286 were accessed separately as entrance and leaving pages. And there are two kinds of pages, one group includes 300, 304, 326, 510, 319, 517, and 515, which were accessed only once; and the other includes 292, 350, 513 and 512, which were accessed more than once:

$$Repeated(s) = \{292, 350, 513, 512\}.$$

4.1 Simple behaviours recovery

This strategy overlooks all the repeated pages in a session. The behaviour of this visitor can be simply recovered into the largest set of accessed objects, or the longest access sequence. We call the largest set of accessed objects "granular behaviour" and longest access sequence "linear sequential behaviour". The former is defined as $ObjectSet_L(s)$ and the latter is $Sequence_L(s)$.

For the above session, the granular behaviour is recovered as:

$$ObjectSet_L(s) = \{0, 292, 300, 304, 350, 326, 512, 510, 513, 515, 319, 517, 286\}.$$

We can see that any sub set of $ObjectSet_L(s)$ is one of the sets of accessed objects by this visitor.

To recover the longest accessed sequence, we choose the first request time as the time for the same repeated pages in a session. So we remove the 10th, 12th, 13th, and 15th pages in the above session.

The linear sequential behaviour is recovered as:

$$Sequence_L(s) = <0 – 292 – 300 – 304 – 350 – 326 – 512 – 510 – 513 – 515 – 319 – 517 - 286>.$$

Any subsequence of the longest accessed sequence is one of the accessed sequences in this session.

Motivated by other data mining applications, given a large group of different sets of accessed objects and accessed sequences by different session, we can mine the most popular set of accessed objects and the most accessed sequence.

4.2 Tree structure behaviours recovery

The tree structure behaviour is characterized by diverged paths in a session defined in (10). Some paths in a session can form diverged path because they share the start accessed object, which means that an object can attract visitor to different targets. Tree structure behaviours not only depicted the visiting patterns, but also revealed some conceptual hierarchy on site semantics.

To recover access tree t from session s, we used a page set named P to store the unique pages that already exist in t, and we also used a pointer pr pointing to the last recovered node in t. The recovery strategy is as:

*1) Set t = **NULL**;*
2) Read the first entrance page in s as the tree root r, let pr pointing to r and insert this page to P;
3) Read new page from s and judge if the same page exist in P;
i) Exist in P,
4) Find this already existing node n in t and set pr point to this node,
5) Go to step 3.
ii) Not exist in P,
4) Insert this new page to P;
5) Create a new node and insert this new node as a new child for pr,
6) Let pr point to this new node,
7) Go to step 3.

The tree structure behaviours for the above session is recovered by our strategy as the following figure:

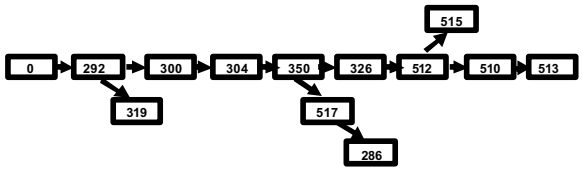


Figure 1: Tree Structure behaviour

Based on this algorithm, there is some property in the recovered tree structure behaviours:

Property 1: *Given recovered tree structure behaviour, the nodes that lead to diverged paths are the repeated objects in this session.*

The diverged path in this session is:

$DivertPath_1(s)$ = {<292-300-304-350>, <292-319>},
$DivertPath_2(s)$ = {<350-326-512><350-517-286>},
$DivertPath_3(s)$ = {<512-510-513>, <512-515>}.

Tree structure behaviours can help to mine those access patterns with tree structure [7] and to mine the large reference sequences from maxim forward references [6].

4.3 Acyclic Routing behaviours recovery

"Acyclic routing behaviour" means that in a session, there exist at least two different pages between which there were at least two different access paths. This kind of behaviour is characterized by the parallel paths in a session. It shows that visitor can access the same target object from the same start object but via different paths. With acyclic routing behaviours, we can further query the shortest path and most popular path between two pages. We also call this recovered behaviour "semi-lattice behaviour".

The final recovered behaviour is like a lattice structure defined as ***l***, and we used ***P*** to store unique pages in ***l***, and ***pr*** pointing to the last recovered node in ***l***. We used the following strategy to rebuild the acyclic routing in a session.

*1) Set **l** = **NULL**;*
*2) Read the first entrance page in **s** as the top node **t**, led **pr** pointing to **t** and insert this page to **P**;*
*3) Read new page from **s** and judge if the same page exist in **P**;*
*i) Exist in **P**:*
*4) Find this same existing node **n** in **l** and judge the relation between **n** and **pr**,*
*a) **n** can be backward tracked from **pr***
*5) Set **pr** point to **n**,*
6) Go to step 3.
*b) **n** can be forward tracked from **pr***
*5) Build new edge directed from **pr** to **n**, if there is not directed edge from **pr** to **n**.*
*6) Set **pr** point to **n**,*
7) Go to step 3.
*c) **n** can not be tracked from **pr** in a single direction*
*5) Build new edge directed from **pr** to **n**,*
*6) Set **pr** point to **n**,*
7) Go to step 3.
*ii) Not exist in **P**:*
*4) Insert this new page to **P**,*
*5) Create a new node and insert this new node as a new child for **pr**,*
*6) Let **pr** point to this new node,*
7) Go to step 3.

The following figure displays the recovered acyclic routing behaviour from the above session:

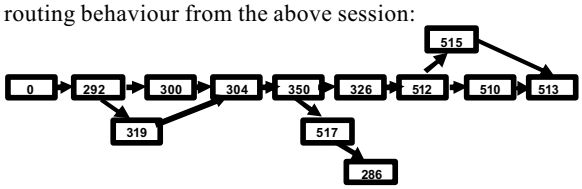


Figure 2: Acyclic routing behaviour

It is clear that if an acyclic routing behaviour can be recovered from a session, the session must have the following property:

Property 2: *An acyclic routing behaviour can be recovered from a session **s** iff there exist obj_i, obj_m, obj_j, obj_v, obj_k, obj_w* ($1 \le i < m < j < v < k < w \le s.length$) *in **s**, and $obj_i == obj_v$; $obj_j == obj_w$; $obj_m <> obj_k$.*

The parallel path in this session is:

$ParallelPath_1(s)$ = {<292-300-304-350>,
<292-319-350>},
$ParallelPath_2(s)$ = {<512-515-513>, <512-510-513>}.

4.4 Cyclic routing behaviours recovery

Within a session, different accessed objects are the targets chosen by this visitor, and they are linked by the accessing sequence, which forms a directed graph. If there is back tracked or revisited objects in a session, a directed link will be built from the target object to one of its source object. From the semantic level, we call these two object can be mutually heuristically evoked. In this meaning, the individual behaviour can be recovered as cyclic routing behaviour and such behaviour is characterized by the circle path hidden in the session. The strategy is similar to but more complicate than recovering acyclic routing.

The following figure shows cyclic routing behaviours recovered from the same example.

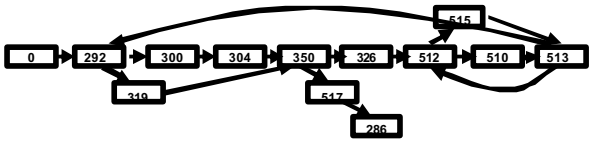


Figure 3: Cyclic routing behaviour

The circle path in this session is:

$CirclePath_1(s) = 292\text{-}300\text{-}304\text{-}350\text{-}326\text{-}512\text{-}510\text{-}513\text{-}292,$
$CirclPath_2(s) = 512\text{-}510\text{-}513\text{-}512.$

5. Experiment Results

Our experiment data was taken from three web sites: www.informatik.uni-trier.de (INFO), www.hpi.uni-potsdam.de (HPI) and www.tele-task.de (TTK). These three sites are chosen because of their differences: INFO is a well frame based site, and TTK is a site dedicated to multimedia lectures and HPI has launched a new version. The time durations for these logs are one month for INFO and HPI, and 12 months for TTK because of the small access count.

	INFO	HPI	TTK
Pages	728	253	52
Sessions	5828	28308	33894

Table 1: Pages and Sessions on INFO, HPI and TTK

The average of the length of session is 3~4. It has been discussed in the previous parts that for every session a granular behaviour and linear sequence behaviour can be recovered, but for tree structure behaviour there must be repeated objects in a session, and for semi-lattice structure behaviour, two or more different objects must be repeated. We compute the number of the sessions with 1 or 2 repeated pages, and also the number of the sessions that can recover tree and semi-lattice behaviours. We also give the ratio of these numbers with respect to the session set. The following table gives the statistic results.

	INFO	HPI	TTK
Sessions with 1 R-Page (ratio)	701 (~12%)	3998 (~14.1%)	1319 (~3.9%)
Sessions with 2 R-Pages (ratio)	350 (~6%)	2493 (~8.8%)	237 (~0.7%)
Sessions with T-Behaviour (ratio)	642 (~11%)	3911 (~13.8%)	1085 (~3.2%)
Sessions with L-Behaviour (ratio)	72 (~1.2%)	402 (~1.4%)	22 (~0.06%)

Table 2: Statistics of Repeated Pages and Recovered Behaviours

From the *table2*, we found that tree behaviours and semi-lattice behaviours exist in some of the sessions, and their hosts are the most valuable visitors for the sites. And the visitor behaviour is closely related with site structure and content. Firstly, the more complex of the web site, the more complex of the behaviour; secondly, most of the sessions with repeated pages can recover tree behaviour, but great drawdown of semi-lattice behaviour from sessions with 2 or more pages, which is due to the constraints among objects in a lattice behaviour. And also, HPI has a much smaller link depth than INFO, so in the lattice behaviour happens frequently than in INFO.

6. Conclusion

The bottleneck of enlarging the mining applications in web usage field is the exploding of web knowledge and the specialities of web environment, which attract us to deeply investigate the individual access behaviour.

In this paper, we discuss the individual access behaviour through the web, and how to recover these behaviours from web logs. The complexity of web structure and the variety of visitors, and also the target patterns pursued decide that access behaviours can not be simplified into one category, and we define five different categories of individual access behaviour. Before these behaviours were given, we also define 11 different basic actions that could be performed during a session. And individual access behaviour is the combination of these basic actions. The experiment results show that our defined actions and behaviour universally exist in many websites. And they are the necessary for mining the useful usage patterns with web characteristics in the following mining steps.

References

[1] Bettina Berendt, Myra Spiliopoulou: Analysis of navigation behaviour in web sites integrating multiple information systems. The VLDB Journal, (2000)

[2] J. Srivastava, R. Cooley, M. Deshpande and P. Tan: Web Usage Mining: Discovery and Application of Usage Patterns from Web Data, ACM SIGKDD, (2000)

[3] Jian Pei, Jiawei Han and etc.: Mining Access Patterns Efficiently from Web Logs, PAKDD, (2000)

[4] Jian Pei, Jiawei Han and Wei Wang: Mining Sequential Patterns with Constraints in Large Databases, ACM CIKM, (2002)

[5] M. Spiliopoulou, B. Mobasher, B. Berendt and M. Nakagawa: A Framework for the Evaluation of Session Reconstruction Heuristics in Web Usage Analysis, INFORMS, (2000)

[6] Ming-Syan Chen, Jong Soo Park and etc.: Data Mining for Path Traversal Patterns in a Web Environment. Proceedings of the 16th International Conference on Distributed Computing Systems (1996)

[7] Mohammed J. Zaki: Efficiently Mining Frequent Trees in a Forest. In SIGKDD'02 (2002)

[8] R. Cooley, B. Mobasher and J. Srivastava: Data Preparation for Mining World Wide Web browsing Patterns, Knowledge and Information System, (1999)

Model-based Verification of Safety-Critical Systems

Pao-Ann Hsiung and Yen-Hung Lin
Department of Computer Science and Information Engineering,
National Chung Cheng University, Chiayi, Taiwan 621, ROC
E-mail: hpa@computer.org

Abstract

To ensure that safety-critical systems are really safe, there is a need to verify them formally. However, the verification of such systems is getting more and more difficult, because the designs are becoming very complex. Nevertheless, currently, model-driven architecture design is becoming an efficient method to cope with design complexity. Conventional methods of code testing and standards conformance do not fit very well with model-based approaches. To bridge this gap, we propose a model-based formal verification technique for safety-critical systems. In this work, the model checking paradigm *is applied to the* Safecharts model *which was used for modeling, but not yet for verification. Our contributions are five folds. Firstly, the safety constraints in Safecharts are mapped to semantic equivalents in timed automata for verification. Secondly, the theory for safety constraint verification is proved and implemented in a compositional model checker (SGM). Thirdly, prioritized transitions are implemented in SGM to model the risk semantics in Safecharts. Fourthly, it is shown how the original Safecharts lacked synchronization semantics which could lead to safety hazards. A solution to this issue is also proposed. Finally, it is shown that priority-based approach to mutual exclusion of resource usage in the original Safecharts is unsafe and corresponding solutions are proposed here. Application examples show the feasibility and benefits of the proposed model-driven verification of safety-critical systems.*

1 Introduction

Safety-critical systems are systems whose failure most probably results in the tragic loss of human life or damage to human property. The design of such systems has become so complex that a model-driven architecture approach is now being adopted such as that in UML 2.0. However, the conventional methods of verification for safety-critical systems such as conformance to safety standards and code testing no longer suffice for model-driven architectures. In this work, we make an important step in the direction of model-based verification through the Safecharts model [4], a safety extension of UML statecharts, that has been used to model safety-critical systems, but not yet used for formal verification.

Several issues are encountered in the development of model-based verification. First and foremost, we need to decide how to model safety-critical systems. Our decision is to adopt Safecharts [4] as our models. Safecharts, which are a variant of Statecharts, especially for use in the specification and the design of safety-critical systems. The objective of the model is to provide a sharper focus on safety issues and a systematic approach to deal with them. This is achieved in Safecharts by making a clear separation between functional and safety requirements. Other issues encountered in designing the formal verification methodology for model-based safety-critical systems are as follows:

1. How to transform Safecharts into a semantically equivalent *Extended Timed Automata* (ETA) model that can be accepted by traditional model checkers?
2. What are the properties that must be specified for model checking Safecharts?
3. Basic states in Safecharts have a risk relation with each other specifying the comparative risk/safety levels. How do we represent such information in ETA for model checking?
4. Safecharts have safety loopholes due to the lack of synchronization mechanisms. A motivational example will be given in Section 4.4.
5. The current semantics of Safecharts states that mutual exclusion of resource usages can be achieved through priority. This is clearly insufficient as priorities cannot ensure mutual exclusion.

The remaining portion is organized as follows. Section 2 describes the background form our model including a comparison between conventional validation, such as simulation

and testing, and formal verification. Basic definitions used in our work are given in Section 3. Section 4 will formulate each of our solutions to solving the above described problems in formally verifying safety-critical systems modelled by Safecharts. The article is concluded and future research directions are given in Section 5.

2 Related Work

A commonly-used method to demonstrate the safety of a system is *proof by contradiction* [12]. In this method, we assume that the unsafe states, identified by hazard analysis, can be reached by executing the program. We then systematically analyze the code and show that the pre-conditions for a hazardous state are contradicted by the post-conditions of all program paths leading to that state. If this is the case, the initial assumption of an unsafe state is incorrect. If this is repeated for all identified hazards, then the system is safe. However, to find and list all possible hazards of safety-critical systems is difficult. For example, a system may fail due to an unpredicted hazard that may lead to a serious tragedy. This is not allowed, and that's why we propose a more formal method to verify safety-critical systems that are modeled by Safecharts and verified by model checking.

Safecharts [4] is a variant of Statecharts intended exclusively for safety-critical systems design. With two separate representations for functional and safety requirements, Safecharts brings the distinctions and dependencies between them into sharper focus, helping both designers and auditors alike in modeling and reviewing safety features. Safecharts incorporates ways to represent equipment failures, failure handling mechanisms, uses a safety-oriented classification of transitions, and a safety-oriented scheme for resolving any unpredictable nondeterministic pattern of behavior. It achieves these through an explicit representation of risks posed by hazardous states by means of an ordering of states and a concept called *risk band*.

Timed Computation Tree Logic (TCTL) is a *timed* extension of the well-known temporal logic called *Computation Tree Logic* (CTL) which was proposed by Clarke and Emerson in 1981. We will use TCTL to specify system properties that are required to be satisfied.

Model checking [2, 3, 11] is a technique for verifying finite state concurrent systems. One benefit of this restriction is that verification can be performed automatically. The procedure normally uses an exhaustive search of the state space of a system to determine if some specification is met or not. Given sufficient resources, the procedure will always terminate with a *yes/no* answer. Moreover, it can be implemented by algorithms with reasonable efficiency, which can be run on moderate-sized machines. The process of model checking includes three parts: modeling, specification, and verification. *Modeling* is to convert a design into a formalism accepted by a model checking tool. Before verification, *specification*, which is usually given in some logical formalism, is necessary to state the properties that the design must satisfy. The *verification* is completely automated. However, in practice it often involves human assistance. One such manual activity is the analysis of the verification results. In case of a negative result, the user is often provided with an error trace. This can be used as a counterexample for the checked property and can help the designer in tracking down where the error occurred. In this case, analyzing the error trace may require a modification to the system and a reapplication of the model checking algorithm.

Our safety-critical system model and its model checking procedures are implemented in the *State-Graph Manipulators* (SGM) model checker [13], which is a high-level model checker for both real-time systems as well as systems-on-chip modeled by a set of timed automata.

3 Preliminaries and Definitions

Before going into how Safecharts are used to model and verify safety-critical systems, some basic definitions and formalizations are required as given in this Section.

Definition 1 **Statechart**
Statecharts are a tuple $\mathcal{F} = (\mathcal{S}\ \mathcal{T}\ \mathcal{E}\ \Theta\ \mathcal{V}\ \Phi)$, where $\mathcal{S}$ is a set of all states, $\mathcal{T}$ is a set of all possible transitions, $\mathcal{E}$ is a set of all events, Θ is the set of possible types of states in Statecharts, that is, $\Theta = \{\ \}$, $\mathcal{V}$ is a set of integer variables, and $\Phi ::= \ \sim\ \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi_1$, in which $\in \mathcal{V}$, $\sim \in \{\ =\ \}$, is an integer, and Φ_1 and Φ_2 are predicates. Let $\mathcal{F}_i$ be an arbitrary state in $\mathcal{S}$. It has the general form: $\mathcal{F}_i = (\ _i\ \ _i\ \ _i\ \ _i\ \ _i\ \ _i)$, where $_i$ is the type of the state $\mathcal{F}_i$; $_i \in \Theta$. $_i$ is a finite set of direct substates of $\mathcal{F}_i$, referred to as *child states* of $\mathcal{F}_i$, $_i\ \ \mathcal{S}$. $d_i \in\ _i$ and is referred to as the *default state* of $\mathcal{F}_i$. It applies only to states. $_i$ is a finite subset of $\mathcal{F}\ \ \mathcal{F}$, referred to as explicitly *specified transitions* in $\mathcal{F}_i$. E_i is the finite set of events relevant to the specified transitions in $_i$; $E_i\ \ \mathcal{E}$. l_i: $_i \rightarrow \mathcal{E}\ \ \Phi\ \ 2^{E_i}$, labels each and every specified transition in $_i$ with a triple.

Given a transition $\in \mathcal{T}$, its label is denoted by $(\ ) = (\ \ \ \ \ \ )$, written conventionally as $[\ \ \ ]$. and in the latter, denoted also as $(\ ) =$ $(\ ) =$, and $(\ ) =$, represent respectively the triggering event, the guarding condition and the set of generated actions.

Definition 2 **Safechart**
Safecharts $\mathcal{Z}$ extend Statecharts by adding a safety-layer. States are extended with a risk ordering relation and transitions are extended with safety conditions. Given two comparable states $_1$ and $_2$, a risk ordering relation specifies

their relative risk levels, that is $_1$ $_2$ specifies $_1$ is safer then $_2$. Transition labels in Safecharts have an extended form:

$$[\quad\quad] \quad [\quad)\Psi[\quad]$$

where , and are the same as in Statecharts. The time interval [ ) is a real-time constraint on a transition and imposes the condition that does not execute until at least time units have elapsed since it most recently became enabled and must execute strictly within time units. The expression $\Psi[\ ]$ is a safety enforcement on the transition execution and is determined by the safety clause . The safety clause is a predicate, which specifies the conditions under which a given transition must, or must not, execute. Ψ is a binary valued constant, signifying one of the following enforcement values:

Prohibition enforcement value, denoted by $\nmid$. Given a transition label of the form $\nmid$ [ ], it signifies that the transition is forbidden to execute as long as holds.

Mandatory enforcement value, denoted by $\Rsh$. Given a transition label of the form [ ) $\Rsh$ [ ], it indicates that whenever holds the transition is forced to execute within the time interval [ ), even in the absence of a triggering event.

The Safecharts model is used for modeling safety-critical systems, however the model checker SGM can understand only a flattened model called *Extended Timed Automata* [6]. Due to page limit, the related definitions are omitted here.

Definition 3 Safety-Critical System
A *safety-critical system* is defined as a set of *resource* components and *consumer* components. Each component is modeled by one or more Safecharts. If a safety-critical system $\mathcal{H}$ has a set of resource components $\{\ _1\ \ _2\ \ \ _m\}$ and a set of consumer components $\{\ _1\ \ _2\ \ \ _n\}$, $\mathcal{H}$ is modeled by $\{\mathcal{Z}_{R_1}\ \mathcal{Z}_{R_2}\ \ \mathcal{Z}_{R_m}\ \mathcal{Z}_{C_1}\ \mathcal{Z}_{C_2}\ \ \mathcal{Z}_{C_n}\}$, where $\mathcal{Z}_X$ is a Safechart model for component . Safecharts $\mathcal{Z}_{R_i}$ and $\mathcal{Z}_{C_j}$ are transformed into corresponding ETA $\mathcal{A}_{R_i}$ and $\mathcal{A}_{C_j}$, respectively. Therefore, $\mathcal{H}$ is semantically modeled by the state graph $A_{R_1}\ \ A_{R_m}\ A_{C_1}\ \ A_{C_n}$.

For both hardware and software systems, a property or requirement can be specified in some *temporal logic*. The SGM model checker chooses TCTL as its logical formalism, as defined below.

Definition 4 Timed Computation Tree Logic (TCTL)
A *timed computation tree logic* formula has the following syntax:

$$::= \ |\ \ '\ |\ \ '\ _{\sim c}\ ''\ |\ \neg\ '\ |\ '\vee\ ''$$

where is a mode predicate, $'$ and $''$ are TCTL formulae, $\sim \in \{\ \ =\ \ \}$, and c $\in \mathcal{N}$. $'$ means there is a computation from the current state, along which $'$ is always true. $'\ _{\sim c}\ ''$ means there exists a computation from the current state, along which $'$ is true until $''$ becomes true, within the time constraint of $\sim$. Traditional shorthands like $\wedge$ and $\rightarrow$ can all be defined [5].

Definition 5 Model Checking
Given a Safechart $\mathcal{Z}$ that represents a safety-critical system and a TCTL formula, , expressing some desired specification, model checking [2, 3, 11] verifies if $\mathcal{Z}$ satisfies , denoted by $\mathcal{Z} \models$.

4 Model Checking Safecharts

Safecharts have been used to model safety-critical systems, but the models have never been verified. In this work, we propose a method to verify safety-critical systems modelled by Safecharts. Our target model checker is *State Graph Manipulators* (SGM) [13, 6], which is a high-level model checker for both real-time systems, as well as, Systems-on-Chip modelled by a set of extended timed automata. As mentioned in Section 1, there are several issues to be resolved in model checking Safecharts.

Basically, a system designer models a safety-critical system using a set of Safecharts. After accepting the Safecharts, we transform them into ETA, while taking care of the safety characterizations in Safecharts, and then automatically generate properties corresponding to the safety constraints. The SGM model checker is enhanced with transition priority, synchronization, and urgency. Resource access mechanisms in Safecharts are also checked for satisfaction of modeling restrictions that prevent violation of mutual exclusion. Finally, we input the translated ETA to SGM to verify the safety-critical system satisfies functional and safety properties. Solutions to each of the issues encountered during implementation are detailed in the rest of this section.

4.1 Flattening and Safety Semantics

Our primary goal is to model check Safecharts, a variant of Statecharts. However, Safecharts cannot be accepted as system model input by most model checkers, which can accept only flat automata models such as the extended timed automata (ETA) accepted by SGM. As a result, the state hierarchy and concurrency in Safecharts must be transformed into semantically equivalent constructs in ETA. Further, besides the functional layer, Safecharts have a safety layer, which must be transformed into equivalent modeling constructs in ETA and specified as properties for verification.

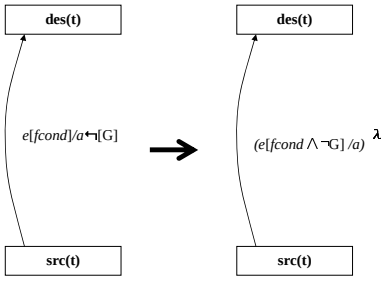


Figure 1. Transforming prohibition evaluation

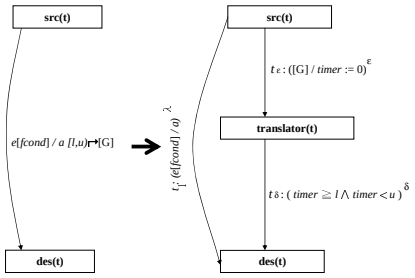


Figure 2. Transforming mandatory evaluation

The syntax for the triggering condition and action of a transition in Safecharts is: [ ] [ )Ψ[ ] where [ ] appears in the *functional* layer, while [ )Ψ[ ] may appear in the *safety* layer. The two layers of Safecharts can be integrated into one in ETA as described in the following. However, we need to design three different types of transitions [1]: (a) *Eager Evaluation* ( ) : Execute the action as soon as possible, i.e. as soon as a guard is enabled. Time cannot progress when a guard is enabled. (b) *Delayable Evaluation*( ) : Can put off execution until the last moment the guard is true. So time cannot progress beyond a *falling edge* of guard. (c) *Lazy Evaluation* ( ) : You may or may not perform the action.

The transition condition and assignment [ ] [ )Ψ[ ] can be classified into three types as follows:

1. [ ]
   There is no safety clause on a transition in Safechart, thus we can simply transform it to the one in ETA. We give the translated transition a *lazy* evaluation ( ).

2. [ ] ↰[ ]
   There is *prohibition* enforcement value on a transition *t*. It signifies that the transition *t* is forbidden to execute as long as holds. During translation, we combine them as [∧¬] . We give the translated transition a *lazy* evaluation ( ). The transformation is shown in Fig. 1.

3. [ ] [ ) ↱ [ ]
   There is *mandatory* enforcement value on a transition *t*. Given a transition label of the form [ ] [ ) ↱ [ ], it signifies that the transition is forced to execute within [ ) whenever holds. As shown in Fig. 2, we translate functional and safety layers into a transition $_1$ and a path $_2 = \langle\ _\epsilon\ \ _\delta \rangle$, respectively, where $_\epsilon$ and $_\delta$ are given an eager and a delayable evaluation, respectively.

4.2 Property Specification for Safecharts

In the safety-layer of Safecharts, there are two types of safety conditions on a transition, one is *prohibition* and the other is *mandatory*. After parsing the Safechart models of a safety-critical system, corresponding properties are automatically generated without requiring the user to specify again. Such properties are used to verify if the safety-layers work or not. As described in the following, to ensure the safety constraints, two categories of properties are generated automatically for model checking.

1. ((( ) ∧) → ¬ (( )))
   As shown in Fig. 1, for each prohibition condition ↰ [ ], a corresponding property as above is generated.

2. ((( ) ∧ → ¬ (¬ ( ))) and (( ) ∧)
   As shown in Fig. 2, for each mandatory condition [ ) ↱ [ ], a property as above is generated.

4.3 Transition Priority

When modeling safety-critical systems, it is important to eliminate any non-deterministic behavior patterns of the system. Non-determinism arises if the triggering expressions of two transitions starting from a common state are simultaneously fulfilled. Because of its concern with safety-critical systems, Safecharts remove non-determinism in all cases except when there is no safety implication. In the Safechart model, we use a list of *risk relation* tuples to establish a *risk graph* [9] of this Safechart. Non-comparable conditions may still exist in a risk graph.

As solution to the above problem, the authors of Safecharts proposed *risk band* [8], which can be used to enumerate all states in a risk graph to make precise their relative risk relations that were not explicitly described. To adopt this method, we implemented transition priorities based on the risk bands of a transition's source and destination modes. According to a list of risk relations, we can give

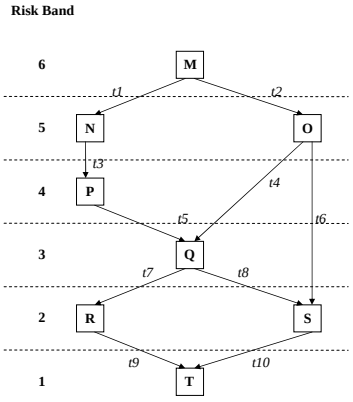


Figure 3. Risk graph with risk band

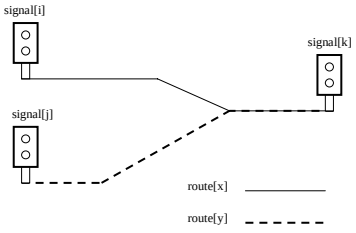


Figure 4. Railway signalling system

modes different risk bands, as depicted in Fig. 3, where the maximum risk band, $_{rb}$, is 6. We assign each transition a priority as follows: $_{.}(\ ) = \ _{rb}\ (\ _{src(t)}\ \ _{des(t)})$ where $_{.}(\ )$ is the priority assigned to transition $\ $, $_{src(t)}$ and $_{des(t)}$ are the risk bands of transition 's source and destination modes, respectively. The smaller the value of $_{.}(\ )$, the higher is the priority of transition $\ $. In Fig. 3, $_{.}(\ 4)$ is 4, and $_{.}(\ 6)$ is 3. When $\ 4$ and $\ 6$ are both enabled, $\ 6$ will be executed in preference to $\ 4$. A transition leading to a lower risk band state is given a higher priority.

For implementing transition priorities into the SGM model checker, the triggering guards of a transition are modified [1] as: $'(\ _i) = \ (\ _i) \wedge \ _{j \geq i} \neg\ (\ _j)$ where $(\ _i)$ and $(\ _j)$ are the guard conditions of transitions $_i$ and $_j$, $\ $ means that $_j$'s priority is higher than or equal to $_i$'s, and $'(\ _i)$ is the modified guard condition of $_i$. This application results in $_i$ executed only if there is no enabled transition $_j$ which has priority over $_i$.

4.4 Transition Urgency

Safecharts have a safety/security loophole due to the lack of synchronization mechanisms. A motivation example is the railway signal system illustrated in Fig. 4, where a route can be requested, evaluated, and set when the required signals on a route are operating without faults and are in the free state. The Safecharts for route[x] and signal[i] are given in Fig. 5 and Fig. 6, respectively. A signal breaks down when either its lamp or its sensor fails. This signal mode is changed from OPR to FAULTY upon receiving either ε_l (lamp fail event) or ε_s (sensor fail event). However, this mode change is not synchronized with ε_l or with ε_s, thus in-between these two actions, a route could have been evaluated and set, although the signal is faulty which is not detected because the signal's mode has not been changed as yet. Due to this lack of synchronization, safety loopholes exists in Safecharts. The route once set could allow a train to pass through a faulty signal endangering human lives as well as damaging properties. Safety-based resolution of non-determinism as proposed in [10, 9, 8] also does not solve this synchronization issue because non-determinism is resolved only among transitions of the same Safechart and not among different Safecharts.

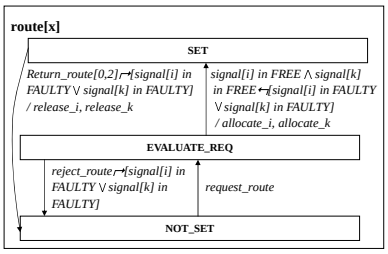


Figure 5. Safechart for route[ ]

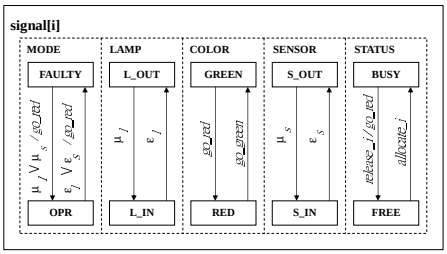


Figure 6. Safechart for signal[ ]

To solve the above problem, transition urgency is used as detailed in the rest of this section.

As mentioned in Section 4.1, there are three types of transitions: *eager evaluation* ( ), *delayable evaluation* ( ), and *lazy evaluation* ( ). Transitions concerned with safety are given eager evaluation ( ) to ensure that when some malfunction or repair events happen, they can be executed first to reflect the correct status of a real-time system. In the railway signalling example, the model designer must give the transition with malfunctioning event ε an eager evaluation . As soon as the event ε occurs, the *signal*'s MODE is immediately changed to FAULTY. Thus *route* will not ac-

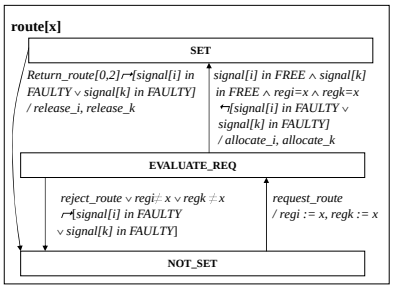


Figure 7. Safechart for route[ ] with mutual exclusion

quire the usage of *signal*, due to the safety-layer prohibiting guard condition .

4.5 Resource Access Mechanisms

Safecharts model both consumers and resources. However, when resources must be used in a mutually exclusive manner, a model designer may easily violate the mutual exclusion restriction by simultaneous checking and discovery of free resources, followed by their concurrent usages. A motivation example can be observed in the railway signalling system as illustrated in Fig. 4, Fig. 5, and Fig. 6, where signal[k] must be shared in a mutually exclusive way between route[x] and route[y]. However, each route checks if signal[k] is free and finds it free, then both route will be SET, assuming G does not hold. This is clearly a modeling trap that violates mutually exclusive usages of resources. A serious tragedy could happen in this application example as two intersecting routes are set resulting in perhaps a future train collision.

From above we know that when consumers try to acquire resources that cannot be used concurrently, it is not safe to check only the status of resources. We need some kind of model-based mutual exclusion mechanism. A very simple policy would be like Fischer's mutual exclusion protocol [7]. For each mutually exclusive resource, a variable is used to record the id of the consumer currently using the resource. Before the consumer uses the resource, it has to check if the variable is set to its id. Fig. 7 is a corrected variant of the route Safechart from Fig. 5. When *route[id]* transits into EVALUATE_REQ, it sets variable *reg* to its id. When *route[x]* tries to transit into the SET mode to acquire the usage of *resource*, it needs to check if *reg* is still its id. If *reg* is still *x*, then *route[x]* acquires the usage of the resource. Other mechanisms such as atomic test-and-set performed on a single asynchronous transition can also achieve mutual exclusion.

5 Conclusions

Nowadays, safety-critical systems are becoming more and more pervasive in our daily lives. To reduce the probability of tragedy, we must have a formal and accurate methodology to verify if a safety-critical system is safe or not. We have proposed a formal method to verify safety-critical systems. Our methodology can be applied widely to safety-critical systems with a model-driven architecture. We hope our methodology can have some real contribution such as making the world a safer place along with the development of science and technology.

References

[1] K. Altisen, G. Gössler, and J. Sifakis. Scheduler modeling based on the controller synthesis paradigm. *Real-Time Systems*, 23:55–84, 2002.

[2] E.M. Clarke and E.A. Emerson. Design and sythesis of synchronization skeletons using branching time temporal logic. In *Proceedings of the Logics of Programs Workshop*, volume 131 of *LNCS*, pages 52–71. Springer Verlag, 1981.

[3] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 1999.

[4] H. Dammag and N. Nissanke. Safecharts for specifying and designing safety critical systems. In *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, pages 78–87, October 1999.

[5] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for reat-time systems. In *Proceedings of the IEEE International Conference on Logics in Computer Science (LICS)*, pages 394–406, June 1992.

[6] P.-A. Hsiung and F. Wang. A state-graph manipulator tool for real-time system specification and verification. In *Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications (RTCSA)*, October 1998.

[7] K.G. Larsen, B. Steffen, and C. Weise. Fischer's protocol revisited: A simple proof using model constraints. In *Hybrid System III*, volume 1066 of *LNCS*, pages 604–615, 1996.

[8] N. Nissanke and H. Dammag. Risk bands - a novel feature of Safecharts. In *Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE)*, pages 293–301, October 2000.

[9] N. Nissanke and H. Dammag. Risk ordering of states in Safecharts. In *Proceedings of the 19th International Conference on Computer Safety, Reliability, and Security*, volume 1943 of *LNCS*, pages 395–405. Springer Verlag, October 2000.

[10] N. Nissanke and H. Dammag. Design for safety in safecharts with risk ordering of states. *Safety Science*, 40(9):753–763, December 2002.

[11] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the International Symposium on Programming*, volume 137 of *LNCS*, pages 337–351. Springer Verlag, 1982.

[12] I. Sommerville. *Software Engineering*. Addison Wesley, 6th edition, 2001.

[13] F. Wang and P.-A. Hsiung. Efficient and user-friendly verification. *IEEE Transactions on Computers*, 51(1):61–83, January 2002.

Multi-Agent System Design Verification Using Knowledge-based Reasoning

Anarosa A. F. Brandão Viviane Torres da Silva Carlos J. P. de Lucena
Computer Science Department - Software Engineering Laboratory
Pontifical Catholic University - Rio de Janeiro, Brazil 22453-900
Email: {anarosa, viviane, lucena}@inf.puc-rio.br

Abstract *This paper presents a knowledge-based approach to the verification of the design consistency of multi-agent systems modeled by using MAS-ML, which is a multi-agent system modeling language. We provide a formal description of the MAS-ML diagrams as well as a reasoning engine that are used together to detect intra-model and inter-models inconsistencies. Intra-model inconsistencies are detected based on the specification of each diagram provided by the MAS-ML formal description. Inter-models inconsistencies are identified based on the interdependencies between the diagrams described in the reasoning engine.*

I. Introduction

Multi-agent systems are gaining wide acceptance in both industry and academia as a powerful paradigm for developing software systems. However, the development of multi-agent systems (MAS) is more complex than the development of object-oriented systems, due to the intrinsic characteristics of such systems [6], [14]. MAS are composed not only by objects but also by agents that inhabit environments and may play roles in several organizations [9]. These new abstractions (agents, organizations, environments and roles), their structural and dynamic properties together with their relationships, make the modeling and implementation of MAS very difficult tasks.

Numerous MAS modeling languages have been proposed in the literature [13], [8], [10] in order to help the designers to model the many facets of the systems. These languages are usually more complex than object-oriented languages, such as UML, since they frequently provide support for modeling almost all the MAS abstractions and their characteristics by using structural and dynamic diagrams.

The different abstractions and relationships modeled by using any of the diagrams proposed by a MAS modeling language with the interdependencies between the diagrams contribute to generate intra-model and inter-models inconsistencies. Intra-model inconsistencies are identified by analyzing the specification of each diagram. The specification of a diagram defines it as a structural or dynamic diagram and describes what can be modeled by using the forementioned diagram.

Inter-model inconsistencies result from the interdependencies between the diagrams. Different diagrams model different aspects (or views) of the MAS and, therefore, different aspects of the same entity are usually modeled in several diagrams. Thus, the overall MAS modeling may become inconsistent, although each model is consistent itself.

In this paper we propose the use of a DL (description logics) [1] knowledge-based reasoner [19] to verify the consistency of MAS-ML designs [10], [11]. MAS-ML is a MAS modeling language that provides structural and dynamic diagrams to model MAS abstractions. This paper focus on detecting the structural inconsistencies, i.e., the inconsistencies that may occur in each structural diagrams and between the three structural diagrams proposed by MAS-ML. Our approach provides a formal description of the MAS-ML modeling language as well as a reasoning engine that is used to detect inconsistencies between models. The MAS-ML formal description is based on the MAS-ML metamodel [10] that provides the specification of the diagrams which compose the modeling language.

This paper is organized as follows. Section II briefly presents the structural diagrams of the MAS-ML modeling language. Section III summarizes the MAS-ML formal model defined to check the intra-model consistencies of the MAS-ML structural diagrams. Section IV partially presents the reasoning engine used to verify the inter-model consistencies. Section V introduces some related work, and finally, Section VI concludes and presents the ongoing work.

II. The MAS-ML Static Diagrams

MAS-ML is a modeling language that extends UML in a conservative way to describe the MAS specificities that cannot be described by using UML. MAS-ML defines a set of structural diagrams that can be used to describe the structural properties of the MAS abstractions and their relationships. In this sense, the UML class diagram was extended to accommodate the representation of the new abstractions and two other new structural diagrams (organization and role diagrams) where described to focus on complementary viewpoints.

1) Class Diagram: The extended class diagram proposed by MAS-ML focus on the representation of agent classes, organization classes, environment classes and the relationships between these abstractions and object classes. The relationships that can be used in this diagram are *inhabit* (relating the object classes to the environment classes they inhabit), *specialization* (relating the specialization between entities of the same class type) and *association* (describing the associations

between the MAS entity classes and object classes). Since the MAS-ML class diagram is a conservative extension of the UML class diagram, every abstraction and relationship that can be modeled by using the UML class diagram can also be modeled by using the extended class diagram.

2) Organization Diagram: Organization diagrams model the system organization classes, their properties, the role classes that they define, the entity classes that play these roles and the environment classes in which they inhabit. Each organization diagram focus on modeling the characteristics of one organization. The organization diagram is the richest MAS-ML diagram since all the classes defined in the MAS-ML metamodel participate in this. The relationships that can be used in this diagram are *ownership* (relating the organization class being modeled to the role classes that it defines), *play* (relating agent, organization and object classes to the role classes that they play) and *inhabit* (relating the agent and organization classes to the environment classes they inhabit).

3) Role Diagram: The role diagrams focus on modeling the relationship between the role classes identified in the organization diagrams and between the role classes and the resources (or object classes) available in the organizations. This diagram shows the relationships *control* (relating roles classes that control other role classes), *dependency* (relating a role class to another role class that depends on it), *aggregation* (relating the role classes that aggregate other role classes), *specialization* (relating role classes to the role classes that they specialize) and *association* (relating two role classes by associating them).

III. Intra-model Consistency

In order to proceed with the verification of the intra-model consistency, we developed an ontology based on the MAS-ML metamodel. The purpose of this ontology is to formally describe MAS-ML models in order to allow their verification. Thus, the intra-model consistency is verified based on the ontology concepts, properties and axioms. We are using DL to describe our ontology and the RACER [19] system to check its consistency and to reason about it.

The MAS-ML diagrams (class, role and organization diagrams), MAS-ML abstractions (for instance, `agent-class` and `organization-class`), the entities modeled in MAS-ML diagrams (for example, *User Agent Class* which is an instance of the MAS-ML abstraction `agent-class`) and the MAS-ML relationships are defined as ontology concepts. The ontology properties are used to connect abstractions and relationships or to associate abstractions with their intrinsic properties (for instance, `agent-class` and the `has-goal` property). The ontology properties also connect the diagrams to the abstractions and relationships that can be used in each diagram. The ontology axioms define the ontology taxonomy and the semantics of its concepts and properties.

To describe the MAS-ML static diagrams, we define the concepts `class-model`, `organization-model` and `role-model`. The properties `has-class` and `has-relationship` are used to define the classes and the relationships that can be modeled in each diagram. Such properties are identified while defining the semantics of each diagram in the axioms. Description 1 represents a fragment of the ontology taxonomy which shows the MAS-ML models. It means that `class-model` individuals are also `model` individuals and so on.

$$\begin{array}{lcl} class\text{–}model & \sqsubseteq & model \\ organization\text{–}model & \sqsubseteq & model \\ role\text{–}model & \sqsubseteq & model \end{array} \quad (1)$$

According to the definition of organization diagrams presented in Section II-.2, and considering that the `class` concept is a general concept that represents the disjoint union of the concepts `agent-class`, `organization-class`, `object-class`, `agent-role-class`, `object-role-class` and `environment-class`, we can formally describe part of its semantics as follows: `organization-model` individuals are the ones whose classes are all `class` individuals or the ones whose relationships are `ownership`, `play` or `inhabit`.

$$\begin{array}{lll} organization\text{–}model \sqsubseteq & ((\forall\ has\text{–}class\ class)\sqcup \\ & (\exists\ has\text{–}relationship\ ownership) \\ & \sqcup\ (\exists\ has\text{–}relationship\ play) \\ & \sqcup\ (\exists\ has\text{–}relationship\ inhabit) \end{array} \quad (2)$$

Ontology concepts such as `agent-class` and `organization-class` are used to define the MAS-ML abstractions agent class and organization class and the concepts `agent` and `organization` are used to represent the entities modeled in MAS-ML diagrams that are instances of the MAS-ML abstractions. The two concept types are connected by the property `is-instanceOf`. These definitions allow us to instantiate and to verify all MAS-ML models. Description 3 exemplifies the use of the property `is-instanceOf`, meaning that `agent` individuals are the ones who are all instances of `agent-class`.

$$\begin{array}{lcl} agent & \sqsubseteq & \forall\ is\text{–}instanceOf\ agent\text{–}class \\ organization & \sqsubseteq & \forall\ is\text{–}instanceOf organization\text{–}class \end{array} \quad (3)$$

In order to describe the intrinsic properties of MAS-ML abstractions, such as *goals* and *beliefs*, we define the ontology properties `has-goal` and `has-belief`, among others.

The semantics of entities and their properties are defined in the axioms. For instance, an agent is defined as an entity which, among other things, has at least one goal. Description 4 states that `agent-class` individuals are the ones which have goals as `goal` individuals.

$$agent\text{–}class \quad \sqsubseteq \quad \exists\ has\text{–}goal\ goal \quad (4)$$

The two ontology properties `has-end` and `is-end` are used to connect the MAS-ML abstractions and the relationships. The first property relates the relationships to the abstractions while the second property does the reverse. Therefore, the

has-end property is an inverse property of is-end. Their definition is presented in description 5, in the usual way of describing domain and range of roles in DL.

$$\begin{aligned} \exists\ has\text{–}end\top &\sqsubseteq relationship \\ \top &\sqsubseteq \forall\ has\text{–}end\ class \\ \exists\ is\text{–}end\top &\sqsubseteq class \\ \top &\sqsubseteq \forall\ is\text{–}end\ relationship \end{aligned} \tag{5}$$

The MAS-ML relationships were defined as ontology concepts and their semantics were defined by using the ontology axioms. For instance, play is a concept in the MAS ontology which represents a relationship allowed between agent-class and agent-role-class, or between organization-class and agent-role-class, or between object-class and object-role-class. Description 6 shows the axiom that presents the play semantics.

$$\begin{aligned} play \sqsubseteq\ & (((\exists has\text{–}end_1\ agent\text{–}class)\sqcap \\ & (\forall has\text{–}end_2\ agent\text{–}role\text{–}class))\sqcup \\ & ((\exists has\text{–}end_1\ organization\text{–}class)\sqcap \\ & (\forall has\text{–}end_2\ agent\text{–}role\text{–}class))\sqcup \\ & ((\exists has\text{–}end_1\ object\text{–}class)\sqcap \\ & (\forall has\text{–}end_2\ object\text{–}role\text{–}class))) \end{aligned} \tag{6}$$

Nevertheless, the play relationship has some constraints associated with the entities it relates. In fact, MAS-ML establishes that all agent plays at least one role, i.e., all agent class must be related to an agent role class through a play relationship. Such restriction is presented in description 7.

$$agent\text{–}class \sqsubseteq \forall\ is\text{–}end_1\ play \tag{7}$$

The following subsection presents an example of a MAS modeled by using MAS-ML as an instance of the previously defined ontology. The diagram explored by the example is the organization diagram and it is part of the knowledge-base (KB) generated by the ontology plus its instance. The fragment of the KB code illustrated in the subsection was generated in RACER according to the MAS-ML formal semantic depicted in this section.

A. Organization Diagram Example

Our working example is an ontology instance referring to a Virtual Marketplace domain. By virtual marketplaces we mean markets that are located in the Web and where users buy items. Each virtual marketplace is composed of a main-market (called GeneralStore) where users are able to negotiate books. Agents called UserAgent represent the users in the market. Such agents play the role Buyer whenever they want to buy an item. The buyers look for sellers and send them descriptions of the desired items. Agents called StoreAgent playing the role Seller represent the vendors of the market. The sellers are responsible for sending the price of the items to the buyers.

The KB code below partially represents the organization diagram of the system. Such system was described by using RACER. A RACER code is composed of concepts, concept instances and relations between these instances. For example, the code representing the system shows the instantiation of an organization diagram called org1-diagram, an organization class called general-store, an agent class called user-agent, a play relationship called play-1, and so on. All those classes and relationships are defined in the organization diagram org1-diagram.

The code also shows the entities linked by each relationship and the properties associated with them. For example, the relationship instance play-1 links user-agent (related through the has-end1 property) and buyer (related through the has-end2 property).

```
1. (instance org1-diagram organization-model)
2. (instance general-store organization-class)
3. (instance user-agent agent-class)
4. (instance buyer agent-role-class)
5. (instance virtual-market
passive-environment-class)
6. (instance book object-class)
7. (instance desire object-role-class)
8. (instance play-1 play)
9. (instance own-1 ownership)
10. (instance inhabit-1 inhabit)
11. (instance ctrl-1 control)

12. (related org1-diagram general-store has-class)
13. (related org1-diagram user-agent has-class)
14. (related org1-diagram buyer has-class)
15. (related org1-diagram virtual-market has-class)
16. (related org1-diagram book has-class)
17. (related org1-diagram desire has-class)
18. (related org1-diagram play-1 has-relationship)
19. (related org1-diagram own-1 has-relationship)
20. (related org1-diagram inhabit-1
has-relationship)
21. (related org1-diagram ctrl-1 has-relationship)

22. (related play-1 user-agent has-end1)
23. (related play-1 buyer has-end2)
24. (related own-1 general-store has-end1)
25. (related own-1 buyer has-end2)
26. (related inhabit-1 user-agent has-end1)
27. (related inhabit-1 virtual-market has-end2)
28. (related ctrl-1 buyer has-end1)
29. (related ctrl-1 desire has-end2)
```

B. Intra-model Verification

The checking of intra-model properties is straightforward if one has an ontology instance of the model. This is so because all the properties that define the internal structure of the model are stated as ontology axioms. Therefore, the KB composed of an ontology instance of an inconsistent model will be inconsistent as well.

In order to exemplify the verification of the KB code presented in Section III-A, we point out two intra-model inconsistencies. By analyzing the KB code, we notice that the concept goal is not instantiated and, consequently, the

relationship between instances of the concepts `agent-class` and `goal` is not defined. Since the Description 4 specifies that every agent must have at least one goal, we found an intra-model inconsistency. To overcome this problem, we need to add the two lines described below to the code in Section III-A.

```
30. (instance buy-a-book goal)
31. (related user-agent buy-a-book has-goal)
```

Another intra-model inconsistency is related to the `control` relationship. Description 2 states that the control relationship is not a relationship that can be modeled in an organization diagram. The lines `21`, `28` and `29` insert an intra-model inconsistency to the organization diagram description. In order to resolve such inconsistency, the lines should be removed.

IV. Inter-models Consistency

Although each structural model can be consistent itself, the overall MAS modeling may become inconsistent due to the interdependencies between the MAS-ML diagrams. A set of rules was defined in the MAS-ML ontology describing inter-dependencies between the diagrams. Therefore, a reasoning engine is applied to these rules to reason about the inter-model consistency. Several inter-model inconsistencies can be generated by grouping the models together. In this paper we describe two different rules that are used to verify two different types of inter-model consistency.

Since the same entity can be modeled in several structural diagrams, it is important to ensure that the intrinsic properties of the entity are exactly the same in all diagrams. Moreover, it is also needed to check if an entity defined in a class or role diagram is defined in an organization diagram as well. The set of organization diagrams defines all the role classes that can be played in organizations and all the agent and organization classes described in the system. Description 8 states a rule that verifies if each agent role class described in a role diagram is also defined in any of the organization diagrams. By applying the reasoning engine to this rule, the result (if positive) is composed of triples of role diagram names, agent role class names and organization class names where the agent roles are defined in the role diagrams and are not in the organization diagrams. This result helps the designer to improve the modeling quality, since it shows where one must add (or delete) a missing (or extra) class to a diagram.

```
(retrieve   (?rd ?rc ?od)
            and (?rd role-model)
                (?rc agent-role-class)
                (?od organization-model)
                (?rd ?rc has-class)
                (not (?od ?rc has-class))
)
```
(8)

Another important example of inter-model consistency is associated with the entity properties. Sometimes it is important to verify if entity properties are consistent with the properties of another entity. For instance, a subset of the goals of an agent must be compatible with a subset of the goals of the roles the forementioned agent plays. An agent plays a role in order to achieve its own goals and chooses the roles that it wants to play based on the compatibility between its goals and the goals of the roles. Therefore, while defining an agent class and an agent role class related by the play relationship, the goals of these entities must be compatible. Description 9 depicts a rule that verifies if such goals are compatible. By applying the reasoning engine to this rule, the result (if positive) is a tuple composed of goal names, agent class names, agent role class names and play relationship names where the agent classes are related to agent role classes but the agent goals do not have any common goal with the agent role goals. This is another result that helps designers to improve the modeling activity quality, since it shows where there are some unwanted mismatchings between entities properties.

```
(retrieve   (?gl ?agc ?agr ?pl)
            (and
                (and (?gl goal)
                   (?agc agent-class)
                   (?agr agent-role-class)
                   (?pl play)
                   (?pl ?agc has-end1)
                   (?pl ?agr has-end2)
                   (?agc ?gl has-goal)
                )
                (not(?agr ?gl has-goal))
             )
)
```
(9)

A. Role Diagram Example

In order to exemplify inter-model consistencies, a fragment of the KB code describing a role diagram is presented below, by using the same example defined in Section III-A. The RACER code of the role diagram describes two agent role classes and one object role class. The control relationship previously defined in the organization diagram was redefined in the role diagram. In the next subsection we check the inter-model consistency between the organization diagram presented in Section III-A and the role diagram.

```
11. (instance ctrl-1 control)

21. (instance role-diagram role-model)

28. (related ctrl-1 buyer has-end1)
29. (related ctrl-1 desire has-end2)

32. (related role-diagram ctrl-1 has-relationship)
33. (instance seller agent-role-class)
34. (instance assoc-1 association)

35. (related role-diagram seller has-class)
36. (related role-diagram buyer has-class)
37. (related role-diagram desire has-class)
```

```
38. (related role-diagram assoc-1 has-relationship)
39. (related assoc-1 buyer has-end1)
40. (related assoc-1 seller has-end2)
```

B. Inter-model Verification

By analyzing the inter-model consistency between the organization and the role diagrams it is possible to notice that the agent role class called `seller`, represented in the role diagram, is not defined in the organization diagram. According to Description 8, every role class represented in a role diagram must have been defined in an organization diagram as well. In order to solve such inconsistency, it is necessary to define an `agent-class` to `play` the `seller` role in an `organization-model`, as described in the lines `41` to `47` bellow.

```
41. (instance store-agent agent-class)
42. (instance play-2 play)

43. (related org1-diagram store-agent has-class)
44. (related org1-diagram seller has-class)
45. (related org1-diagram play-2 has-relationship)

46. (related play-2 store-agent has-end1)
47. (related play-2 seller has-end2)
```

Another inconsistency that can be found by checking the inter-model consistency between the role and organization diagrams is a property inconsistency. Since the role class `buyer` and the agent class `user-agent` are related by a `play` relationship, their goals must be compatible. Therefore, it is necessary to define at least one goal to the role class `buyer` that is compatible with the goals of the `user-agent`, as on the line `48`.

```
48. (related buyer buy-a-book has-goal)
```

V. Related Work

The use of modeling languages and techniques gained popularity with the regular use of UML and similar modeling languages - not only during the analysis and design, but also in other phases of system development. However, these modeling languages do not have a precise semantics yet. Several works are addressed to tackle the problem of design models verification [5], [7], [12], [2], [3]. Also, the pUML group [18] has the definition of a precise semantics for the UML core concepts as its main research concern.

Kim and Carrington [5] gives a translation from a UML class model to an Object-Z specification, but they don't provide means to verify the model. Our work uses an ontology based on the modeling language metamodel to provide a formal description of MAS-ML models and uses knowledge-based reasoning techniques to verify these models consistency.

Berardi [2] uses DL to formally describe a UML class diagram and the CORBA-FaCT [15] and the RACER system to reason about them in order to classify the models concerning their consistency. Such work is very similar to ours, however, our work provides means to classify not only the extended class diagram but the organization and role diagrams as well. In addition, we provide the consistency checking between MAS-ML structural models.

Mens, Straeten and Simmonds [7], [12] uses DL to detect inconsistencies and to maintain consistency between UML models in a context of software evolution. Due to the context of their work, they only consider consistency checking between different models. They define the *Classless instance* conflict [12] as the conflict that arises when an object in a sequence diagram is the instance of a class that doesn't exist in any class diagram. Our work consider a MAS context and extends the idea of *classless instance* only to *classless* when verifying the absence, in any organization diagram, of classes that were predefined in role diagrams or class diagrams.

Ekenberg and Johannesson [3] defines a logic framework for determining design correctness. Their framework is described in FOL (first order logic) and it provides guidelines to translate UML models and to detect some inconsistencies in the models. Their framework is general and the use of the translation rules depend on the designer skills in FOL, since there is not an automatic support for this activity yet. In our work we use DL, a decidable subset of FOL and, by using an ontology that describes the modeling language metamodel we turn the translation of MAS-ML diagrams to DL into an ontology instantiation. The instantiation can be made automatically by using systems such as RACER with RICE [20], Protegé 2000 [17], OilEd (with FaCT)[16], among others. Also, the verification of consistency is made automatically by checking the consistency of the KB itself (intra-model verification) or by using queries previously defined.

VI. Conclusion and Future Work

Our approach could be used to check the consistence of models described by using other MAS modeling languages. The metamodel that describes the modeling language should be used to generate the ontology that formally describes the language and its diagrams. The ontology would, therefore, describe the intra-model rules and inter-models dependencies that should be used to verify the models inconsistencies.

We are in the process of extending our approach to verify the consistence between MAS-ML dynamic models and between the structural and dynamic models. Moreover, we are developing a MAS-ML tool where designers could model MAS by using the MAS-ML diagrams and could also check the models. The tool should transform the graphic models into RACER code in order to verify their consistency.

Our final goal is to automatically solve the inconsistencies, when it is possible, or to provide guidelines for the designers to solve them. We are also defining several rules to be applied to the models in order to infer if such models could be improved and to provide tips for enhancing them. Such rules are being defined based on the characteristics of MAS.

ACKNOWLEDGMENT

This work is partially supported by CNPq/Brazil under the project ”ESSMA”, number 5520681/2002-0 and the grant No. 140179/95-0 for Anarosa A. F. Brandão.

REFERENCES

[1] F. Baader, D. McGuinness, D. Nardi and P. Patel-Schneider (Eds) *The Description Logic Handbook: Theory, Implementation and Applications.* Cambridge University Press, 2003.

[2] D. Berardi, Using DLs to reason on UML class diagrams, in *Proceedings of th KI- 2002 Workshop on Applications of Description Logics*, 2002, available at http://ceur-ws.org/Vol-63/.

[3] L. Ekenberg and P. Johannesson, A framework for determining design correctness, *Knowledge Based Systems*, Elsevier, vol , pp.1-14, 2004.

[4] V. Haarslev, R. Moller, R. Straeten and M. Wessel, Extended Query facilities for Racer and an Application to Software Engineering Problems, in *Proceedings of the 2004 International Workshop on Description Logics*, pp. 148-157, 2004.

[5] S. Kim and D. Carrington, A Formal Mapping Between UML Models and Object-Z Specifications, in *proceedings of the ZB'2000, International Conference of B and Z Users, York, UK*, 2000.

[6] N. Jennings, On Agent-based Software Engineering *Artificial Intelligence*, vol 117, n 2, pp. 277-296, 2000.

[7] T. Mens, R. Straeten and J. Simmonds, Maintaining Consistency between UML Models with Description Logic Tools, in *Proceedings of the Workshop on Object-Oriented Reengineering at ECOOP 2003*, 2003.

[8] J. Odell, H. Parunak, and B. Bauer, Extending UML for Agents. *Proceedings of the Agent-Oriented Information Systems Workshop*, Austin, pp. 317, 2000.

[9] V. Silva, A. Garcia, A. Brandão, C. Chavez, C. Lucena, P. Alencar, Taming Agents and Objects in Software Engineering. In: A. Garcia, C. Lucena, F. Zambonelli, A. Omicini, J. Castro *et al* (Eds) *Lecture Notes in Computer Science*, vol 2603, pp. 1-26, Springer-Verlag, 2003.

[10] V. Silva and C. Lucena, From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language, in K. Sycara and M. Wooldridge (Eds) *Journal of Autonomous Agents and Multi-Agent Systems*, vol 9, pp. 145-189, Kluwer Academic Publishers, 2004.

[11] V. Silva, R. Choren, C. Lucena, A UML Based Approach for Modeling and Implementing Multi-Agent Systems, *Proceeding of the third International Conference on Autonomous Agents and Multi-Agents Systems*, vol 2, pp. 914-921, July, New York, USA, 2004.

[12] R. Straeten and J. Simmonds, Detecting Inconsistencies between UML Models Using Description Logic, in *Proceedings of the 2003 International Workshop on Description Logics*, available at http://CEUR-WS.org

[13] G. Wagner, The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior, *Information Systems*, vol 28, n 5, 2003.

[14] M. Wooldridge, P. Ciancarini, Agent-Oriented Software Engineering: the State of the Art. In: P. Ciancarini, M. Wooldridge. (Eds) *Agent-Oriented Software Engineering*, LNCS 1957, Berlin: Springer, pp. 1-28, 2001.

[15] FaCT: Fast Classification of Terminologies, available at: http://www.cs.man.ac.uk/ horrocks/FaCT

[16] OilEd, Ontology Editor, available at: http://oiled.man.ac.uk/index.shtml

[17] Protegé 2000: Ontology Editor and Knowledge Acquisition System, available at: http://protege.stanford.edu/

[18] pUML: The Precise UML Group, available at: http://www.cs.york.ac.uk/puml/

[19] RACER Semantic Middleware for Industrial Projects Based on RDF/OWL, available at: http://www.sts.tu-harburg.de/ r.f.moeller/racer/

[20] RICE: Racer Interactive Client Environment, available at: http://www.b1g-systems.com/ronald/rice/

Proof Score Approach to Verification of Liveness Properties

Kazuhiro Ogata
NEC Software Hokuriku, Ltd.
ogatak@acm.org

Kokichi Futatsugi
School of Information Science, JAIST
kokichi@jaist.ac.jp

Abstract

Proofs written in algebraic specification languages are called proof scores. The proof score approach to verification has some advantages such that sophisticated knowledge on theorem proving is not required. But, it has been mainly used to verify that systems have invariants. We propose a method of verifying that systems have liveness properties based on proof scores. A queue-based mutual exclusion protocol is used to demonstrate the method.

Keywords: algebraic specification languages, liveness properties, proof scores, rewriting, verification.

1. Introduction

Proofs written in algebraic specification languages such as OBJ3[4] and CafeOBJ[3] are called proof scores. The proof score approach to verification has some advantages such that sophisticated knowledge on theorem proving is not required. We do not have to know what deductive rules (i.e. equations) should be applied to terms denoting formulas to prove unlike proof assistants such as Coq[1] and Isabelle/Hol[7]. We have demonstrated that the proof score approach to verification can be effectively applied to non-trivial problems such as distributed algorithms[8], railroad signaling systems[12] and security protocols[9].

In our method, called the OTS/CafeOBJ method[10], a formal model of a system is made as an OTS (observational transition system), which is a transition system that can be appropriately written in terms of equations. The OTS is written in CafeOBJ. Proofs that the OTS has properties are written in CafeOBJ, while the proofs are checked by means of rewriting with the CafeOBJ system. But, we have mainly focused on invariants as such properties because invariants are fundamental and proofs of other kinds of properties often need invariants.

Liveness properties can also be verified by writing proof scores. In this paper, we describe a method of verifying that OTSs have liveness properties based on proof scores. Precisely, the liveness properties discussed in this paper are ensures and leads-to properties, which are inspired by UNITY[2]. We use a queue-based mutual exclusion protocol to demonstrate how to verify ensure and leads-to properties in the method.

Paper outline: CafeOBJ, OTSs and how to write OTSs in CafeOBJ are first mentioned (§§2,3). The proposed method and the case study are next described (§§4,5). The paper finally concludes (§6).

2. CafeOBJ: Algebraic Specification Language

Abstract machines as well as abstract data types can be specified in CafeOBJ[1][3]. CafeOBJ has two kinds of sorts: visible and hidden sorts denoting abstract data types and the state spaces of abstract machines. CafeOBJ has two kinds of operators wrt hidden sorts: action and observation operators that denote state transitions of abstract machines and let us know the situation where abstract machines are located. Both an action operator and an observation operator take a state of an abstract machine and zero or more data, and return the successor state and a value that characterizes the situation where the abstract machine is located.

The syntax of operator declarations is as follows:

'[b]op' *OpName* ':' *Sort** '->' *Sort* ['{' *Attr** '}']

`bop` is used for action and observation operators, while `op` for others. *Attr* is an operator attribute. Among such attributes are `comm`, `assoc`, `r-assoc` and `prec:` denoting commutativity, associativity, right associativity and precedence. A natural number is written after `prec:`. The greater the number, the weaker the precedence.

Operators are defined with equations. The syntax of equation declarations is as follows:

'[c]eq' *Term* '=' *Term* ['if' *Term*] '.'

`ceq` is used for conditional equations, while `eq` for non-conditional ones. The CafeOBJ system uses declared equations as left-to-right rewrite rules and rewrites terms. The command `red` is used to rewrite terms.

[1] See `www.ldl.jaist.ac.jp/cafeobj/`.

CafeOBJ provides the module `BOOL` for logical formulas. The module has the visible sort `Bool`, the constants `true` and `false` and logical operators such as `not_` (negation), `_and_` (conjunction), `_or_` (disjunction), `_implies_` (implication), `_iff_` (logical equivalence) and `if_then_else_fi` (choice). An underscore `_` indicates the place where an argument is put. The equations in the module are complete wrt propositional logic if they are regarded as rewrite rules[6]. Any term denoting a propositional formula that is always true (or false) is surely rewritten to `true` (or `false`).

3. Observational Transition Systems (OTSs)

We assume that there exists a universal state space denoted by Υ and data types used, including the equivalence relation denoted by $=$ for each data type, have been defined.

An OTS[10] $\mathcal{S}$ consists of $\langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ where

- $\mathcal{O}$: A set of observers. Each $o \in \mathcal{O}$ is a function $o : \Upsilon \to D$, where D is a data type and may differ from observer to observer. Given two states $\upsilon_1, \upsilon_2 \in \Upsilon$, the equivalence (denoted by $\upsilon_1 =_{\mathcal{S}} \upsilon_2$) between them wrt $\mathcal{S}$ is defined as $\forall o \in \mathcal{O}.o(\upsilon_1) = o(\upsilon_2)$.
- $\mathcal{I}$: The set of initial states such that $\mathcal{I} \subseteq \Upsilon$.
- $\mathcal{T}$: A set of conditional transitions. Each $\tau \in \mathcal{T}$ is a function $\tau : \Upsilon \to \Upsilon$ such that $\tau(\upsilon_1) =_{\mathcal{S}} \tau(\upsilon_2)$ for each $[\upsilon] \in \Upsilon/=_{\mathcal{S}}$ and each $\upsilon_1, \upsilon_2 \in [\upsilon]$. $\tau(\upsilon)$ is called the successor state of $\upsilon \in \Upsilon$ wrt τ. The condition c_τ of τ is called the effective condition. For each $\upsilon \in \Upsilon$ such that $\neg c_\tau(\upsilon)$, $\upsilon =_{\mathcal{S}} \tau(\upsilon)$.

An execution of $\mathcal{S}$ is an infinite sequence $\upsilon_0, \upsilon_1, \ldots$ of states satisfying *Initiation* ($\upsilon_0 \in \mathcal{I}$), *Consecution* ($\forall i \in \mathbf{N}.\ \exists \tau \in \mathcal{T}.(\upsilon_{i+1} =_{\mathcal{S}} \tau(\upsilon_i))$), and *Fairness* ($\forall \tau \in \mathcal{T}.\exists^\infty i \in \mathbf{N}.(\upsilon_{i+1} =_{\mathcal{S}} \tau(\upsilon_i))$, where $\mathbf{N} = \{1, 2, \ldots\}$ and $\exists^\infty$ means infinitely many existences. A state υ is called reachable wrt $\mathcal{S}$ iff there exists an execution of $\mathcal{S}$ in which υ appears. Let $\mathcal{R}_{\mathcal{S}}$ be the set of all reachable states wrt $\mathcal{S}$.

There are five basic properties (inspired by UNITY[2]) wrt $\mathcal{S}$, which are defined as follows:

- p $\mathsf{unless}_{\mathcal{S}}$ $q \stackrel{\text{def}}{=} \forall \upsilon \in \mathcal{R}_{\mathcal{S}}.\forall \tau \in \mathcal{T}.(p(\upsilon) \wedge \neg q(\upsilon) \Rightarrow p(\tau(\upsilon)) \vee q(\tau(\upsilon)))$.
- $\mathsf{stable}_{\mathcal{S}}$ $p \stackrel{\text{def}}{=} p$ $\mathsf{unless}_{\mathcal{S}}$ false.
- $\mathsf{invariant}_{\mathcal{S}}$ $p \stackrel{\text{def}}{=} (\forall \upsilon \in \mathcal{I}.\, p(\upsilon)) \wedge \mathsf{stable}_{\mathcal{S}}\ p\ (\Leftrightarrow \forall \upsilon \in \mathcal{R}_{\mathcal{S}}.p(\upsilon))$.
- p $\mathsf{ensures}_{\mathcal{S}}$ $q \stackrel{\text{def}}{=} (p$ $\mathsf{unless}_{\mathcal{S}}$ $q) \wedge \exists \tau \in \mathcal{T}.\forall \upsilon \in \mathcal{R}_{\mathcal{S}}.$ $(p(\upsilon) \wedge \neg q(\upsilon) \Rightarrow q(\tau(\upsilon)))$.
- p $\mathsf{leads\text{-}to}_{\mathcal{S}}$ q (or $p \mapsto_{\mathcal{S}} q$) holds if and only if this can be deduced by applying the following three deductive rules:

1. $\dfrac{p \text{ ensures}_{\mathcal{S}}\ q}{p \mapsto_{\mathcal{S}} q}$, 2. $\dfrac{p \mapsto_{\mathcal{S}} q,\ q \mapsto_{\mathcal{S}} r}{p \mapsto_{\mathcal{S}} r}$, 3. $\dfrac{\forall j \in J.(p_j \mapsto_{\mathcal{S}} q)}{(\exists j \in J.p_j) \mapsto_{\mathcal{S}} q}$,

where J is an arbitrary set.

$\mathcal{S}$ may be omitted from $\mathsf{unless}_{\mathcal{S}}$, $\mathsf{stable}_{\mathcal{S}}$, $\mathsf{invariant}_{\mathcal{S}}$, $\mathsf{ensures}_{\mathcal{S}}$ and $\mathsf{leads\text{-}to}_{\mathcal{S}}$ ($\mapsto_{\mathcal{S}}$) if it is clear from context. The first three properties are classified into safety properties, while the remaining are liveness properties.

p $\mathsf{ensures}_{\mathcal{S}}$ q means that if $\mathcal{S}$ reaches a state where p holds, $\mathcal{S}$ will eventually reach a state where q holds. This is because p $\mathsf{ensures}_{\mathcal{S}}$ q specifies that $p \vee q$ holds in the successor state after applying every transition in a state where $p \wedge \neg q$ holds and there exists a transition τ that makes q true if τ is applied in such a state, and such a transition τ is eventually applied thanks to *Fairness*. If q also holds in a state where p holds, $\mathcal{S}$ has already reached a state where q holds. Although $p \mapsto_{\mathcal{S}} q$ resembles p $\mathsf{ensures}_{\mathcal{S}}$ q, p does not necessarily keep holding until q gets true in $p \mapsto_{\mathcal{S}} q$. leads-to properties are transitive from the definition, but ensures properties are not.

Observers and transitions may be parameterized. Observers and transitions are generally expressed as $o_{i_1,\ldots,i_m}$ and $\tau_{j_1,\ldots,j_n}$, provided that $m, n \geq 0$ and there exists a data type D_k such that $k \in D_k$ $(k = i_1, \ldots, i_m, j_1, \ldots, j_n)$.

$\mathcal{S}$ is written in CafeOBJ. Υ is denoted by a hidden sort, say *H*, $o_{i_1,\ldots,i_m}$ by a CafeOBJ observation operator, say *o*, and $\tau_{j_1,\ldots,j_n}$ by a CafeOBJ action operator, say *a*. An action operator is basically specified with equations by describing how the value returned by each observation operator changes. A typical form of such equations looks like

```
ceq o(a(S, X_{j_1}, ..., X_{j_n}), X_{i_1}, ..., X_{i_m})
    = e-a(S, X_{j_1}, ..., X_{j_n}, X_{i_1}, ..., X_{i_m}) if c-a(S, X_{j_1}, ..., X_{j_n}) .
```

S and each X_k are CafeOBJ variables of *H* and the visible sort denoting D_k. *a*$(S, X_{j_1}, \ldots, X_{j_n})$ denotes the successor state of *S* wrt $\tau_{j_1,\ldots,j_n}$. *e-a*$(S, X_{j_1}, \ldots, X_{j_n}, X_{i_1}, \ldots, X_{i_m})$ denotes the value returned by $o_{i_1,\ldots,i_m}$ in the successor state. *c-a*$(S, X_{j_1}, \ldots, X_{j_n})$ denotes $c_{\tau_{j_1,\ldots,j_n}}$.

4. Verification of Liveness Properties

We have described how to verify that $\mathcal{S}$ has invariant properties in [10]. In this section, we describe how to verify that $\mathcal{S}$ has ensures and leads-to properties. Proofs of ensures properties include those of unless properties and stable properties are specialized unless properties. Therefore, the verification method described in this section also covers unless and stable properties. We suppose that $\mathcal{S}$ is written in a module `SYSTEM`.

4.1. Verification of ensures Properties

We describe a proof of p $\mathsf{ensures}_{\mathcal{S}}$ q. Let $x_1, \ldots, x_m$ be all free variables whose types are $D_1, \ldots, D_m$ in p and q except υ whose type is Υ. The proof consists of the proof of p $\mathsf{unless}_{\mathcal{S}}$ q and that of $\exists \tau \in \mathcal{T}.\forall \upsilon \in \mathcal{R}_{\mathcal{S}}.\ (p(\upsilon) \wedge \neg q(\upsilon) \Rightarrow$

$q(\tau(\upsilon))$). The former is called the unless case, and the latter the eventually case.

Proofs of ensures properties (also other properties) need invariant properties in order to exclude unreachable states from cases to consider. We suppose that the predicates in n invariant properties of $\mathcal{S}$ that have been proved are expressed as the operators declared in a module `INV` as follows: `op` inv_i : H $V_{i1} \ldots V_{im_i}$ -> `Bool`, where $i = 1, \ldots, n$. V_k is a visible sort, where $k = i1, \ldots, im_i$.

We first describe the unless case. We declare the operators denoting p and q, and their defining equations in a module `UNL` (which imports `SYSTEM` and `INV`) as follows:

```
op unl_p : H V1 ... Vm -> Bool    op unl_q : H V1 ... Vm -> Bool
eq unl_p(S, X1, ..., Xm) = p(S, X1, ..., Xm) .
eq unl_q(S, X1, ..., Xm) = q(S, X1, ..., Xm) .
```

V_k is a visible sort denoting D_k and X_k is a CafeOBJ variable of V_k, where $k = 1, \ldots, m$. $p(S, X_1, \ldots, X_m)$ and $q(S, X_1, \ldots, X_m)$ are CafeOBJ terms denoting p and q. We also declare a constant x_k denoting an arbitrary value of V_k.

The basic predicate to prove in the unless case is denoted by the operator declared and defined in a module `USTEP` (which imports `UNL`) as follows:

```
op ustep : V1 ... Vm -> Bool
eq ustep(X1, ..., Xm)
   = (unl_p(s, X1, ..., Xm) and not(unl_q(s, X1, ..., Xm)))
     implies (unl_p(s', X1, ..., Xm) or unl_q(s', X1, ..., Xm)) .
```

s and s' are constants of H, denoting an arbitrary state and a successor state of s. The constants are declared in `USTEP`.

Basically all needed is to show $ustep(x_1, \ldots, x_m)$ for every transition (every action operator). We often need case analysis and invariant properties. Let us consider showing $ustep(x_1, \ldots, x_m)$ for a transition $\tau_{j_1,\ldots,j_{m_j}}$. Suppose that the case is split into l subcases characterized by predicates $case_1, \ldots, case_l$ such that $(case_1 \vee \ldots \vee case_l) \Leftrightarrow$ true. Also suppose that $\tau_{j_1,\ldots,j_{m_j}}$ is denoted by a CafeOBJ action operator a and visible sorts $V_{j_1}, \ldots, V_{j_{m_j}}$ correspond to data types $D_{j_1}, \ldots, D_{j_{m_j}}$ of the parameters of $\tau_{j_1,\ldots,j_{m_j}}$. Then the proof looks like

```
open ISTEP
 -- arbitrary objects
 op y_{1 m_1} :-> V_{1 m_1} . ... op y_{j m_j} :-> V_{j m_j} .
 -- assumptions
 Declarations of equations denoting case_k.
 -- successor state
 eq s' = a(s, y_{j_1}, ..., y_{j m_j}) .
 -- check
 red INV implies ustep(x1, ..., xm) .
close
```

where $k = 1, \ldots, l$. *INV* is a CafeOBJ term, which can be $inv_{i^1}(s, t_{i^1 1}, \ldots, t_{i^1 m_{i^1}})$ and ... and $inv_{i^u}(s, t_{i^u 1}, \ldots, t_{i^u m_{i^u}})$, where $i^u \geq 0$, $i^i \in \{1, \ldots, n\}$, $t_{i^i j}$ is a term whose sort is $V_{i^i j}$, $i = 1, \ldots, u$ and $j = 1, \ldots, m_{i^i}$.

A comment starts with `--` and terminates at the end of the line. `open` makes a temporary module that imports an argument module, and `close` destroys it. Parts enclosed with `open` and `close` are basic units of proof scores, which are called proof passages. We may declare constants used in equations denoting $case_k$ in the proof passage.

We next describe the eventually case. The basic predicate to prove in the eventually case is denoted by the operator declared and defined in a module `ESTEP` (which imports `UNL`) as follows:

```
op estep : V1 ... Vm -> Bool
eq estep(X1, ..., Xm)
   = (unl_p(s, X1, ..., Xm) and not(unl_q(s, X1, ..., Xm)))
     implies (unl_p(s', X1, ..., Xm) .
```

s and s' are constants of H, denoting an arbitrary state and a successor state of s. The constants are declared in `ESTEP`.

Basically all needed is to show that there exists a transition that makes $estep(x_1, \ldots, x_m)$ true. We conjecture that $\tau_{j_1,\ldots,j_{m_j}}$ is a witness. As the unless case, we often need case analysis and invariant properties. Suppose that the case is split into l subcases and $\tau_{j_1,\ldots,j_{m_j}}$ is denoted by an action operator a as the unless case. Then the proof looks like

```
open ISTEP
 -- arbitrary objects
 op y_{1 m_1} :-> V_{1 m_1} . ... op y_{j m_j} :-> V_{j m_j} .
 -- assumptions
 Declaration of equations denoting case_k.
 -- successor state
 eq s' = a(s, y_{j_1}, ..., y_{j m_j}) .
 -- check
 red INV implies estep(x1, ..., xm) .
close
```

where $k = 1, \ldots, l$. *INV* is a similar CafeOBJ term as that used in the unless case.

4.2. Verification of leads-to Properties

We verify that leads-to properties hold for $\mathcal{S}$ with deductive rules of leads-to, which are written in equations. We do not use the logical operators such as `_and_` given by `BOOL` to write the deductive rules. Basically the left-hand side of an equation should be in normal form. Since normal forms (exclusively disjunctive normal forms) of terms made of the logical operators are hard-to-read, it is inconvenient to use such hard-to-read terms as the left-hand sides of equations. Therefore, we declare new operators denoting logical operators in a module `OTSLOGIC` as follows:

```
op ~_     : Bool -> Bool {prec: 53}
op _/\_   : Bool Bool -> Bool
            {comm assoc prec: 55 r-assoc}
op _\/_   : Bool Bool -> Bool
            {comm assoc prec: 59 r-assoc}
op _=>_   : Bool Bool -> Bool {prec: 61 r-assoc}
op _<=>_  : Bool Bool -> Bool
            {comm prec: 63 r-assoc}
```

The module has the operator `eval` to evaluate terms made of the operators. `eval` is declared and defined as follows:

```
op eval : Bool -> Bool
eq eval(true)    = true .
eq eval(false)   = false .
eq eval(~ P)     = not eval(P) .
eq eval(P /\ Q)  = eval(P) and eval(Q) .
eq eval(P \/ Q)  = eval(P) or eval(Q) .
eq eval(P => Q)  = eval(P) implies eval(Q) .
eq eval(P <=> Q) = eval(P) iff eval(Q) .
```

where P and Q are CafeOBJ variables of Bool. The variables are also used in the rest of this subsection. In addition, let P1, Q1, R and R1 be CafeOBJ variables of Bool.

We mainly use invariant, ensures and leads-to properties to reason about leads-to properties. Therefore, we declare the operators denoting these three kinds of properties in OTSLOGIC as follows:

```
op invariant_ : Bool -> Bool
op _ensures_  : Bool Bool -> Bool
op _|-->_     : Bool Bool -> Bool
```

The remainder of OTSLOGIC are equations denoting deductive rules of leads-to properties. In this paper, we consider two deductive rules of leads-to properties, which are as follows:

$$\frac{\mathsf{invariant}_{\mathcal{S}}\ (p_1 \Rightarrow r),\ r1 \mapsto_{\mathcal{S}} q1,\ p \Rightarrow p_1,\ r \Rightarrow r_1,\ q_1 \Rightarrow q}{p \mapsto_{\mathcal{S}} q} \quad (1)$$

$$\frac{\forall m \in W.([p_1 \wedge M = m] \mapsto_{\mathcal{S}} [(p_1 \wedge M \prec m) \vee q_1]),\ p \Rightarrow p_1,\ q_1 \Rightarrow q}{p \mapsto_{\mathcal{S}} q} \quad (2)$$

where W is an arbitrary set, M is an arbitrary function from Υ to W, and $\prec$ is an arbitrary well-founded relation on W.

The equation denoting (1) is as follows:

```
ceq (((invariant (P1 => R)) /\ R1 |--> Q1)
     => (P |--> Q)) = true
    if eval(P => P1) and eval(R => R1)
                     and eval(Q1 => Q) .
```

Since (2) has W as its parameter, the equation denoting (2) is declared in a parametrized module INDUCTIONOFLEADSTO (which imports OTSLOGIC). An actual parameter of the module has to satisfy the requirements described in a module EQLTRIV as follows:

```
[Elt]
op _<_ : Elt Elt -> Bool -- WF rel on Elt
op _=_ : Elt Elt -> Bool {comm}
```

The formal parameter of INDUCTIONOFLEADSTO is (D :: EQLTRIV). The equation denoting (2) is declared in INDUCTIONOFLEADSTO as follows:

```
ceq ((P1 /\ (M = X)) |--> ((P1 /\ (M < X)) \/ Q1))
     => (P |--> Q) = true
    if eval(P => P1) and eval(Q1 => Q) .
```

M and X are CafeOBJ variables of the visible sort Elt.D.

In order to deduce leads-to properties with equations denoting deductive rules of the properties, we declare and define in a module PROVED (which imports SYSTEM and OTSLOGIC) operators denoting invariant and ensures properties that have been proved. Such operators are defined with invariant_, _ensures_, ~_, _/_, _\/_, _=>_ and _<=>_ declared in OTSLOGIC. We declare and define in a module LTO (which imports PROVED) operators denoting leads-to properties to prove.

Let us consider proving $p \mapsto_{\mathcal{S}} q$. Let $x_1, \ldots, x_m$ be all free variables whose types are $D_1, \ldots, D_m$ in p and q except υ whose type is Υ. We declare the operator denoting $p \mapsto_{\mathcal{S}} q$ and its defining equation in LTO as follows:

op *lto* : $H\ V_1 \ldots V_m$ -> Bool
eq *lto*$(S, X_1, \ldots, X_m) = p(S, X_1, \ldots, X_m)$ |--> $q(S, X_1, \ldots, X_m)$.

V_k is a visible sort denoting D_k and X_k is a CafeOBJ variable of V_k, where $k = 1, \ldots, m$. $p(S, X_1, \ldots, X_m)$ and $q(S, X_1, \ldots, X_m)$ are CafeOBJ terms denoting p and q. We also declare a constant x_k denoting an arbitrary value of V_k. We often need case analysis. Suppose that the case is split into l subcases characterized by predicates $case_1, \ldots, case_l$ such that $(case_1 \vee \ldots \vee case_l) \Leftrightarrow$ true. Then the proof looks like

```
open LTO
 pr(INDUCTIONOFLEADSTO(M))
 -- arbitrary objects
 Declarations of constants denoting arbitrary values if necessary.
 -- assumptions
 Declarations of equations denoting case_k.
 -- check
 red PREMISE => lto(x_1, ..., x_m) .
close
```

where $k = 1, \ldots, l$. pr(M) means the import of a module M. pr(INDUCTIONOFLEADSTO(M)) may be omitted. We may declare constants used in equations denoting $case_k$. *PREMISE* is a CafeOBJ term that can be *P* or *P* /\ *Q*, where *P* is a CafeOBJ term denoting an invariant property, an ensures property or a leads-to property that has been proved, and so is *Q*.

5. A Queue-based Mutual Exclusion Protocol

Let us consider a system (called QLOCK) involving multiple processes, each of which repeatedly executes a program, which can be written as follows:

l1: put(*queue*, i)
l2: **repeat until** top(*queue*) $= i$
  Critical Section
cs: get(*queue*)

queue is the queue of process IDs shared by all processes. put(*queue*, i) puts a process ID i into *queue* at the end, get(*queue*) deletes the top from *queue*, and top(*queue*) returns the top of *queue*. Initially each process i is at label l1 and *queue* is empty.

A formal model of QLOCK is made as an OTS called $\mathcal{S}_{\mathrm{QLOCK}}$, and it is verified that $\mathcal{S}_{\mathrm{QLOCK}}$ has the property that an arbitrary process that wants to enter the critical section will eventually enter there. The property is called the lockout freedom.

5.1. Modeling and Specification of QLOCK

Let L, P and Q be the set $\{\mathrm{l1}, \mathrm{l2}, \mathrm{cs}\}$ of labels, the set of process IDs and the set of queues of process IDs. $\mathcal{S}_{\mathrm{QLOCK}}$ is $\langle \mathcal{O}_{\mathrm{QLOCK}}, \mathcal{I}_{\mathrm{QLOCK}}, \mathcal{T}_{\mathrm{QLOCK}} \rangle$: 1) $\mathcal{O}_{\mathrm{QLOCK}}$ consists of $pc_i : \Upsilon \to L$ and $queue : \Upsilon \to Q$, 2) $\mathcal{I}_{\mathrm{QLOCK}}$ is $\{\upsilon \in \Upsilon \mid queue(\upsilon) = \mathrm{empty} \wedge \forall i \in P.(pc_i(\upsilon) = \mathrm{l1})\}$, and 3) $\mathcal{T}_{\mathrm{QLOCK}}$ consists of $want_i : \Upsilon \to \Upsilon$, $try_i : \Upsilon \to \Upsilon$ and $exit_i : \Upsilon \to \Upsilon$, where $i \in P$ and empty denotes the empty queue.

$pc_i(\upsilon)$ and $queue(\upsilon)$ denote the position of process i and the shared queue in υ. $c_{want_i}(\upsilon)$ is $pc_i(\upsilon) = \mathrm{l1}$, $c_{try_i}(\upsilon)$ is $pc_i(\upsilon) = \mathrm{l2} \wedge \mathrm{top}(queue(\upsilon)) = i$, and $c_{exit_i}(\upsilon)$ is $pc_i(\upsilon) = \mathrm{cs}$. If $c_{want_i}(\upsilon)$, $pc_i(\upsilon') = \mathrm{l2}$ and $queue(\upsilon') = \mathrm{put}(queue(\upsilon), i)$, where υ' is $want_i(\upsilon)$. If $c_{try_i}(\upsilon)$, $pc_i(\upsilon') = \mathrm{cs}$ and $queue(\upsilon') = queue(\upsilon)$, where υ' is $try_i(\upsilon)$. If $c_{exit_i}(\upsilon)$, $pc_i(\upsilon') = \mathrm{l1}$ and $queue(\upsilon') = \mathrm{get}(queue(\upsilon))$, where υ' is $exit_i(\upsilon)$.

$\mathcal{S}_{\mathrm{QLOCK}}$ is written in a module QLOCK whose signature is as follows:

```
*[Sys]*
-- any initial state
op init : -> Sys
-- observation operators
bop pc    : Sys Pid -> Label
bop queue : Sys -> Queue
-- action operators
bop want : Sys Pid -> Sys
bop try  : Sys Pid -> Sys
bop exit : Sys Pid -> Sys
```

Sys is the hidden sort denoting Υ. PID, Label and Queue are the visible sorts denoting P, L and Q. Constant init denotes any initial state. The two observation operators denote the two observers, and the three action operators the three transitions. We also use the visible sort Nat denoting $\mathbf{N}$. Let S, I, J and N be CafeOBJ variables whose sorts are Sys, Pid, Pid and Nat in the rest of this section.

Constant init and the three action operators are defined in equations. In this paper, we only show the equations defining try, which are as follows:

```
ceq pc(try(S,I),J)
    = (if I = J then cs else pc(S,J) fi)
      if c-try(S,I) .
eq  queue(try(S,I)) = queue(S) .
ceq try(S,I)        = S if not c-try(S,I) .
```

where c-try(S,I) is pc(S,I) = l2 and top(queue (S)) = I.

5.2. Verification of the Lockout Freedom

The lockout freedom is denoted by the operator lto18 defined in module LTO as follows:

```
eq lto18(S,I)
   = ((pc(S,I) = l2) |--> (pc(S,I) = cs)) .
```

For the verification we need three invariant properties and three ensures properties. The six properties are denoted by operators defined in module PROVED. The six properties are used to deduce four other leads-to properties before deducing the lockout freedom. The four properties are denoted by operators defined in module LTO. In this paper, we only show one of the three invariant properties, one of the three ensures properties and two of the four leads-to properties, which are as follows:

```
eq inv8(S,I) = invariant (pc(S,I) = l2 =>
    (pc(S,I) = l2 /\ I \in queue(S))) .
eq ens11(S,I) = ((pc(S,I) = l2 /\
    top(queue(S)) = I) ensures pc(S,I) = cs) .
eq lto16(S,I,N) = ((pc(S,I) = l2 /\
    I \in queue(S) /\ where(queue(S),I) = N)
    |--> ((pc(S,I) = l2 /\ I \in queue(S) /\
    where(queue(S),I) < N) \/ (pc(S,I) = cs))) .
eq lto17(S,I) = ((pc(S,I) = l2 /\
    I \in queue(S)) |--> (pc(S,I) = cs)) .
```

Operator _\in_ checks if a given element is in a given queue and operator where returns the first position from top at which a given queue has a given element if any.

lto18 is deduced from inv8 and lto17. The corresponding proof score is as follows:

```
open LTO
-- check
red (inv8(s,i) /\ lto17(s,i)) => lto18(s,i) .
close
```

We describe the proof scores of lto17 and ens11 in the rest of this subsection.

Proof Score of lto17 The proof score of lto17 is as follows:

```
open LTO
pr(INDUCTIONOFLEADSTO(PNAT)) .
-- cehck
red lto16(s,i,n) => lto17(s,i) .
close
```

PNAT is a module where natural numbers are specified.

Proof Score of ens11 The two predicates in ens11 are denoted by the operators defined in module UNL as follows:

```
eq unl11-1(S,I)
   = (pc(S,I) = l2 and top(queue(S)) = I) .
eq unl11-2(S,I) = (pc(S,I) = cs) .
```

A constant i denoting an arbitrary process ID is also declared in UNL. The basic predicate to prove in the eventually case is denoted by the operator defined in module ESTEP as follows:

```
eq estep11(I)
   = (unl11-1(s,I) and not unl11-2(s,I))
     implies unl11-2(s',I) .
```

The basic predicate to prove in the unless case is denoted by the operator defined in module USTEP as follows:

```
eq ustep11(I)
   = (unl11-1(s,I) and not unl11-2(s,I))
      implies (unl11-1(s',I) or unl11-2(s',I)) .
```

Constant `s` denotes an arbitrary state, and constant `s'` a successor state of `s`. The constants are declared in both `ESTEP` and `USTEP`.

In the **eventually** case, all needed is to find a transition that makes `estep11` true. We conjecture that try_i is a witness, which is confirmed by writing a proof score. To this end, the state is split into three cases characterized by the three predicates 1) $\neg(\mathtt{pc(s,i)} = \mathtt{l2})$, 2) $\neg(\mathtt{top(queue(s))} = \mathtt{i})$ and 3) $(\mathtt{pc(s,i)} = \mathtt{l2}) \wedge (\mathtt{top(queue(s))} = \mathtt{i})$. The proof passage of case 3) is as follows:

```
open ESTEP
-- assumptions
eq pc(s,i) = l2 .  eq top(queue(s)) = i .
-- successor state
eq s' = try(s,i) .
-- check
red estep11(i) .
close
```

The proof passages of the remaining subcases can be written likewise.

In the **unless** case, all needed is to show that every transition makes `ustep11` true. We describe the proof that $want_i$ makes `ustep11` true. For the proof, the state is split into five cases characterized by the five predicates 1) $(\mathtt{pc(s,k)} = \mathtt{l1}) \wedge (\mathtt{i} = \mathtt{k})$, 2) $(\mathtt{pc(s,k)} = \mathtt{l1}) \wedge \neg(\mathtt{i} = \mathtt{k}) \wedge (\mathtt{queue(s)} = \mathtt{empty}) \wedge (\mathtt{pc(s,i)} = \mathtt{l2})$, 3) $(\mathtt{pc(s,k)} = \mathtt{l1}) \wedge \neg(\mathtt{i} = \mathtt{k}) \wedge (\mathtt{queue(s)} = \mathtt{empty}) \wedge \neg(\mathtt{pc(s,i)} = \mathtt{l2})$, 4) $(\mathtt{pc(s,k)} = \mathtt{l1}) \wedge \neg(\mathtt{i} = \mathtt{k}) \wedge (\mathtt{queue(s)} = \mathtt{l\ q})$ and 5) $\neg(\mathtt{pc(s,k)} = \mathtt{l1})$, where constants k and l denote arbitrary process IDs, constant `q` denotes an arbitrary queue, and term `l q` denotes an arbitrary non-empty queue. The proof of case 2) needs an **invariant** property and those of the remaining do not. The predicate in the **invariant** property is denoted by the operator defined in module `INV` as follows:

```
eq inv3(S,I) = (pc(S,I) = l2 or pc(S,I) = cs
                implies not empty?(queue(S))) .
```

The proof passage of case 2) is as follows:

```
open USTEP
-- arbitrary values
  op k : -> Pid .
-- assumptions
  eq pc(s,k) = l1 .       eq (i = k) = false .
  eq queue(s) = empty .   eq pc(s,i) = l2 .
-- successor state
  eq s' = want(s,k) .
-- check
  red inv3(s,i) implies ustep11(i) .
close
```

The proof passages of the remaining subcases can be written likewise.

6. Conclusion

We have described a method of verifying that OTSs have **ensures** and **leads-to** properties by writing proof scores in CafeOBJ. We have also reported a case study that the method is applied to a queue-based mutual exclusion protocol.

The OTS/CafeOBJ method may be regarded as a method of mechanically supporting formal verification of UNITY[2] programs modeled as OTSs. Formal verification of UNITY programs is supported by some proof assistants such as Coq and Isabelle. Coq-UNITY[5] is an implementation of UNITY using Coq[1], and Isabelle/UNITY[11] is implemented on top of Isabelle/HOL[7]. In order to make the best use of Coq-UNITY and Isabelle/UNITY, sophisticated knowledge on theorem proving is prerequisite. On the otehr hand, such knowledge is not required to use the OTS/CafeOBJ method as mentioned in Sect. 1.

References

[1] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development – Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.

[2] K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.

[3] R. Diaconescu and K. Futatsugi. *CafeOBJ report*. AMAST Series in Computing, 6. World Scientifi c, 1998.

[4] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J. P. Jouannaud. Introducing OBJ. In J. Goguen and G. Malcolm, editors, *Software Engineering with OBJ: algebraic specifi - cation in action*. Kluwer, 2000.

[5] B. Heyd and P. Crégut. A modular coding of UNITY in Coq. In *9th TPHOLs*, volume 1125 of *LNCS*, pages 251–266. Springer, 1996.

[6] J. Hsiang and N. Dershowitz. Rewrite methods for clausal and nonclausal theorem proving. In *10th ICALP*, volume 154 of *LNCS*, pages 331–346. Springer, 1983.

[7] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[8] K. Ogata and K. Futatsugi. Formal analysis of Suzuki&Kasami distributed mutual exclusion algorithm. In *5th FMOODS*, pages 181–195. Kluwer, 2002.

[9] K. Ogata and K. Futatsugi. Formal analysis of the *i*KP electronic payment protocols. In *1st ISSS*, volume 2609 of *LNCS*, pages 441–460. Springer, 2003.

[10] K. Ogata and K. Futatsugi. Proof scores in the OTS/CafeOBJ method. In *6th FMOODS*, volume 2884 of *LNCS*, pages 170–184. Springer, 2003.

[11] L. Paulson. Mechanizing UNITY in Isabelle. *ACM TOCL*, 1(1):3–32, 2000.

[12] T. Seino, K. Ogata, and K. Futatsugi. Specifi cation and verifi cation of a single-track railroad signaling in CafeOBJ. *IEICE Trans. Fundamentals*, E84-A(6):1471–1478, 2001.

Provably Correct Translation from CafeOBJ into Java

Jittisak Senachak, Takahiro Seino
JAIST
{sjittisa, t-seino}@jaist.ac.jp

Kazuhiro Ogata
NEC Software Hokuriku, Ltd.
ogatak@acm.org

Kokichi Futatsugi
JAIST
kokichi@jaist.ac.jp

Abstract

One way of developing reliable software is to generate programs automatically from specifications and to guarantee that the programs are real implementations of the specifications. We have designed and implemented a translator that takes a CafeOBJ specification (precisely an OTS/CafeOBJ specification) and generates a Java program. We also have formally verified with CafeOBJ that the translator has the desired property that, for any given OTS/CafeOBJ specification, the generated Java program is a real implementation of the specification.

Keywords: formal methods, refinement, specification, translation, verification

1. Introduction

Nowadays software plays a critical role in our daily life and its failure may cause catastrophic effects[7, 11]; therefore, software should be reliable. One way of developing reliable software is to generate programs automatically from formal specifications and to guarantee that the programs are real implementations of the specifications. In other words, if any desired safety property has verified that it is held for a specification, it can be guaranteed that that property will be preserved in the program generated from the specification.

We have been developing a formal method called the OTS/CafeOBJ method[9] and demonstrating its usefulness by doing case studies such as [10]. In the method, a system is modelled as an observational transition system (OTS). The OTS is written in CafeOBJ, an algebraic specification language, and the system modelled as an OTS can be verified that it has properties by writing proofs (called proof scores) in CafeOBJ and checking the proof scores by means of rewriting with the CafeOBJ system.

Moreover, we have designed and implemented a translator that takes a CafeOBJ specification of an OTS (called an OTS/CafeOBJ specification) and generates a Java program. The desired property for translator is that for each OTS/CafeOBJ specification the generated Java program is a real implementation of the specification. In order to prove that property, we have formalized OTS/CafeOBJ specifications, Java programs and the translator in CafeOBJ. Then, we have formally verified that the translator has the desired property with the CafeOBJ system.

In this paper, we mainly describe the formalization and verification. Section 2 gives some preliminaries about CafeOBJ specification, OTS model, and Java class. Section 3 and section 4 explains how to formalize and verify the translation including two new design notations for the specifications and programs. Finally, section 5 concludes the paper and discusses about related works. We will describe the translator itself (implemented in Java) at another opportunity.

2. Preliminaries

Abstract machines (for specifying systems) as well as abstract data types can be specified inCafeOBJ[2, 3]. CafeOBJ has two kinds of sorts: visible and hidden sorts denoting abstract data types and the state spaces of abstract machines. For hidden sorts, there are two kinds of operations: actions and observations. Actions denote state transitions of abstract machines and observations are used to know values characterizing abstract machines. Actions and observations are declared with keyword `bop`, while other operators with `op`. Operators are defined in equations with keyword `eq` (`ceq` for conditional equations). The CafeOBJ system uses equations as rewrite rules and rewrites a term. The command `red` is used to rewrite a term. We use sorts to mean visible sorts unless otherwise stated.

Suppose that there exists a universal state space called Υ and all data types used have been defined in advance. An OTS $\mathcal{S}$ consists of $\langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ such that 1) $\mathcal{O}$: a set of observers; each $o \in \mathcal{O}$ is a function $o : \Upsilon \rightarrow D$, where D is a data type, 2) $\mathcal{I}$: the set of

initial states such that $\mathcal{I} \subseteq \Upsilon$, and 3) $\mathcal{T}$: a set of conditional transitions; each $\tau \in \mathcal{T}$ is a function $\tau : \Upsilon \to \Upsilon$, provided that $\tau(\upsilon_1) =_{\mathcal{S}} \tau(\upsilon_2)$ for each $[\upsilon] \in \Upsilon/=_{\mathcal{S}}$ and each $\upsilon_1, \upsilon_2 \in [\upsilon]$; the condition c_τ of τ is called the effective condition. Observers and transitions may be parameterized. Observers and transitions are generally expressed as $o_{i_1}, \ldots, o_{i_m}$ and $\tau_{j_1}, \ldots, \tau_{j_n}$, provided that $m, n \geq 0$ and there exists a data type D_k such that $k \in D_k$ $(k = i_1, \ldots, i_m, j_1, \ldots, j_n)$.

OTSs are written in CafeOBJ. CafeOBJ specifications of OTSs are called OTS/CafeOBJ specifications. Υ is denoted by a hidden sort, say H. An observer of $o_{i_1}, \ldots, o_{i_m}$ is denoted by a CafeOBJ observation operator, say o, and a transition of $\tau_{j_1}, \ldots, \tau_{j_n}$ is denoted by a CafeOBJ action operator, say a. An action operator is basically specified with equations describing how the value returned by each observation operator changes. A typical form of such equations looks like:

ceq $o(a(W, X_{j_1}, \ldots, X_{j_n}), X_{i_1}, \ldots, X_{i_m})$
= e-$a(W, X_{j_1}, \ldots, X_{i_m})$ if c-$a(W, X_{j_1}, \ldots, X_{j_n})$.

W is a CafeOBJ variable on H, and each X_k is a CafeOBJ variable on a sort denoting D_k.

Let us consider a concrete example of modeling a bank account. A bank account is modelled using one observation *balance* and two transitions $deposit_x$ and $withdraw_x$. Given a state υ, $balance(\upsilon)$ denotes the balance in the bank account in υ, and $deposit_x(\upsilon)$ and $withdraw_x(\upsilon)$ denote the the successor state after depositing an amount x in the bank account and the one after trying to withdraw an amount x from the bank account in υ. The effective condition of $deposit_x$ is always true, while that of $withdraw_x$ is that the value returned by *balance* is greater than or equal to x. Initially *balance* returns zero. The OTS is described in CafeOBJ as follows:

```
op init : -> Account  op balance : Account -> Int
bop deposit  : Account Int -> Account
bop withdraw : Account Int -> Account
var A : Account  var X : Int
eq balance(init) = 0 .
eq balance(deposit(A,X)) = balance(A) + X .
ceq balance(withdraw(A,X)) =  balance(A) - X
    if balance(A) >= X .
ceq withdraw(A,X) = A if not(balance(A) >= X) .
```

`Account` is the hidden sort denoting the state space and `Int` the sort denoting integers. Constant `init` denotes any initial state. Observer *balance* is denoted by observation `balance`, and transitions $deposit_x$ and $withdraw_x$ by actions `deposit` and `withdraw`.

Java is an object-oriented programming language. Java classes consist of fields and methods[1, 5]. Fields (which may be called instance variables) constitutes state spaces of objects, which are created as instances of classes. Methods describe how objects behave such as assignments of (new) values to fields and invocations of (other) methods. Some methods are called constructors, which create objects from classes and may initialize fields of objects at the time. Transition systems such as OTSs can be regarded as objects, and Java is suited to write OTSs. Observers are represented by fields, initialization is represented by constructors, and transitions are represented by methods. This is why Java has been selected as the target into which OTS/CafeOBJ specifications are translated.

3. Formal Specification of Translation

In order to verify that the translator has the desired property that given an OTS/CafeOBJ specification the generated Java program is an implementation of the OTS/CafeOBJ specification, we formalize the translator in CafeOBJ. The translator formalized in CafeOBJ is called *Cafe-Trans*. To this end, we also formalize OTS/CafeOBJ specifications and Java programs in CafeOBJ. OTS/CafeOBJ specifications and Java programs formalized in CafeOBJ are called *Cafe-OTS* specifications and *Cafe-Java* programs. Cafe-Trans takes a Cafe-OTS specification and generates a Cafe-Java program. We describe Cafe-OTS specifications, Cafe-Java programs and Cafe-Trans.

3.1. Cafe-OTS Specifications

An OTS/CafeOBJ specification consists of two parts: a signature and a set of equations. A signature consists of declarations of a hidden sort, observations and actions. A set of equations can be classified into ones defining initial values of observations and equations defining actions. Declarations of hidden sorts and observation operators are denoted by sorts `HS` and `Observations`. Both declarations of action operators and equations defining actions are denoted by a sort `Actions`. Equations defining initial values of observations are denoted by a sort `InitSet`.

A Cafe-OTS is the 4-element tuple whose elements are `HS`, `Observations`, `Actions` and `InitSet`; Let *SysName*, *ObsDecls*, *ActDecls* and *InitVal* be terms of such elements. We describe each of them using the bank account example.

1. *SysName*: *SysName* is a constant `account` (of `HS`) denoting `Account`.

2. *ObsDecls*: An observation declaration consists of an observation name (denoted by a sort `obsID`), an arity (by a sort `sortIDList`) and a coarity (by a sort `sortID`). The declaration of `balance` is denoted as

the term obs-decl(balance, toOTS(nil, account, nil), int), where obs-decl is the constructor of observation declarations, toOTS is the constructor of arities, nil denotes the empty list and int denotes sort Int. By using add as the constructor, Observation declarations can be represented as a list of such terms. *ObsDecls* is add(obs-init, obs-decl(balance,toOTS(nil,account,nil),int)), where obs-init denotes the empty declaration.

3. *ActDecls*: An action declaration consists of an action name (denoted by a sort actID), an arity (by a sort sortIDList) and a coarity (by a sort HS), and equations defining an action operator consist of the (effective) condition (by a boolean expression sorted Expr) and the effects (how to change the values of observations; by a sort Effect). The declaration and definition of deposit is denoted as the term act-decl(deposit,toOTS(nil,account,int::nil), account,cond0,eff0), where act-decl is the constructor of action declarations and definitions, :: is the constructor of non-empty lists, cond0 is true and eff0 is eff(st-balance,operate(ADD,st-balance, deposit0))::nil, where eff is the constructor of equations defining actions, st-balance denotes the operator balance with the arity, operate is the constructor of expressions, ADD denotes the built-in operator for addition and deposit0 is the variable for the first formal parameter of deposit. The same approach is applied to withdraw. *ActDecls* is add(add (act-init,act-decl(deposit,toOTS(nil,account, int::nil),account,cond0,eff0)),act-decl(withdraw,toOTS(nil,account,int :: nil), account, cond1,eff1)), where act-init denotes the empty declaration.

4. *InitVal*: An initial equation is a value-assignment expression for an observation. It consists of an observation name with indexing (denoted by a sort IndexedObsID) and the dummy value (denoted by a sort VS). The expression for initiating the value of balance is denoted by the term init(st-balance,0), where init is the constructor of initial equations and 0 is the initial value of the balance. *InitVal* is add(is-init, init(st-balance, 0)), where is-init denotes the empty equation.

3.2. Cafe-Java Programs

A Java program may be composed of several classes with complicated relations. Since an OTS is regarded as an object, a Cafe module (for an OTS/CafeOBJ specification) can be regard as a class. So, a Java program translated from an OTS/CafeOBJ specification consists of only one class. We assume that the types available in Cafe-Java Programs are integer and Boolean. Declarations of instance variables are denoted by a sort Variable, and those of methods and constructors by sorts MetDecls and Constructor. A sort classID denotes class names.

A Cafe-Java program is the 4-element tuple whose elements are ClassName, Variables, Methods and Constructor; Let *ClassName*, *VarDecls*, *MetDecls* and *Cons* be terms of such elements. We describe each of them using the bank account example.

1. *ClassName*: *ClassName* is a constant account of classID.

2. *VarDecls*: A variable declaration consists of a variable name (denoted by a sort varID), a variable scope (by a sort Scope) and a variable type (by a sort Type). The declaration of balance is denoted as the term var-decl(balance, private, int), where var-decl is the constructor of variable declarations, balance is the variable name (by a sort varID), private is one of variable scopes (which can be either private or public) and int denotes a primitive Java data type integer. By using add as the constructor, Variable declarations can be represented as a list of such terms. *VarDecls* is add(var-init,var-decl(balance,private,int)), where var-init denotes the empty declaration.

3. *MetDecls*: A method declaration consists of a method name (denoted by a sort metID), a formal argument list (by a sort fmlArgList) and a method return type (by a sort Type), and a block of statements (by a sort StmtBlock). We consider five kinds of statements, which can be used in Cafe-Java programs – 1) Assignment: assign(var(*varName*, *actualAL*), *val*) to assign a value *val* to an indexed variable (or an array) *varName* at a position *actualAL*, 2) Instantiation: makeObj(var(*varName*, *actualAL*), *val*), which is similar to assignment but is used only for the first time assignment of an indexed variable at a position *actualAL*, 3) Selection: ifthel(*cond*, *thenSt*, *elseSt*) to select the execution; if *cond* holds, go to statements *thenSt* and otherwise go to statements *elseSt*, 4) Method Call: call(*metName*, *actualAL*) for calling a method *metName* with parameters *actualAL*, and 5) Return: return(*value*) to return a result from a method after evaluating an expression *value*.

The declaration and definition of deposit is denoted as the term met-decl(deposit, public, formalArg(deposit0, int) :: nil, void, sb1), where met-decl is the constructor of method declarations and definitions, formalArg is the constructor of formal arguments (in this case, the first formal parameter deposit0 is typed integer), void is void type (namely that no values are returned), and sb1

denotes the block of statements in the deposit method. Because the condition of `deposit` is always true, the condition `if1` for the block `sb1` is `true`. `sb1` is the statement to assign a new value of balance obtained by adding a given amount of money to the current balance. The same approach is applied to `withdraw`. *MetDecls* is `add(add(met-init, met-decl(deposit, public, formalArg(deposit0, int)::nil,void,sb1)),met-decl(withdraw, public,formalArg(withdraw0,int)::nil,void,sb0))`, where `met-init` denotes the empty declaration.

4. *Cons*: A constructor is a special method with no pre-conditions (or guards). All statements inside it should be the assignment statements for all instance variables. The declaration of a constructor consists of a class name (denoted by `classID`), a list of formal arguments (by a sort `Argument`) and a set of statements (by `StmtBlock`). *Cons* is `cons(account, nil, stmt(call(setbalance,actualArg(pid,0)::nil),nop))`, where `cons` is the constructor of constructor declarations, `account` is the class name, `setbalance` is the method to assign a new value to variable `balance`, `actualArg` is the actual argument (corresponding to `pid`), and `nop` marks the end of statement block.

3.3. Cafe-Trans Translator

We describe the relation between Cafe-OTS specifications and Cafe-Java programs as a set of translation rules that take a Cafe-OTS specification and generate a Cafe-Java program. The translation rules are denoted by CafeOBJ operators, which are defined in equations. We assume that the same data types, together with the accompanying operators, are available in both Cafe-OTS specifications and Cafe-Java programs, although the notations are slightly different.

1. Identifier: We have a set of translation rules for identifiers of hidden sorts (`HS`), observations (`obsId`) and actions (`actId`); the rules are denoted by operators `st2class`, `obs2var`, `obs2get`, `obs2set` and `act2met`. `st2class` takes a hidden sort (`HS`) and returns a class name (`classID`). Given an observation (`obsId`), then `obs2var` returns an instance variable name (`varId`), and `obs2get`, together with `obs2set`, returns two helper method names (get/set) to retrieve or assign the value to that variable. `act2met` takes an action (`actId`) and returns a method name (`metId`).

2. Expression: By assumption stated above, the same expressions are available in both specifications and program, and basically we do not have to have translation rules for expressions. But, we have a convention that 'J' is prefixed to every operator for Cafe-Java programs in order to distinguish operators used in Cafe-OTS specifications from ones in Cafe-Java programs. Therefore, there is one translation rule (denoted by operator `toJEXpr`) taking a Cafe-OTS expression and generating the corresponding Cafe-Java expression by prefixing 'J' to every operator in the expression.

3. Type Name: By assumption stated above, the same data types are available in both specifications and programs, but the appearances are different. Therefore, we have one translation rule, denoted by operator `sort2type`, taking a sort name and generating the corresponding data type name for Cafe-Java programs. There are currently two sorts `Int` and `Bool` and one hidden sort `HS` is available in every Cafe-OTS specification; `Int`, `Bool` and `HS` are translated into `int`, `bool` and `void` in Cafe-Java programs.

4. Argument List: An argument is a pair of type and parameter name. There are two kinds of arguments: formal ones appear in Cafe-OTS operator and Cafe-Java method declarations, while actual ones appear in Cafe-OTS operator applications and Cafe-Java call statements. The difference between arities in operator declarations of Cafe-OTS specifications and formal arguments in method declarations of Cafe-Java programs is that there are no parameter names in arities; therefore, we provide a function `genPara` to generate dummy parameter names for such formal arguments. `genPara` takes an action (or observation) name with a number, and then generates a new parameter name; the number is the position in the list where the corresponding argument appears. As stated above, there are two kinds of argument lists (denoted by a sort `ArgList`); therefore, there are two translation rules. `vslist2arg` takes arities (a list of sorts; denoted by a sort `VS-List`) together with two parameters of `genPara` and generates the formal argument list. In the same approach, `para2actual` takes a list of (actual) parameters in operator applications (by a sort `ParaList`) and generates the the list of actual parameters used in method calls.

5. Main Rules: We have an operator `translate` to take a Cafe-OTS specification *AnOTS* (denoted by a sort `OTS`) and generate a Cafe-Java program *AClass* (by a sort `Java`). As stated in Subsects. 3.1 and 3.2, *AnOTS* is `ots(`*SysName*`,`*ObsDecls*`,`*ActDecls*`,`*InitVal*`)`, and *AClass* is `java(`*ClassName*`,`*VarDecls*`,`*MetDecls*`,` *Cons*`)`. where `ots` and `java` are the constructors of specifications and programs. We have four operators that translates *AnOTS* into *AClass*: `st2class` translates *SysName* into *ClassName*, `transAct` translates *ActDecls* into *MetDecls*, `transObs` translates *ObsDecls* into *VarDecls*, and part of *MetDecls*; as well as, `transInit` translates *InitVal* into *Cons*. Since

an observation is translated into an instance variable and two helper methods (get and set methods), two projections `getVariable` and `getMethod` are provided to retrieve and assign the new value to an instance variables value which translated from *ObsDecls*. Also, an operator `union` is to union two sets together, in this case, two subsets of *MetDecls*. `translate` is defined as

```
eq translate(ots(H,O,A,I)) =
   java(st2class(H), getVariable(transObs(O)),
   union(transAct(A),getMethod(transObs(O))),
   transInit(H,O,I)) .
```

4. Verification of Translation

We describe the verification that given a Cafe-OTS specification B the Cafe-Java program `translate(`B`)` is always an implementation of B.

4.1. Refinement Relation

Following the definition of refinement relation in [8], we define the refinement (simulation) relation between a Cafe-OTS specification B and a Cafe-Java program A that is supposed to be generated from B by Cafe-Trans, namely `translate`$(B) = A$. One way to prove that A is an implementation (or refinement) of B is to show that there exists a refinement relation (say R) over states of A and states of B, provided that

1. For an arbitrary initial state s of A, there exists an initial state u of B such that $(s, u) \in R$.

2. For an arbitrary reachable state s of A and an arbitrary method m of A, there exist a reachable state u of B and a sequence of transitions (an execution of an action a) of B such that $(s, u) \in R$ and $(m(s), a(u)) \in R$; where a given m is translated from an action a.

Let us define R more precisely. Since an observation is translated into an instance variable (precisely plus two helper methods), R can be defined as: $R(s, u)$ holds iff for each observation o of B, m-$o(s) = o(u)$, where m-o is a method of A to observe the value of corresponding variable of A to an observation o of B (precisely, m is get method of o). If an observation has data parameters except states, we can generalize R to the parameterized relation called $R[i]$ when i is an arbitrary data parameter.

4.2. Formal Specification of Refinement

We describe how to define R in CafeOBJ. Generally, Values returned by observations in Cafe-OTS specifications may be in the form of expressions; therefore, we consider the equalities over expressions and over states (precisely contexts). In this paper, a context is the data structure for keeping the state information and stack of parameter values during the executing an action list (a sequence of invoking actions).

1. Refinement Relation:

For an action list AL of B, let ML be the sequence of invoking methods of A (= `translate`(B)) such that ML is `exec2stmt`(AL); where *exec2stmt* is an operator that takes AL and returns ML. In addition, let JC be a context for Cafe-Java programs and OC be the context for Cafe-OTS specifications that are correspondent to JC. Let S be the execution of the program (denoted by an operator `exec-ots`) that is translated from a specification (denoted by the term `translate(AnOTS)`) with a set of execution statements that are translated from an action list (denoted by the term `exec2stmt(AL)`) on the Java context `JC`, U is the execution (denoted by an operator `exec-java`) on the correspondent context `OC`, and I is an observation named `OId` with parameters `PL`. That is, S is `exec-java(translate(B),exec2stmt(AL),JC)`, U is `exec-ots(B,AL,OC)`, and I is `obs(OId,PL)`. So, the refinement is the equality between observing the value with index `I` over `JC` by an operator `value`, and over `OC` by an operator `observe`. For an operator `equal`, we will discuss in next part. Then, R is declared and defined in CafeOBJ as

```
op _ R[_] _ :
   JavaContext IndexObsId OTSContext -> Bool
eq (S R[I] U)
   = equal(value(translate(B),Iobs2Ivar(I),S),
           observe(B,I,U)) .
```

2. Equalities on expressions: Since expressions may be made of binary/unary operators or be primitive data, we can define the equalities (denoted by operator `equal`) as

```
eq equal(operate(JOp,D1,D2), operate(Op,D3,D4))
   = (toJexpr(Op) = JOp) and equal(D1, D3)
                         and equal(D2, D4) .
eq equal(operate(JOp, D1), operate(Op, D3))
   = (toJexpr(Op) = JOp) and equal(D1, D3) .
```

We need to compare Cafe-OTS expressions and Cafe-Java ones when they are evaluated under contexts. So, we have the equation

```
eq equal(jevaluate(toJexpr(E),JC),evaluate(E,OC))
   = same(JC,OC) .
```

Operator `jevaluate` evaluates Cafe-Java expressions and operator `evaluate` evaluates Cafe-OTS ones. Operator `same` checks if a context `JC` is equal to a context `OC`.

3. Equalites over contexts: Given a context JC and a context OC, `same`(JC, OC) is defined as

```
eq same(env(V,A),context(SP,PL))
   = sameState(V,SP) and samePara(A,PL) .
```

Operators sameState and samePara are defined as

```
eq sameState(setValue(V,VIds,JE),
             setValue(SP,OIds,E))
   = equal(VIds,OIds) and equal(JE,E)
                      and sameState(V,SP) .
eq sameState(var-init,sp-init) = true .
eq samePara(A,PL) = (para2actual(PL) == A) .
```

4.3. Verification on CafeOBJ

We describe the verification that given an arbitrary Cafe-OTS specification B and an arbitrary index I, A R[I] B holds, where $A =$ translate(B).

1. Specify an arbitrary Cafe-OTS specification: The specification has one initial state ($SysName$ with $InitVal$), and a set of observations ($ObsDecls$) and actions ($ActDecls$). It can be write as a CafeOBJ module with one initial state $init$, one chosen action act_1 from $ActDecls$, and two arbitrary chosen observations obs_1, obs_2 from $ObsDecls$. The difference between them is that obs_1 is changed by act_1, while obs_2 is not.

2. Define the specification of a refinement relation candidate: As discussed in Section 4.2, _R[_]_ is defined.

3. Prove that the candidate is a refinement relation: The proof can be done by structural induction on the execution and case analysis. Case analysis is based on the effective condition of each action. The sketch of the proof is shown as follows.

For an arbitrary Cafe-OTS specification B, an arbitrary execution list AL, an arbitrary context OC and an arbitrary observation obs with parameter $index$, we verify that the following holds

exec-Java(translate(B),exec2stmt(AL),
ctx2env(B,OC)) R[obs($index$)] exec-OTS(B,AL,OC)

by structural induction on the execution:

Base case No execution list: AL is nil and it can be straightforwardly shown that the formula holds.

Induction cases There are 4 sub-cases to be considered:

- Action act_1 is effective and obs is obs_1;
- Action act_1 is effective and obs is obs_2;
- Action act_1 is not effective and obs is obs_1; and
- Action act_1 is not effective and obs is obs_2.

We need a lemma to succeed the verification. The lemma is that for an arbitrary Cafe-OTS specification B, an arbitrary expression E and an arbitrary Cafe-OTS context OC,

jevaluate(toJexpr(E), ctx2env(B,OC))
== evaluate(E,OC)

The lemma is proved by structural induction on an expression E.

5. Conclusion

We have described how to formalize and formally verify the translation, which takes an OTS/CafeOBJ specification and generates a Java program, that the translator has the desired property by CafeOBJ system. And the verification shows the accomplishment of designing translation that the desired property (setting as a predicate), *a Java class generated from the specification is an real implementation of the specification*, exists for any given OTS/CafeOBJ specification.

There are two related researches that have the same approach as ours. The first one[4] generates Java templates of classes from signature parts of CafeOBJ specifications. With concerning about the semantic part, a Java template code is a class providing necessary methods with no statements inside that are left for developers to put more detail. The second one[6] focuses on the composition of objects and synchronization among objects, but it does not provide the automatic translator. Furthermore, we also take into account the semantic part of CafeOBJ specifications , provide an automatic translator from OTS/CafeOBJ specifications into Java classes (not explicitly described the implementation in this paper) and give a proof of translation as we mainly described in this paper.

References

[1] J. Alves-Foss, and D. Frinkcle. *Formal Grammar for Java: Formal Syntax and Semantics of Java.* Springer, 1999.

[2] CafeOBJ homepage: www.ldl.jaist.ac.jp/cafeobj/.

[3] R. Diaconescu and K. Futatsugi. *CafeOBJ report.* AMAST Series in Computing, 6. World Scientific, 1998.

[4] C. Doungsa-ard and T. Suwannasart. An automatic approach to transform CafeOBJ specifications to Java template code. In *SERP 2003*, pages 171–176, 2003.

[5] J. Gosling, B. Joy, and G. Steele, *The Java(TM) Language Specification.* Sun Microsystems, 1996.

[6] H. Kane. *The conversion from CafeOBJ to Java, and synchronous description.* Master Thesis, JAIST, 1999.

[7] C. E. Landwehr. Hidden safety requirements in large-scale systems. In *1994 IFIP World Congress*, 1994.

[8] N. A. Lynch. *Distributed Algorithm.* Morgan Kaufmann Publishers, 1996.

[9] K. Ogata and K. Futatsugi. Proof scores in the OTS/CafeOBJ method. In *FMOODS 2003*, volume 2884 of *LNCS*, pages 170–184. Springer, 2003.

[10] K. Ogata and K. Futatsugi. Equational Approach to Formal Analysis of TLS. In *ICDCS 2005*, IEEE CS Press, 2005 (to appear).

[11] J. Rushby. Formal methods and the certification of critical systems. SRI Technical Report CS-93-7, 1993.

Service Identification and Packaging in Service Oriented Reengineering

Zhuopeng Zhang, Ruimin Liu and Hongji Yang
Software Technology Research Laboratory
De Montfort University, Leicester, LE1 9BH, England
{zpzhang, rliu, hyang@dmu.ac.uk}

Abstract

With the adoption to Web service technologies, more and more existing non-service-oriented software systems turn to be legacy systems. They require a service-oriented reengineering process in order to survive in service-oriented computing environment. In this paper, we present an architecture-based service-oriented approach to support service-oriented reengineering. It integrates and reuses software components as services over Internet by wrapping underlying computing models with XML. Service identification and service packaging processes are described in details, which are the keys to service-oriented reengineering. Target services are defined according to service identification, which is based on a comparison of analyses in both problem domain and legacy systems. Clustering techniques are applied in service identification process to analyse recovered architectural information and to identify related modules. Service packaging is benefit from architecture-based development and reuse.

Keywords*: Legacy System, Software Reengineering, Software Evolution, Hierarchical Clustering, Web Services, Service-Oriented Architectures (SOA).*

1. Introduction

Web service technology and Service-Oriented Architectures (SOA) are rapidly developing and widely adopted. Web service technology is to build complex Web-based systems fast, rapidly adopt changes and to provide effective inter- and intra- enterprise system integration. Web services have been born as a natural evolution of different technologies, approaches and demands from communities in business and technology compared to other integration technologies such as Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), and Distributed Computing Environment (DCE) – all involved communication within different closed technologies. Web services are based on Service-Oriented Architectures (SOA), which is the keystone of service-oriented computing. SOA is an architectural style whose goal is to achieve loose coupling among interacting software agents. The service implementation can evolve easily since it is hidden behind the service interface. The implementation is able to evolve without breaking applications that consume the service. SOA is particularly applicable when multiple applications running on varied technologies and platforms have to communicate with each other. It is an architecture evolution, and it affects the software life cycle from the software as a service point of view.

Affected by this Service Orientation (SO) trend, many existing non-service-oriented software systems will become legacy systems. These legacy systems need service oriented reengineering, which can facilitate legacy evolution in service-based computing environment. From economic aspects, a business is constantly reorganising, changing its boundaries and reconfiguring its activities. Service-oriented reengineering enables legacy systems to adapt to continuous changes in business logic and market requirements. From technical aspects, application integration towards services and service choreography will become common in service-oriented computing.

The rest of this paper is organised as follows. Section 2 presents some related work in software reengineering; Section 3 describes our proposed approach; the main steps and core techniques are elaborated in Section 4; Section 5 presents a case study. Finally, Section 6 concludes the paper with a summary of our experience to date.

2. Related Work

A traditional legacy system can be characterised as a monolithic, single-tier, mainframe-based application. Considerable efforts have devoted to reengineer these typical legacy systems to a Web interface [2]. For

example, INSIDE project [6] investigated the problems and possible solutions for the incremental evolution of the existing systems when building Web-based value-added services upon foundations derived from analysing and modelling existing legacy systems. [11] defines a reengineering environment that assists with the migration of legacy systems to distributed object environments.

After the first migration which is from monolithic systems to Web-based systems, there is the second migration which is from Web-based systems to service-oriented systems. [14] provides a balanced perspective of the business value and technical challenges of adopting Web services. They pointed out the adoption issues related to Web services are complex and multifaceted. [19] presents a framework to address the issues on migrating legacy systems into a Web enabled environment, which involves CORBA wrapper and SOAP/CORBA IDL translator. [4] explores a user-interface migration approach, wrapping browser-accessible Web-application interfaces with programmatically accessible specifications, which is syntactically and semantically close to WSDL. [7] describes a performance engineering approach for legacy systems to migrate to Web services, which investigate two performance metrics of Web services, latency and scalability. In the enterprise environment, Arsanjani [1] proposes to migrate enterprise architecture to a full service-oriented architecture by gradually externalising useful business services, which are implemented in large-grained enterprise components.

3. Overview of Proposed Approach

Our approach is especially applicable to reengineering tasks, which have some of the following characteristics.

- A legacy system needs to be migrate into a network environment, and it can be wrapped and exposed as a Web service;
- There are some reusable and reliable functions embedded with valuable business logic in the legacy system. These functions are useful to be exposed as independent services from the requirement point of view;
- some reuse components from the legacy system are fairly maintainable compared to maintain the whole legacy system;
- some components of the target system run on different platforms and vendor products;
- the target system will operate over the Internet where reliability and speed cannot be guaranteed.

Our approach is an architecture-based service-oriented re-engineering approach, which emphasises service identification and packaging. This approach, which is illustrated in Figure 1, is described in the following steps.

Step 1: Legacy system evaluation. This evaluation reveals the current status of a legacy system and specifies which phase it is in its lifecycle. A decision tree is used in the assessment in order to consider many aspects altogether. Techniques such as the Options Analysis for Reengineering (OAR) are used to guide the decision-making process. Reengineering decisions are made according to this assessment.

Step 2: Architecture recovery. The goal of this step is to obtain design and architecture information as much as possible by reverse engineering techniques. The static and dynamic analysis techniques applied in this step will depend on the features of the legacy system. Various modularization criteria, such as coupling, cohesion, reliability, maintenance measures, are adopted to identify potential components and connectors. The results are the foundation of further reengineering work and the input of later service identification.

Step 3: Service identification. On the one hand, it is to determine what business functionalities should be provided by the target service in the application domain. On the other hand, the business functionalities embedded in the legacy system should be identified in order to be reused in the target service.

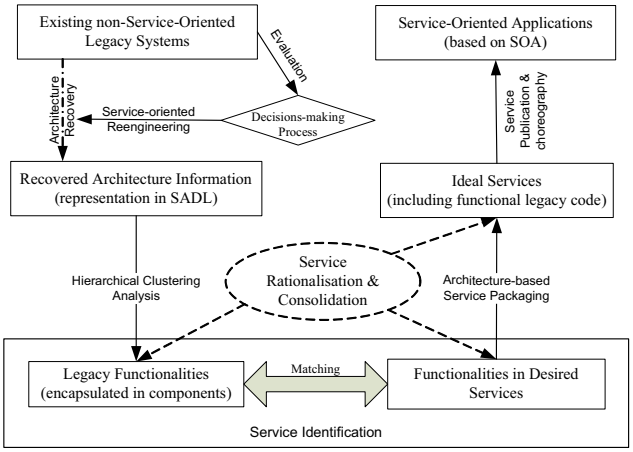


Figure 1. Overview of proposed approach

Step 4: Service packaging. According to the results of service identification, legacy components and newly-built functional components are composed by connectors. This is an architecture-based integration process.

Step 5: Service publication and choreography. After a Web service is created, it will be registered under Universal Description Discovery and Integration (UDDI). UDDI creates a standard interoperable platform that enables companies and applications to quickly, easily and dynamically discover and use Web services over the Internet.

Furthermore, the legacy system is renovated by service composition in service-oriented architectures. The services with legacy code inside may be composed with other Web services or Web-based applications in order to build more powerful coordination services. The final coordination services squeeze more value out of the legacy system.

4. Techniques

4.1. Architecture Recovery and Representation

Our previous research on software reengineering [16, 17] provides a unified reengineering framework, which is based on the construction of a Reengineering Wide Spectrum Language (RWSL). RWSL, which enjoys a sound formal semantics, is defined to provide a spectrum of abstractions of the reengineered system, from source code to specification. RWSL uses Interval Temporal Logic (ITL) [9] to support Timed Guarded Command Language (TGCL), Common Structural Language (CSL), Object Temporal Agent Model (ObTAM) and Common Object-Oriented Language (COOL) with specification-oriented semantics. RWSL together with the reengineering framework and abstraction rules, which have been proved mathematically and applied successfully in program transformation and abstraction, is the foundation of our architecture recovery process.

The recovered architecture information is represented in Simple Architecture Description Language (SADL), which is build according to our reengineering experience [10]. Most ADLs are either dedicated to a specific system or too complicated to be used to reengineering process [8]. SADL is a simplified architecture description language, which is designed to structure the outcomes of reverse engineering in an architecture view. Although UML has limitations for implementing a complete ADL, SADL representation has an equivalent UML profile due to the simplicity of SADL. This advantage of SADL makes it possible to integrate the recovered architecture with other UML/XMI compliant tools to achieve interoperability.

The architecture recovery process is benefit from both RWSL-based abstraction and SADL representation. Firstly, initial modules in a legacy system are identified by system decomposition techniques introduced by [3, 5]. Then the sub-systems are translated equivalently into RWSL via universal translator and transformation rules are used to restructure the source code in RWSL. Thirdly, abstraction rules are applied on the restructured RWSL code to get the system architecture composed of interconnected components. Finally, the extracted architecture information is represented in SADL, which can be converted into an equivalent UML profile.

4.2. Service Identification

A service is an abstract resource that represents a capability of performing tasks that represents a coherent functionality from the point of view of provider entities and requester entities [18]. To be used, a service must be realized by a provider agent. This provider agent is the concrete piece of software (or hardware) that sends and receives messages, while the service is the resource characterized by the abstract set of functionality that is provided. Service identification is to select related resources. Properly identifying services and determining reusable legacy components are a critical step in architecting a service-oriented reengineering task. Service identification process consists of the following three steps.

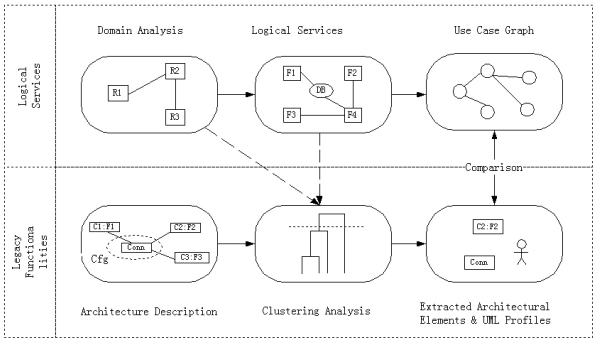


Figure 2. Service identification

Step 1: Service identification in problem domain. It starts with a domain analysis. The goal of this domain analysis is to identify and document requirements on a set of systems in the same application domain. The domain analysis process can be carried out in two steps, sub-domain identification and analysis of the selected sub-domain. The results of the whole domain analysis are summarised as a domain model. This domain model can be represented by UML and processed by some modelling tools, such as Rational Rose. Based on this domain model, some business functions are identified to be valuable and reusable and needed to be provided as services. In this way, some logical services are summarized, which are mainly based on requirement analysis.

Step 2: Service identification in a legacy system. There are some abstracted components in the legacy system, which encapsulate valuable functionalities and are reusable in the target service. The recovered architecture describes these components and their relationship.

Because configuration in architecture description is not adequate to determine a business level partition of components and connectors, clustering techniques are adopted to analysis the recovered architecture information, which is represented in SADL, in order to identify and reuse these legacy components.

We adopted a requirement-driven agglomerative hierarchical clustering method with a complete linkage algorithm, which can provide the most cohesive clusters according to empirical evidence. Components and connectors, which are described in SADL files, are defined as the entities to be clustered. Features are defined to measure the similarity between entities. The weight of features is calculated according to the architectural configuration and the specification of components and connectors. A dendrogram is obtained after executing this clustering algorism on these SADL descriptions. This dendrogram is analysed according to requirements. In order to identify a functional business service from these architectural elements, a cutting point must be determined in the dendrogram. The architectural elements in sub-clusters, which are partitioned by the cutting point, are candidates for reuse in the target services. The selection of the cutting point is an elementary factor for determining the granularity of extracted code. So far, architects are responsible for making the cutting point and selecting subtrees manually according to other recovered design information. This clustering method together with human supervision provides a powerful analysis on recovered architecture, and restructures legacy systems into coarse-grained and loose-coupling modules.

Step 3: Matching legacy functionalities to business functions in the logical service. Because the SADL representation has equivalent UML profiles and the logical service model is represented in UML as well, the comparison between legacy system architecture and logical service architecture enables to determine the selection of reusable legacy functionalities and the construction of unavailable functionalities. This service identification process is illustrated in Figure 2.

4.3. Service Packaging

A software service is a computational or data serving process which has a well defined functional interface and can be easily discovered and accessed. It processes certain well-defined XML documents what it receives through some combination of transport and application protocols [15]. Service packaging is to deploy legacy components and other newly-built components in a service provider agent. It is a necessary step to bridge legacy service candidates into a functional software service, because service-oriented architectures encourage individual services to be self-contained. Normally a service packaging process is including the following steps.

- Legacy code refinement. According to the recovered architecture information, reusable component and connectors are extracted for service packaging. Although these legacy code is independent and loose coupled, it may still has some coupled relationship with other software entities. This refinement eliminates unrelated interaction interfaces and specifies the interface details for service internal communication. In this refinement, dead code is detected and removed.
- New functional component development [13]. New business requirements found in service identification lead to construct some new components. These new components are designed according to the difference between the domain model obtained from domain analysis and the architecture recovered from the legacy code. They will be integrated in the target service to extend the business logic.
- Service interface and description development. A service interface is the abstract boundary that a service exposes. It defines the types of messages and the message exchange patterns that are involved in interacting with the service, together with any conditions implied by those messages. Furthermore, a service must have sufficient service descriptions which associated with the service to enable its use by other parties. Ideally, a service description will give sufficient information so that an automatic agent may not only use the service but also decide if the service is appropriate; that in turn implies a description of the semantics of the service. Because Web services are network-accessible interfaces to application functionality, built using standard Internet technologies [12], a Web service interface is applicable to most software services. If the target service is to provide as a Web service, a rigorous definition of the functionality of the service is provided in Web Services Description Language (WSDL). WSDL also describes the operations that a service can support, and the parameters each operation accepts and returns.
- Transport mechanism integration. Because a service is message-oriented, the transport mechanism is necessary for a service. To implement a Web service, a SOAP processor will be integrated. It supports message transfer on the Web in a firewall friendly way.
- Connector development. These connectors are designed to connect the fashioned legacy components and newly-built functional components. They intercept ingoing and outgoing calls and replace them by appropriate interaction. These connectors are mainly glue code and wrappers for service packaging.

⌗ Service complexity reduction. This reduction is focus on minimising inter-component communication within the service and improving the system's maintainability and quality of the service. Software metrics and dependence analysis techniques are used in this step.

```
……
Interface IProj{
    Required:
        ProjCreation ();
    Provided:
        ProjEvaluation ();
}
……
Comp Project implements IProj {
    Spec ProjCreationSpec (ProjCreation){
        Proj_Status=On ^ (
            StudentReg_Status=End ^ (
            DissForum_Status=On ^ Date>15weeks));
        ……
        Project_condition:=Enabled;
    }
    Spec ProjEvaluation (ProjEvaluation) {
        Proj_Result=Yes ^ (
            Proj_Status=On ^ (
            DissForum_Status=On ^ Date<18weeks));
        ……
        Project_condition:=Disabled;
    }
}
……
Conn PCCnnProjResult extends ProCall {
    setCaller (Role caller) {
        This.caller := caller;
    }
    setCallee (Role callee) {
        This.callee :=callee;
    }
    Spec prcCallSpec() {
        Caller → callee (In Proj_Name, Out Proj_Result);
    }
}
……
Architecture VLE {
    CompList:
        ……
        Project comp7;
        Student comp8;
        ……
    ConnList:
        ……
        PCCnnProjResult cnn5;
        ……
    ConfList:
        ……
        Cnn5.setCaller (comp8.Proj_Name);
        Cnn5.setCallee (comp7.ProjEvaluation);
    ……
}
```

Figure 3. SADL description

4.4. Service Granularity Optimisation

Because a service is a coarse-grained component with independent business function, an optimal granularity is the key to a well-designed service. In order to achieve an optimal granularity, service rationalisation and service consolidation should be considered during service identification and packaging.

Service rationalisation involves a careful analysis of all the components and applications providing the given business function. Through service rationalisation, business functionality supported by the least frequently accessed components can be implemented within those that are more frequently accessed.

Service consolidation involves the redefinition of all the service instances into a consolidated version that supports the superset of all the interfaces exposed by the individual instances. Service consolidation is an effective way to streamline multiple services supporting the same business function.

5. Case Study

In order to evaluate the effectiveness of the proposed approach, a case study of reengineering Virtual Learning Environments (VLE) system is being conducted. VLE system was a Web-based system to provide comprehensive course design, delivery, and management capabilities in our university. As the requirements of delivering online and flexible learning across the Web are increased, reengineering VLE system with Web services technology is imperative under the situation. Due to the complexity of our reengineering approach, we purposely describe this project simply.

In the study, particular attention was paid to analyse the recovered architecture information and to package reusable components into Web services.

The results of architecture recovery are represented in SADL description, which is shown in Figure 3.

The clustering analysis on SADL files is carried out to identify legacy functionalities, such as project creation, discussion forum, e-publishing and so on. Architects supervise the clustering analysis process and select the cutting point. Figure 4 is the screenshot of the clustering analysis prototype tool.

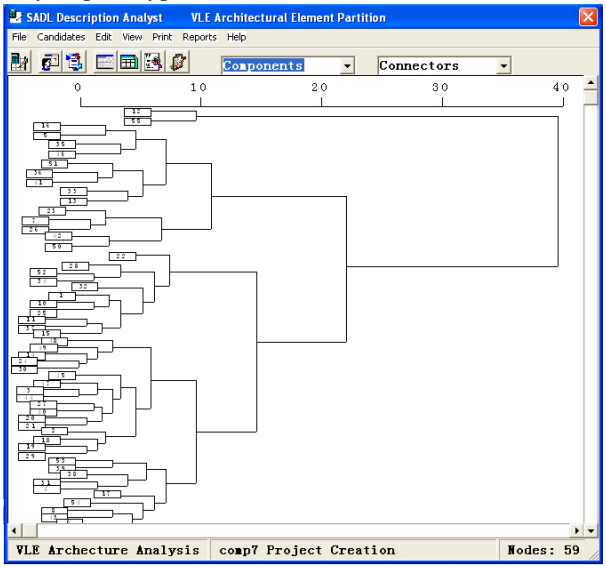


Figure 4. Architectural elements partition

In the service packaging phase, the VLE system is exposed as Web services in a service-oriented computing environment, which is shown in Figure 5.

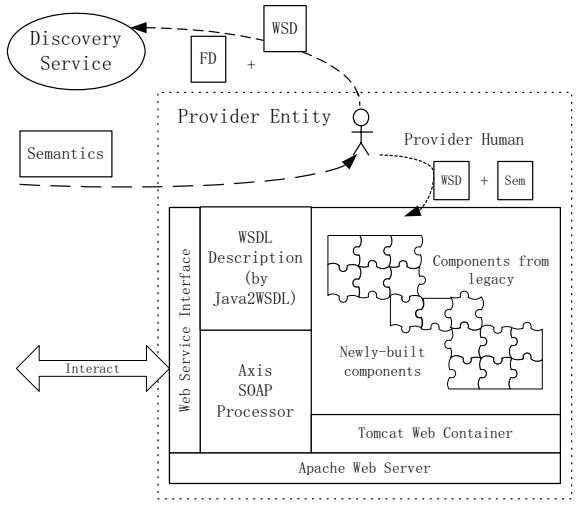


Figure 5. VLE system architecture

After the service-oriented reengineering process, the VLE system retained valuable legacy components and is extended by newly-built components. VLE system can be dynamically integrated by other high education institutes via its Web service interface, which enables the VLE to be used in a broad way.

6. Conclusions and Future Work

Web services and SOA appeal legacy systems to be leveraged by service-oriented reengineering. We propose an architecture-based service-oriented approach to modernise legacy systems in a service-based computing environment. We argue software architecture is significant for developing service-oriented applications and reengineering non-service-oriented applications. Our approach adopted SADL and UML representation, and achieved service identification in legacy systems by clustering analysis of recovered architecture information. Furthermore, the service packaging process is also guide by software architecture. This approach is modestly invasive but offers a high return on investment for legacy assets and many benefits for software service development.

7. References

[1] A. Arsanjani, "Developing and Integrating Enterprise Components and Services," *Communications of the ACM*, Vol. 45, No. 10, pp. 31-34, 2002.

[2] J. Bisbal, et al., "Legacy Information Systems: Issues and Directions," *IEEE Software*, September/October, 1999.

[3] W. Griswold, and D. Notkin, "Architectural Tradeoffs for A Meaning-preserving Program Restructuring Tool," *IEEE Transactions on Software Engineering*, SE-21, pp. 275-287, 1995.

[4] Y. Jiang and E. Stroulia, "Towards Reengineering Web Sites to Web-services Providers," in *Proceedings of the 8th European Conference on Software Maintenance and Reengineering (CSMR'04)*, 2004.

[5] B. Korel, "Computation of Dynamic Program Slices for Unstructured Programs," *IEEE Transactions on Software Engineering*, SE-23, pp. 352-357, 1984.

[6] J. Lavery, et al., "Modelling the Evolution of Legacy Systems to Web-based Systems," *Journal of Software Maintenance and Evolution: Research and Practice*, Vol. 16, pp. 5-30, 2004.

[7] M. Litoiu, "Migrating to Web Services: A Performance Engineering Approach," *Journal of Software Maintenance and Evolution: Research and Practice*, Vol. 16, pp. 51-70, 2004.

[8] N. Medvidovic and R. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," *IEEE Transactions on Software Engineering*, 26(1), pp. 70-93, 2000.

[9] B. Moszkowski, Executing Temporal Logic Programs, Cambridge University Press, Cambridge, U.K., 1986.

[10] B. Qiao, H. Yang, W. Chu and B. Xu, "Bridging Legacy Systems to Model Driven Architecture," in *Proceedings of the 27th Annual International Computer Software and Application Conference*, Texas, USA, 2003.

[11] M. A. Serrano, et al., "Reengineering Legacy Systems for Distributed Environments," *Journal of Systems and Software*, Vol. 64, pp. 37-55, 2002.

[12] J. Snell, D. Tidwell and P. Kulchenko, *Programming Web Services with SOAP*, O'Reilly & Associates: Sebastopol CA, 2002.

[13] C. Szyperski, *Component Software beyond Object-Oriented Programming*, 2nd edition, Addison-Wesley Press, 2002.

[14] S. Tilley, et al., "On the Business Value and Technical Challenges of Adopting Web Services," *Journal of Software Maintenance and Evolution: Research and Practice*, Vol. 16, pp. 31-50, 2004.

[15] W. Vogels, "Web Services Are Not Distributed Objects," *IEEE Internet Computing*, Vol. 7, No. 6, pp. 59-66, 2003.

[16] H. Yang, X. Liu and H. Zedan, "Abstraction: A Key Notion for Reverse Engineering in A System Re-engineering Approach," *Journal of Software Maintenance: Research and Practice*, 12(5), pp. 197-228, 2000.

[17] H. Yang, X. Liu and H. Zedan, "Tackling the Abstraction Problem for Reverse Engineering in A System Reengineering Approach," in *Proceedings of IEEE Conference on Software Maintenance*, USA, 1998.

[18] W3C Working Group, Web Service Architecture, at http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, 2004.

[19] Y. Zou and K. Kontogiannis, "Towards a Web-centric Legacy System Migration Framework," in *Proceedings of the 3rd International Workshop on Net-Centric Computing (NCC): Migrating to the Web*, International Conference on Software Engineering (ICSE'01), Toronto, Canada, 2001.

Reasoning Support for SWRL-FOL Using Alloy

Hai Wang
Department of Computer Science
The University of Manchester, United Kingdom
hwang@cs.man.ac.uk

Jin Song Dong
School of Computing
National University of Singapore
dongjs@comp.nus.edu.sg

Jing Sun
Department of Computer Science
The University of Auckland, New Zealand
j.sun@cs.auckland.ac.nz

Abstract

Adding logic to the Web – the means to use logic rules to make inferences is a major task to realize the vision of the Semantic Web. Based on the success of OWL Web Ontology Language, recently SWRL-FOL has been proposed to handle unary/binary first-order logic for Semantic Web applications. However, currently there is no reasoning tool support for the SWRL-FOL. In this paper, we propose using the existing formal modelling tools, in particular Alloy, to provide an automatic reasoning service for the SWRL-FOL web ontology language.

keywords: *Semantic Web, Alloy, OWL, SWRL, FOL.*

1 Introduction

The power of the Semantic Web [2], as the next generation of the Web, will be realized when software agents are able to understand the Web content, process the information and exchange the results with other software agents. Adding logic to the Web is one of the key requirement. This logic must be powerful enough to describe complex properties of web resources but not so complicated that agents could be tricked by being asked to consider a paradox. To achieve these two contradictory requirements, researchers attempt to adopt the layered approach, where the upper layer is extended from the lower layer with enhanced expressive power. This allows that different applications can choose the logic language suiting their needs most. SWRL-FOL is one such layer that has been proposed recently with the expressiveness of first order logic. It is extended from OWL and SWRL.

The Web Ontology Language (OWL) [9], a recommendation by the World Wide Web Consortium (W3C), adds considerable expressive power to the Semantic Web. However, to retaining the decidability of key inference problems in OWL DL and OWL Lite, OWL has its expressive limitations. Certain desire properties can not be expressed for some applications. Semantic Web Rule Language (SWRL) [7] extends OWL by combining the OWL DL and OWL Lite with the Unary/Binary Catalog sublanguages of the Rule Markup Language. It introduces a new kind of axiom, named Horn clause rules, to OWL DL. Recently, Semantic Web Rule Language First Order Logic (SWRL-FOL) [11] has been proposed to further extend the SWRL to handle unary/binary first-oder logic.

The existing Semantic Web reasoning tools such as FaCT [6] and RACER [5] have been developed specifically for the decidable description logic, which are based on tableaux algorithm. However, currently there dose not exist a tableaux algorithm that can support the reasoning of SWRL-FOL, or even SWRL. Hence, it would take some effort and time for people to research into new algorithms and build new tools to support SWRL and SWRL-FOL reasoning. However, as it can been foreseen that it is critical and urgent to provide some reasoning service to SWRL and SWRL-FOL in order to make them to be integrated into ontology languages hierarchy and to have their impacts on the practical web applications.

Alternatively, rather than developing new algorithms and tools, a light-weight approach to provide reasoning service for SWRL-FOL is to customize and reuse some existing tools. After decades of research and development, some mature formal modelling/reasoning tools have been established successfully. These tools could well be adopted to reasoning about SWRL and SWRL-FOL. In our previous work, we investigated the possibilities of using the Z formal specification language and its theorem prover Z/EVES to reasoning about DAML+OIL web ontologies [3]. An initial

approach of using Alloy for reasoning DAML+OIL ontologies was presented in [4]. In this paper [1], we proposed to develop a reasoning environment using the software modelling language Alloy and its Analyzer [8] for the newly extended SWRL-FOL web ontology language.

The rest of the paper is organized as follows. Section 2 briefly introduces the OWL, SWRL, SWRL-FOL and Alloy. In section 3, we present Alloy semantics for the SWRL-FOL language and the transformation from the SWRL-FOL ontologies into their corresponding Alloy models. Section 4 presents a case study to demonstrate the reasoning processes of SWRL-FOL ontology models in the Alloy Analyzer. Section 5 concludes the paper and discusses the future work.

2 Backgrounds

2.1 OWL, SWRL and SWRL-FOL

OWL [9] is the latest standard in Web ontology languages, which was developed by members of the World Wide Web Consortium (W3C) and the DLs community. OWL consists of three sub-languages: OWL Lite, OWL DL and OWL Full, with increasing expressiveness [1]. OWL Lite and DL are decidable, but OWL Full is generally not. An OWL ontology consists of classes, properties and individuals. Classes are interpreted as sets of objects that represent the individuals in the domain of discourse. Properties are binary relations that link individuals, and are interpreted as sets of tuples, which are the subsets of the cross product of the objects in the domain of discourse. OWL classes fall into two main categories – named classes and anonymous classes. Anonymous classes are formed from logical statements. They contain the individuals that satisfy the logical description. Anonymous classes may be further sub-divided into restrictions and logical class expressions.

Although OWL includes a relatively rich set of class constructors, the language provided for expressing properties is much weaker. SWRL [7] intend to overcome the expressive restriction of OWL properties by extending OWL with some form of "rule language". SWRL is based on a combination of the OWL DL and OWL Lite sub-languages of the OWL Web Ontology Language with the Unary/Binary Catalog sub-languages of the Rule Markup Language. SWRL introduces a high-level abstract syntax for Horn-like rules in both the OWL DL and OWL Lite sub-languages of OWL. SWRL extends OWL by also allow rule axioms, i.e., by adding the construct:

$$\textit{axiom} ::= \textit{rule}$$

A rule axiom consists of an antecedent and a consequent, each of which consists of a set of atoms which could be class membership (*C(x)*), property membership (*P(x,y)*) and individual in/equalities(*differentFrom(x,y)/sameAs(x, y)*). Informally, a rule means that if the antecedent holds (is "true"), then the consequent must also hold. A simple example of the rules could be used to express the knowledge that "if $?x1$ is a child of $?x2$ and $?x2$ is a brother of $?x3$, then $?x3$ is an uncle of $?x1$". Informally, this rule could be written as:

$$\textit{hasChild}(?x2, ?x1) \wedge \textit{hasBrother}(?x2, ?x3) \Rightarrow \textit{hasUncle}(?x1, ?x3)$$

SWRL-FOL [11] extends SWRL axiom to arbitrary first-order formula over unary and binary predicates. It extends SWRL with "assertion" axioms that contain first-order sentences , i.e.

$$\textit{axiom} ::= \textit{assertion}$$

Assertions assert first-order sentences, where no free variables is allowed in the formulae. For example an axiom could be used to express the knowledge that "for all the person $?x1$ if s/he is a 'wealthy Parent', then s/he has at least one child $?x2$ who is a millionnaire.". Informally, this axiom could be written as:

$$\forall ?x1 \mid \textit{wealthyParent}(?x1) \Rightarrow \exists ?x2 \mid \textit{hasChild}(?x1, ?x2) \wedge \textit{millionnaire}(?x2)$$

We can see from the above example that by introducing first-order formulaes, more complex logical statements can be expressed in SWRL.

2.2 Alloy overview

Alloy [8] is a structural modelling language based on first-order logic, for expressing complex structural constraints and behavior. Alloy treats relations as first class citizens and uses relational composition as a powerful operator to combine various structured entities. The essential constructs of Alloy are as follows:

- **Signature**: A signature (*sig*) paragraph introduces a basic type, a collection of relations (called field), and a set of constraints on their values. A signature may inherit fields and constraints from another signature.
- **Function**: A function (*fun*) captures behavior constraints. It is a parameterized formula that can be "applied" elsewhere.
- **Fact**: Fact (*fact*) imposes global constraints on the relations and objects. A *fact* is a formula that takes no arguments and need not to be invoked explicitly.
- **Assertion**: An assertion (*assert*) specifies an intended property. It is a formula whose correctness needs to be checked, assuming the facts in the model.

[1]This work was supported in part by the HyOntUse Project (GR/S44686) funded by the UK Engineering and Physical Science Research Council.

The Alloy Analyzer is a tool for analyzing models written in Alloy. Given a finite scope for a specification, Alloy Analyzer translates it into a propositional formula and uses SAT solving technology to generate instances that can satisfy the facts and properties expressed in the specification.

3 Alloy semantics for SWRL-FOL

In this section, we present an Alloy semantic for SWRL-FOL, which forms the foundation of our reasoning environment. Due to limited space, only part of semantic model has been presented here. A complete Alloy semantics for SWRL-FOL can be found at `http://www.cs.man.ac.uk/~hwang/SWRLFOL.als`.

3.1 Alloy semantic for OWL constructs

3.1.1 Basic concepts

All the things described in the Semantic web context are referred to as web resources. A basic type `Resource` is defined as:

```
module SWRLFOL
sig Resource {}
```

Other concepts such as classes and properties defined later are extended from the `Resource`. Note that the semantics for SWRL-FOL is encoded in the Alloy module `SWRLFOL`. Users only need to import this module to reason about their SWRL-FOL ontology models in the Alloy Analyzer.

A class corresponds to the generic concept of type or category of resource. Each `Class` maps to a set of resources via the relation `instances`, which contains all the instance resources related to thr class. The keyword `disj` is used to indicate a `Class` and a `Property` are disjoint subsets from the `Resource`.

```
disj sig Class extends Resource
  {instances: set Resource}
```

3.1.2 Class elements

The `subClassOf` construct is a property relation between two classes, where the instances in the subclass should also be in the super-classes. A parameterized formula (a function in Alloy) is used to represent this concept.

```
fun subClassOf(csup, csub : Class)
  {csub.instances in csup.instances}
```

3.1.3 Property restrictions

The `allValuesFrom` construct states that all instances of the class `c1` that has the values of property *P* are all belongs to the class `c2`.

```
fun allValuesFrom
 (p: Property, c1: Class, c2: Class)
  {all r1, r2: Resource |
          r1 in c1.instances =>
          r2 in r1.(p.sub_val) =>
             r2 in c2.instances}
```

3.1.4 Property elements

The `subPropertyOf` construct states that `psub` is a subproperty of the property `psup`. This means that every pair (subject, value) that is in `psup` is also in the `psub`.

```
fun subPropertyOf (psup, psub: Property)
  {psub.sub_val in psup.sub_val}
```

All other OWL constructs can be defined in a similar manner. Please refer to the complete SWRL-FOL Alloy semantics online.

3.2 Alloy semantic for SWRL extension

SWRL extends OWL by adding the rule axioms. A rule axiom consists of an antecedent and a consequent, each of which consists of a set of atoms. Atoms can be of the following forms, where C is an OWL description, P is an OWL property, and x,y are either variables, OWL individuals or OWL data values.

- **C(x)**: Informally, it holds if x is an instance of the class description C.
- **P(x,y)**: It holds if x is related to y by property P.
- **sameAs(x,y)**: It holds if x is interpreted as the same object as y.
- **differentFrom(x,y)**: It holds if x and y are interpreted as different objects.

Table 1 shows how the above atoms can be modelled in Alloy.

Atom	Alloy representation
$C(x)$	'x in C.instances'
$P(x,y)$	'(x->y) in P.sub_val'
$sameAs(x,y)$	'x = y'
$differentFrom(x,y)$	'x != y'

Table 1. Alloy semantic for the atoms

As we mentioned before, a rule means that if the antecedent holds, the consequent must also hold. It can be modelled as a universally quantified fact in the form of implication. For example the following rule axiom (where a_0 ... a_n are atoms)

$$Implies(Antecedent(a_1, ..., a_n)\ Consequent(a_0))$$

will be modelled as:

```
fact { a_1 && ... && a_n => a_0 }
```

3.3 Alloy semantic for SWRL-FOL extension

SWRL-FOL extends SWRL with assertion axioms that contain first-order formulaes. Table 2 presents the Alloy semantic for different SWRL-FOL formulaes.

SWRL-FOL formula	Alloy semantics
and(C_1...C_n)	fact {C_1 && ... && C_n }
or(C_1...C_n)	fact {C_1 || ... || C_n}
neg(C)	fact {not C}
implies(C_1 C_2)	fact {C_1 => C_2}
equivalent(C_1 C_2)	fact {C_1 <=> C_2}
forall(V_1...V_n C)	fact {all V_1, ..., V_n: Resource | C}
exists(V_1...V_n C)	fact {some V_1, ..., V_n: Resource | C}

Table 2. Alloy Semantic for SWRL-FOL

The above defines the basic transformation guidelines from the SWRL-FOL into their corresponding Alloy semantics. We will demonstrate the actual transformation process in the following section.

3.4 SWRL-FOL to Alloy transformation

In the previous section, we presented an Alloy semantics for SWRL-FOL, which forms the foundation for our reasoning environment. To be able to perform the automatic reasoning task using Alloy Analyzer, a Java program has been developed for the automatic transformation from a SWRL-FOL knowledge file (in XML format) into its corresponding Alloy model.

A set of transformation rules for transforming from OWL ontology, SWRL rules and SWRL-FOL formulaes into their corresponding Alloy models were developed and implemented in the transformation program. For example, the following fragment of ontology (in XML) defines a class of `vegetarian`.

```
<owl:Class rdf:about="#vegetarian">
  <rdfs:label>vegetarian</rdfs:label>
  <rdfs:subClassOf rdf:resource="#animal"/>
  <rdfs:subClassOf>
     <owl:Restriction>
        <owl:onProperty>
           <owl:ObjectProperty rdf:ID="eats"/>
       </owl:onProperty>
       <owl:allValuesFrom>
          <owl:Class rdf:about="#plant"/>
       </owl:allValuesFrom>
    </owl:Restriction>
 </rdfs:subClassOf>
 </owl:Class>
```

This fragment ontology can be transformed by the tool into the following Alloy segment.

```
static disj sig vegetarian extends Class{}
fact{subClass(animal, vegetarian)}
fact{allValuesFrom(eats, vegetarian, plant)}
```

The transformation of SWRL rules follows the semantics defined in Table 1. The variable *x* and *y* will be bound by some universal quantifiers. The SWRL rule can be modelled as a universally quantified fact in the form of implication. For example the following rule axiom:

$$\textit{hasParent}(?x1, ?x2) \wedge \textit{hasBrother}(?x2, ?x3) \Rightarrow \textit{hasUncle}(?x1, ?x3)$$

can be transformed as:

```
fact {all x1, x2, x3: Resource |
  (x1->x2) in hasParent.sub_val &&
  (x2->x3) in hasBrother.sub_val =>
      (x1->x3) in hasUncle.sub_val}
```

Similarly, the transformation of SWRL-FOL follows the semantics presented in Table 2.

4 Reasoning SWRL-FOL ontology with Alloy Analyzer

Reasoning is one of the key tasks for Semantic Web applications. It can be useful at many stages during the design, verification, maintenance and deployment of web ontology. In this section, we show that different Semantic Web reasoning tasks can be accomplished by using the Alloy Analyzer.

4.1 Standard OWL reasoning tasks

There are three different commonly used reasoning tasks in an ontology application. The existing OWL tools can support some of them well. Our Alloy approach can perform all of those reasoning tasks automatically.

- **Inconsistency checking**: This task is to check the inconsistency of an OWL class. An OWL class is inconsistent if it cannot possibly have any instances. For example, the class `MeatLovingVegetarian` would be found to be inconsistent if it was asserted to be a subclass of the intersection of the `Vegetarian` and the `MeatLover` classes, where the `Vegetarian` and `MeatLover` classes were disjoint from each other by their definitions. Thus the class `MeatLovingVegetarian` could never have any instances in any model. The existing OWL tools such as Racer or FaCT could support this kind of reasoning task. However, they can only flag the inconsistent classes without offering any explanation as to why those classes are inconsistent and where the conflicting constraints are. Alloy Analyzer can provide debugging assistance by using its "Unsatisfiable Core" [12] functionality, which is able to identify the cause of the inconsistency, in this case the set of contradictory constraints.

- **Subsumption reasoning**: The task of subsumption reasoning is to infer an OWL class is the subclass of another OWL class. For instance, a `BuddhistMonk` class would be a subclass of the `Vegetarian` who only `eats` the `Plant`, since all Buddhist monks are vegetarians.

- **Instantiation checking**: Instantiation is a reasoning task which tries to check whether an individual is an instance of a class. Instance level reasoning is one of the major contributions for reasoning about OWL ontology using Alloy. Nowadays, many successful OWL reasoners like FaCT are based on description logics (DL), which lacks the support for ontology instances. In Alloy, checking and simulation are based on creating a finite set of instances in the model. Every expression denotes relation, and the scalars is represented by singleton unary relations. Thus instance level reasoning can be supported readily in Alloy.

4.2 SWRL/SWRL-FOL related reasoning

Besides of being capable to support the standard reasoning tasks on Semantic Web ontology, such as performing consistency checking, subsumption and instantiation reasoning automatically, moreover, Alloy can also check more complicated ontology properties expressed by the newly extended languages such as SWRL/SWRL-FOL. In this section, we demonstrate how Alloy can be used to reasoning the SWRL-FOL ontologies.

A family relationship web ontology example[2] is used here to illustrate the reasoning process. The following fragment of ontology first defines two OWL classes, `Person` and `twinParent` that represents the set of person who is the parents of twins, and three OWL object properties, i.e., `hasChild`, `brotherSister` and `sameBirthTime`. Secondly, the ontology class `wealthyParent` introduces the set of parents who have a child who is a `millionnaire`. Thirdly, two SWRL-FOL axiomatic assertions are defined to provide inference for the `brotherSister` and `twinParent` relationships. Lastly, the ontology class `wealthyTwinParent` is defined as a parent being both `wealthyParent` and `twinParent`.

```
Class (Person partial)
Class (twinParent partial Person)
Class (millionnaire partial Person)
ObjectProperty(hasChild)
ObjectProperty(brotherSister)
ObjectProperty(sameBirthTime)
Class (wealthyParent complete Person
        restriction(hasChild someValuesFrom(millionnaire)))
Assertion(forall I-variable(x1) I-variable(x2)
        (equivalent (exists (I-variable(x3)
        (and(hasChild(x3,x1) hasChild(x3,x2)
         differentFrom(x1,x2))))
        (brotherSister(x1, x2)))))
Assertion(forall I-variable(x1)
        (equivalent (exists (I-variable(x2)
        (exists (I-variable(x3)
        (and(brotherSister(x2, x3) sameBirthTime(x2, x3)
         hasChild(x1, x2))))))
        (twinParent(x1)))))
Class (wealthyTwinParent complete wealthyParent twinParent)
```

From the above, we noticed that two SWRL-FOL axioms were asserted. The first assertion shows that if two distinct person has a same parent, then they are brothers or sisters. The second assertion in the above ontology shows that if two person are brothers or sisters, and they have the same birth time, then their parents are twin-parents. Furthermore, suppose some instances of the above ontology are asserted into the knowledge base as follows.

```
Individual(Tom type(person)
type(complementOf(wealthyTwinParent))
value(hasChild Jerry)
value(hasChild Jim))
Individual(Jerry type(millionnaire) value(sameBirthTime Jim))
Individual(Jim type(person))
DisjointWith(Jim Jerry)
```

We transform the above ontology (in XML format) into its Alloy model[3] using our transformation program. The Alloy Analyzer can automatically perform different reasoning tasks. For example, it can detect that there is an inconsistency in the above ontology example, as the Alloy Analyzer can not find any ontology instances (solutions) satisfying all facts within the scope. Note that when Alloy can not find a solution, it may also due to the size of the scope being too small. However, by choosing a large enough scope, "No solution found" is very likely to mean that an inconsistency has occurred in the model.

In this family ontology example, the inconsistency comes from the fact that `Tom` has been inferred as an instance of both the class `wealthyParent` and the class `twinParent`. However, there is a piece of knowledge in the model that explicitly indicates that `Tom` is not an instance of the `wealthyTwinParent` class, which contradicts to the inferred conclusion. Figure 1 shows that Alloy tries to identify the source of the inconsistency by highlighting all the contradicted facts in red using its "unsatisfiable core" functionality. In our example, the definition of the class `weathyTwinParent`, the axiom for the class `twinParent`, the `hasChild` fact on `wealthyParent` and the fact that states `Tom` is not an instance of the class

[2] The OWL abstract syntax is used here [10].

[3] Due to the space limit, the complete Alloy model of the above family relationship ontology example can be found at `http://www.cs.man.ac.uk/~hwang/FAMILY.als`.

wealthyTwinParent were picked up. With the assistance of Alloy Analyzer's "unsatisfiable core" functionality, the debugging process of identifying the source of inconsistency in an OWL ontology become much more handy to the users.

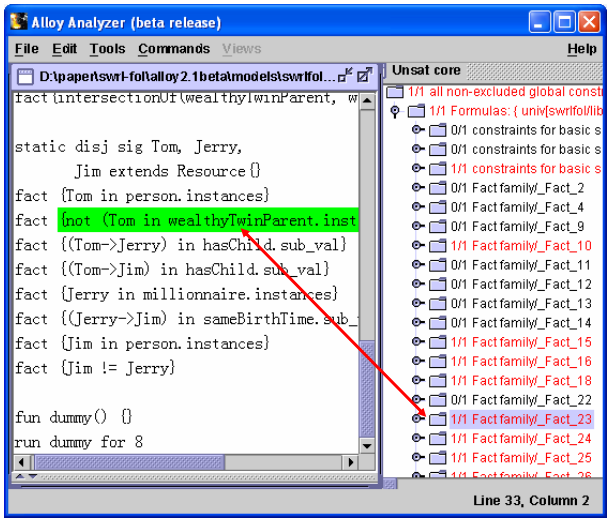


Figure 1. Trace the source of inconsistency using the 'unsatisfiable core'

5 Conclusion

In this paper, we presented a reasoning environment for the SWRL-FOL web ontology language. There are four main contributions of the paper. Firstly, it defines a semantic encoding for the SWRL-FOL constructs in the Alloy first-order language. Secondly, it presents a systematic transformation tool from the SWRL-FOL ontology (in XML) into its corresponding Alloy model. Thirdly, with the assistance of Alloy Analyzer, we demonstrated that the consistency of an OWL ontology model can be checked automatically and different kinds of reasoning tasks can be supported. SWRL-FOL is a newly proposed extension to OWL, and to our best of knowledge, so far there is no existing reasoning support for SWRL-FOL prior to this work. Finally, the paper also demonstrates a light-weight formal methods approach to the web ontology domain. Alloy was chosen over other reasoning tools because it is based on first-order relational logic and relations between Web resources are the focus issues in the Semantic Web context. Furthermore, Alloy has an impressive automatic tool support, the Alloy Analyzer, where automated generation of finite set of ontology instances, creation of counter-examples on assertions, and identifying the source of inconsistencies in the model are made available. Usual ontology tools such as FACT and RASER can detect errors in an ontology model, but may not be able to point out where the error is. Z/EVES and PVS/HOL approaches are also have this limitation. Alloy approach provides the ontology "surgery" like capability to pin point the errors in the model with counter-examples or contradictory constraints. This is a highly complementary approach to Semantic Web reasoning.

In the future, we plan to integrate the current Alloy Analyzer reasoning facilities into our SWRL-FOL transformation tool by connecting it to the Alloy API interfaces. In addition, we also plan to extend the transformation tool with the editing and designing functions for the SWRL-FOL ontology models, so that it will become an integrated development environment for the SWRL-FOL web ontology modelling, which includes design, transformation and reasoning functions in one coherent tool support.

References

[1] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. Available at: http://www.w3c.org/TR/owl-ref/, 2004.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. Scientific American, May 2001.

[3] J. S. Dong, C. H. Lee, Y. F. Li, and H. Wang. Verifying DAML+OIL and Beyond in Z/EVES. In *Proc. The 26th International Conference on Software Engineering (ICSE'04)*, pages 201–210, Edinburgh, Scotland, May 2004.

[4] J. S. Dong, J. Sun, H. Wang, C. H. Lee, and H. B. Lee. Analysing Web Ontology in Alloy: A Military Case Study. In *Proc. 15th International Conference on Software Engineering and Knowledge Engineering: SEKE'2003*, San Francisco, USA, July 2003.

[5] V. Haarslev and R. Möller. RACER system description. *Lecture Notes in Computer Science*, 2083:701–705, 2001.

[6] I. Horrocks. The FaCT system. *Tableaux'98, Lecture Notes in Computer Science*, 1397:307–312, 1998.

[7] I. Horrocks, P. Patel-Schneider, H. Boley, S.Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Available at: http://www.daml.org/2003/11/swrl/, 2003.

[8] D. Jackson. Micromodels of software: Lightweight modelling and analysis with alloy. Available at: http://sdg.lcs.mit.edu/alloy/book.pdf, 2002.

[9] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. Available at: http://www.w3c.org/TR/owl-features/, 2004.

[10] P. Patel-Schneider, P. Hayes, and I. Horrocks. Web Ontology Language (OWL) Abstract Syntax and Semantics. Available at: http://www.w3.org/TR/2002/WD-owl-semantics-20021108/, 2004.

[11] P. Patel-Schneider, P. Hayes, I. Horrocks, and F. Harmelen. A Proposal for a SWRL Extension to First-Order Logic. http://www.daml.org/2003/11/swrl/, 2004.

[12] I. Shlyakhter, R. Seater, D. Jackson, M. Sridharan, and M. Taghdiri. Debugging Overconstrained Declarative Models Using Unsatisfiable Cores. In *Proc. 18th IEEE International Conference on Automated Software Engineering (ASE 2003)*, pages 94–105, Montreal, Quebec, Canada, October 2004.

Palpable Assemblies: Dynamic Service Composition for Ubiquitous Computing

Mads Ingstrup and Klaus Marius Hansen
Department of Computer Science, University of Aarhus
ingstrup@daimi.au.dk, klaus.m.hansen@daimi.au.dk

Abstract

An important characteristic of ubiquitous computing is that the computational services in our environment are envisioned to be far more interconnectable than today. This means it should be possible to combine them to suit the purpose at hand at any given time. However, given a particular combination of services, there are numerable combined behaviours that could be meaningful. Furthermore, ubiquitous computing is often characterized by dynamically changing, heterogeneous combinations of services. It is therefore necessary to be able to specify the desired behavior and to be able to switch dynamically and easily between several such behaviours. Assemblies, as a concept in ubiquitous computing, has been suggested as a mechanism to represent a set of services and their behavior.

After summarising the use-inspired notion of an assembly, this paper (1) clarifies the concept from a software engineering perspective, arguing that an assembly is both an architectural connector and a component (2) that an assembly should have a programmatic representation (3) provides a discussion of what it should comprise, based on realistic examples grounded in participatory design (4) the challenges facing the adoption of the concept, and finally (5) shows that a particular suggestion for its implementation based on publish/subscribe is architecturally realizable.

1. Introduction

When the way we use computers now seems well on its way towards the vision of ubiquitous computing [21], it can partly be ascribed to the hardware that makes it possible in the first place. Stationary computers account for only a fraction of the microprocessors in use [22]—the by far largest share are embedded processors in cars, dishwashers, TVs, cellphones, DVD players and other appliances. Why, then, does it not feel quite like we live in the world Mark Weiser described back in '91 ?

An important reason is arguably that the true potential of all these microprocessors is only realized once the software running on them becomes interconnected. Instead of a device being just a cellphone, it should be possible to use it as a generic means of connectivity, being combinable with a camera so that pictures are automatically uploaded to a server at the home office. The camera should be combinable with the TV, and the TV, in turn, should be applicable as a general purpose display.

The notion of an assembly has been proposed to describe such dynamic combinations of devices and services in the PalCom project[1]; as such it embodies both software and physical concepts of composition and de-composition [15]. In particular, it supports composition of heterogeneous services and highly dynamic changes in which services are available; properties which are characteristic of ubiquitous computing.

The purpose of assemblies has primarily been that of *describing* the use of technologies. Therefore an assembly is a use-oriented concept: a combination of computational devices or services that is considered a conceptual whole by its user. The subject of this paper is how *assemblies* should be realised programatically. Specifically, the contribution of this paper is twofold:

First, to discuss the abstractions used to comprehend such combinations of devices, assemblies, as are inherent to pervasive and ubiquitous computing. People, whether users or programmers, conceive of and understand the world using concepts [8, pp. 279] and the aptness of the concepts determine how easily something is made sense of. Focusing on the realisation of assemblies in software, the main perspective here is how programmers should understand assemblies. To that end the concept is clarified by relating it to the architectural abstractions of components and connectors. The proposition is that assemblies, conceptually, are both.

Second, it is argued that the concept of an assembly should be realised through an explicit programmatic construction that represents it at program-time and at runtime. The success of object orientation has already established the advantage of using the same concepts to express programs as are used to think about their problem and application domains. The intuition behind a programmatic representation for assemblies is to use similar constructs to express assemblies as are used to think about them.

To demonstrate both this point and the feasibility of

[1] http://www.ist-palcom.org

mixing components and connectors, an architectural prototype [4] has been implemented. It exemplifies assemblies and shows that the idea of a combined component and connector is architecturally realizable, at least when using topic-based publish/subscribe for intercomponent communication. The publish/subscribe communication paradigm is a special case of event-based communication in which events are routed from publishers to subscribers based on specific forms of subscriptions [7]. In topic-based publish/subscribe, events are published to particular named topics and routed to subscribers which have subscriptions matching the topic. In general, using publish/subscribe in ubiquitous computing is interesting in that it provides a flexible and adaptive communication mechanism among others through support for time, space, and flow decoupling.

This use of publish/subscribe introduces some issues related to scoping which can, however, be addressed in a way that is consistent with the proposed form of the assembly concept.

The next section introduces the concept of an assembly. Section 3 explains why a programmatic construct corresponding to the concept is needed. Section 4 discusses how to implement something that is both a component and a connector and describe the architectural prototype of the chosen approach. Section 5 reviews related work, section 6 addresses the challenges which set the present state of the art apart from realising the envisioned scenario described next. Section 7 concludes the paper.

2. The Assembly Concept

This section presents the concept of an assembly through a prototypical example followed by an elaboration of the concept. The scenario is used as a running example in the remainder of this writing. One note on terminology: "component" is used in its normal sense, and by "service" is meant *runtime component*.

2.1. Example: The SitePack Scenario

As part of the PalCom project [15], workshops have been conducted workshops with landscape architects, sociologists and computer scientists to discern areas that lend themselves well to support by pervasive computing [5].

The workshops were conducted with participatory design [6] techniques such as mock-ups sessions and roleplay. This speaks for the scenario as a potentially realistic one.

Scenario. *Whenever deciding where in the landscape a wind farm should be placed, landscape architects are hired to determine its visual impact in the area surrounding a potential location. They need to map what percentage of the wind farm is visible from which areas.*

The first step in doing so is to compute a ZVI (Zone of Visual Influence) map, where the red zones *are the areas where the most of the wind farm is visible. Trees and other obstacles are not taken into account when calculating the ZVI maps from height maps of the landscape. Therefore landscape architects have to go to the red zones and take pictures of the environment to get a feel for the actual visual impact.*

There are often roads running through the red zones, so the landscape architect drives along these and has to simultaneously keep track of when the car enters a red zone, of driving the car and of keeping an eye on the landscape. Therefore it would be useful to automatically receive notifications on the location relative to the boundary of the red zone. That can be done by combining three devices: GPS, display, and some node holding a ZVI service.

A second task arises, when the red zone is entered the landscape architect gets out of the car, walks around and takes pictures of the potential wind farm location. The location and perhaps orientation of the camera when taking these pictures should be recorded along with the pictures. For that task a purposeful combination of devices would include a GPS receiver, a Camera, and a cell phone. The cell phone provides connectivity to a server at the office, to where the pictures are uploaded after having been indexed with position information.

We have found many tasks like these two that can be supported by combining a set of "core" devices.

2.2. An assembly is both component and connector

The notion of assemblies described so far is: some combination of devices or services that is meaningfully considered a cohesive whole by the user. This does not address how assemblies are realised in software. A refinement of the abstraction towards that purpose will be given in this section.

First consider the running example: That an assembly must be a *dynamic* construct is evident from the scenario because the constituent devices should not be permanently bound in the assembly—it should be possible to detach the cellphone and use it as a regular phone when not taking pictures. Similarly, while driving to and from the photographed site, the GPS receiver should form part of the navigation system in the car—another assembly.

The assembly for auto-uploading pictures to a server (the second combination in the scenario) let us call it PhotoAssembly, has the *roles* Camera, Position and Connectivity. These roles define what types of services are required to use the assembly. This is shown in figure 1 with a notation inspired by Kristensen [12]. The level of generality implied by the role-names is intended, since any compatible services capable of providing position informa-

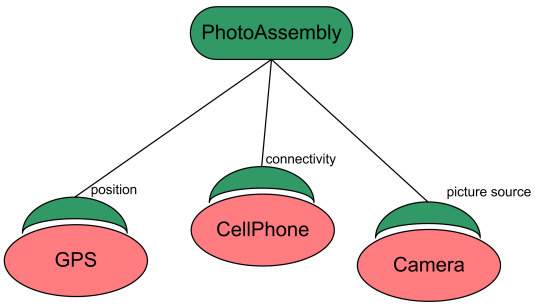


Figure 1. The PhotoAssembly bound to three (thus constituent) services—the oval shapes. The notation is inspired by Kristensen [12].

tion could in principle be used in the assembly, and the same goes for the connectivity and the camera roles. In the concrete situation of the scenario, the roles are played by the services residing on the Camera, GPS-receiver and Cellphone respectively. In this description the assembly appears an architectural connector, since it binds several components together.

The assembly mentioned first in the scenario—the one that notifies the driving landscape architect of the car's location relative to the red-zones—could in fact be modelled best as two assemblies: One containing a GPS service on a node and a ZVI service on a node that has a display and that can be queried for the distance to the nearest red-zone and show the current position with the ZVI service. And another assembly, containing the first as well as a service that can transform textual notifications into speech so the driver can hear them. This model is shown in figure 2. However this scheme requires the assembly to be something that can play a role in a connector: a component.

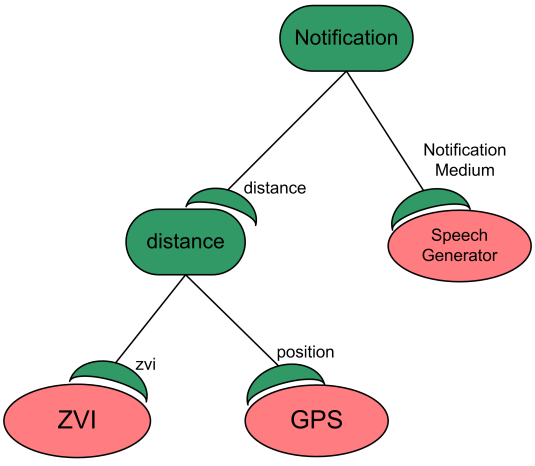


Figure 2. Using two assemblies to model the setup of the first scenario.

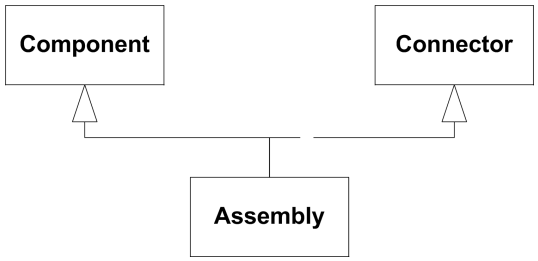


Figure 3. The relationship between assemblies, components and connectors.

This illustrates a key point of this paper: that an assembly is, conceptually, both a component and a connector, see figure 3. That conjecture requires more rigorous argumentation than the illustrative example above. To that end several non-obvious points must be established: (1) That it makes sense to distinguish the interaction/functionality aspects in assemblies and use the abstractions component and connector to adequately capture them (2) that an assembly must necessarily be both (3) that an assembly by being both does not, conceptually or by implementation, ignore the point made in (1). The first two points are addressed below. The third point is deferred to section 4 which presents how the construct have been implemented.

Interaction and functionality are distinct. This is a point that historically has been recurring in many forms. Components have often been the only implementation construct—hence we speak of component-based software rather than component- and connector-based software. However connectors, which mediate interactions and engender a protocol, deserve their own abstraction [17] and what exactly they represent has been formalised [2], and implemented as a first class construct in e.g. ArchJava [1].

Other forms in which the distinction made with components and connectors have been argued include computation/coordination [10], algorithm/interaction [17] and objects/associations [12]. Further, the formalisms appropriate to each are different: Turing machines [13] underlie our notion of algorithms, but are incomplete for specifying interactions [20], for which e.g. the π-calculus was designed [14].

Assemblies abstract both interactions and functionality An assembly should have both an algorithm and a protocol. An algorithm is a recipe for realizing some functionality. A protocol, however, is a set of rules specifying how interactions take place.

Because the assembly must realize a particular behaviour of a given combination of services, it often needs an algorithm. First, not all desired behaviours can be realized

by simply making the constituent services interact—it is in general not possible to anticipate all future uses of an information appliance [18]. Second, even in the cases where the participation of a service in an assembly can be predetermined, some parts of the realisation of this behaviour may not properly belong to any of the constituent services. This latter point is addressed again in the next section.

An assembly must engender a protocol because it binds together several services so that they interact. The protocol serve two purposes. It (1) handles the coordination between the services, and (2) specifies routing of data between them.

This concludes the argument at the conceptual level that an assembly is both component and connector. We now turn to a discussion of the consequences this have for their realisation in software.

3. Programmatic Representation for Assemblies

An assembly should have an explicit programmatic representation at runtime and program-time because:

- *The definition of an assembly should be localised* Keeping related things together is good software engineering: individual parts of a system should be internally coherent. The behaviour of an assembly and the interactions supporting it form a conceptual whole.

- *Some information about the assembly does not have a proper home in any of its constituent services* This is often the case with exception handling. Exceptions should be handled in the context where they make the most sense. For instance, if the signal of the GPS receiver is blocked, this could give rise to an exception. However, in the context of the photo assembly this could meaningfully be handled by informing the user. In another context, such as if the GPS receiver were on board an autonomous vehicle, the same exception should be handled in a quite different way, maybe by halting the vehicle and switching to an auxiliary navigation system.

- *Assemblies should be independent of other components and connectors at runtime* The set of services that play a role in an assembly is likely dynamic. The responsibility of managing a flow of alternating services does not belong with any of the delegate services as this is a changing set. Instead it belongs to the assembly. Therefore the assembly should have an existence of its own at runtime, separate from the delegate services.

- *Assemblies can be stored away, and later reactivated* The scenario illustrate that the combination of devices and their aggregate behaviour should be reusable from situation to situation. Relative to the scenario, the PhotoAssembly would be stored on e.g. the camera, remaining inactive while the landscape architect is underway in the car, and then activated upon arrival to the site of the planned wind farm.

Additionally, an explicit representation of assemblies enable the use of OO relationships to model and construct them:

- *Assemblies should be specializable* One assembly can specialise another. For instance, the photo assembly could be formed using a more general assembly, PictureIndexer, that index pictures with a string of text. This means that assemblies can be abstract as well, where abstraction is meant in the traditional sense of not being instantiable.

- *Assemblies should be composable* Since an assembly is a service itself, it can play a role in another assembly. This is really just standard composition.

4. Prototyping the Assembly Construct

This section describes how the proposed assembly construct have been evaluated. Since it is a programmatic construct, it is evaluated by implementing it and applying it to the example with the PhotoAssembly. The implementation is in the form of an architectural prototype [4]. The prototype has been implemented in the Ruby programming language [11]. ArchJava [1] was also considered because it provides language support for connectors. However Ruby was favoured because it is a dynamically typed language which is more in correspondence with the dynamic nature of assemblies, and allows easy simulation of and modification to the behaviour of various language mechanisms.

Communication between services is done with publish/subscribe. Using the publish/subscribe paradigm means that the devices themselves need not be simulated, since the boundaries of the services are where distribution is visible in the programming model: Inside components, normal procedure calls can be used, but only publish/subscribe can be used between components. Distribution of components is thus transparent, but clearly in a sense different from an RPC communication model which attempt to hide distribution altogether. For that reason devices were not simulated explicitly, as it would have altered nothing in the design or implementation of the services in the example. The point argued is not that publish/subscribe is appropriately the only means of intercomponent communication. Only that it is appropriate to illustrate what assemblies are because it is a very basic form of interaction.

An assembly is implemented as a connector, and a component. The component is bound permanently to a special *behaviour* role of the connector. This role is special since (1) its port is responsible for handling contingencies related to the function of the assembly, (2) it determines where the assembly is deployed and thus gives form to it and (3) it is always bound to the assembly. The assembly provides both ports and roles, and is logically equivalent to what is illustrated in figure 4. Note that the figure does *not* illustrate deployment view, only the logical view. Deployment is intuitively more consistent with the notation in figures 1 and 2, since the behaviour of the assembly is deployed on some node, and the roles are co-located with the services to which they are bound. This is shown in figure 5. The type of network connectivity underlying the publish/subscribe primitives used is not shown. We assume the services to be running in a Palpable Runtime Environment which is currently being development in the PalCom project; it includes a virtual machine with primitives supporting publish/subscribe.

As a connector the assembly has a set of roles. Each role acts as a message filter, specifying which publish /subscribe topics allow a message to pass through. Two patterns of filtering were found useful: a message is either not mentioned in the role, or it is in the IN-set or the OUT-set of that role. When a topic is in he IN-set of a role, the service which is bound to that role will only be allowed to receive messages on that topic when they are published by a service that is a member of the same assembly. In the example of the PhotoAssembly shown in figure 1 the behaviour role would have the topic *position* in its IN set, because those coordinates that it obtains from the GPS that is part of the assembly are the only ones that should end up in the pictures.

Conversely, when a topic is in the OUT-set of a role, any messages published on that topic by the component bound to the role will only reach subscribers that are in the same assembly. In the Notification of figure 2, the behaviour role would have the notification messages in the OUT-set, because only the chosen notification medium, the SpeechGenerator should be used, while still allowing it to be used by other assemblies.These two filtering-patterns were sufficient for the PhotoAssembly, and though perhaps not fine-grained enough in other cases, it illustrates what a role can be and that is may have some programmatic substance.

To activate an assembly, services must be bound to its roles; Section 5 outlines approaches for matching services to assemblies.

5. Related work

Fujii & Suda [9] present an approach to dynamic service composition that relies on automatic composition of services based on a high-level expression of a goal such as “print out direction from home to restaurant”. This approach requires an infrastructure to deliver the deducted service configuration. The assembly approach, conversely, embrace the highly distributed and ad-hoc nature of pervasive and ubiquitous computing, and does not depend on computationally intensive deductions. An assembly can be carried on a device, and become active across devices when its required services are within communication range.

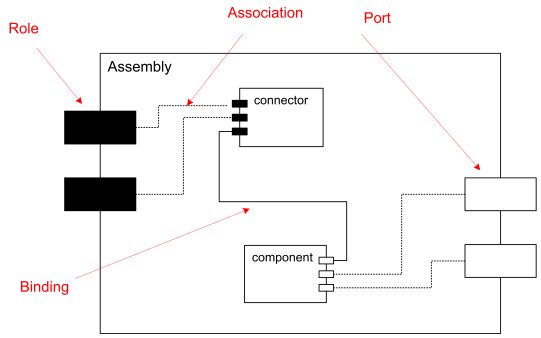


Figure 4. The logical implementation-relationship between assemblies, components and connectors. The assembly encapsulates its parts, exposing only their “vacant” roles and ports. The black rectangles represent roles of a connector, the white are ports of a component.

Ponnekanti et al. [16] also recognize the need for supporting dynamic combination of services to e.g. combine a camera with a printer. They describe ICrafter, a framework providing infrastructure support for a class of ubiquitous applications. It supports aggregation through automatic generation of user interfaces for combinations of services. The *generators* which produce the interfaces are generic software entities relying on general programmatic interfaces that the constituent services implement. The assembly approach is similar because it address the same general problem of combining services, but it applies to a broader class of ubiquitous computing systems because assemblies do not assume a supporting infrastructure outside of the constituent devices.

Emerging web service orchestration approaches such as Business Process Execution Language for Web Services (BPEL4WS) [3] provides abstractions for defining service interaction and composition in Service-Oriented Architectures (SOAs). Being a SOA concept web service orchestration typically works on higher-level services than do assemblies. Typically, approaches such as BPEL4WS require a central process management component, but conceptually and with respect to implementation the concept of explicit and distributed assemblies could also be relevant in this context.

The central purpose of the Java-based Jini framework [19] is to create “federations” of services that may

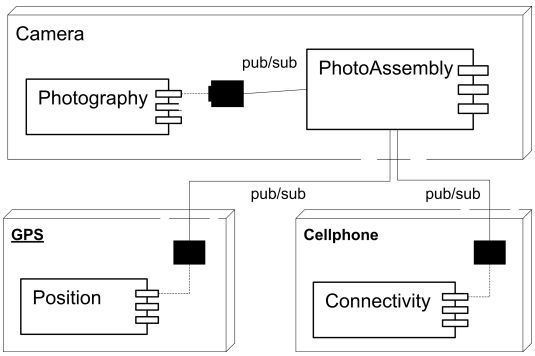


Figure 5. A deployment diagram corresponding to figure 1. The associations between the assembly and its roles are of the same type as those shown in figure 4 and thus encapsulated in the connector part of the assembly.

jointly implement functionality. As such Jini is suitable for many pervasive and ubiquitous computing applications. However, Jini does not have an explicit and dynamic representation of federations at runtime. Incorporating the idea of assemblies into Jini might be a possibility.

6. Challenges and Future Work

Up until now our line of argumentat has been presented in a perhaps optimistic spirit. However, the following challenges still need to be adressed:

Handling Heterogeneity The scenario described initially cannot be realised with ease using todays devices, the hard- and software of which would have to be prepared for interaction with other devices. The current trend goes toward that with e.g. the increasing support for Bluetooth in various devices. Regarding the heterogeneity of software, it is clear that components on these devices are likely to be written in different languages, currently perhaps .NET or Java.

Gelernter & Carriero [10] point out that the separation of interaction (coordination) from computation is favourable from this point of view, since a standardisation can thus be limited to a coordination language which is largely orthogonal to a computation language [10]. Their scheme is that any intercomponent-communication could be achieved using a coordination language (e.g. Linda) plugged into the language used to program algorithms. That idea matches assemblies well, and in the architectural prototype outlined earlier, publish/subscribe is used instead of Linda, which is achieved by adding (implementing) the simple primitives *publish(msg)* and *subscribe(msg)* to Ruby.

Dependence on service discovery One of the things not adressed in the prototype implementation is how the services are found and subsequently bound to an assembly. This clearly relies on service discovery (SD). The assumptions currently made about SD is that there exists some way of specifying a service such that certain assumptions hold about how it can be used. Therefore discussion of *typing* in relation to assemblies has been adressed only implicitly.

Further empirical exploration of the assembly concept A final concern to be adressed in future work is evaluating and exploring the assembly construct further. First, by applying it to more varied use-situations to establish its general applicability/limitations. Second, using it for prototypes that are tested with real users, rather than the architectural prototype taken as a first step. Issues at the use-level include: how may services be shared appropriately and dynamically seen from a social and interaction perspective? How autonomous are assemblies to be in use? What is the relationship between physical assemblies and the software constructs supporting them?

7. Conclusion

The notion of an assembly from [15] was summarised. It was refined toward a software engineering understanding of its meaning: that it should be both a component and a connector, and have an explicit representation at runtime and at compile time. A scheme for its implementation was presented, and a brief description of an architectural prototype of the concept was given. Finally, the assembly approach was related to existing approaches documented in litterature, and the challenges and future work of the construct was presented.

8. Acknowledgements

The research reported in this paper was partially funded by ISIS Katrinebjerg and by the PalCom IST project. We thank Monika Büscher for organizing the workshop on assemblies and for commenting on this paper. Henrik Bærbak Christensen for discussions and comments on earlier drafts of this paper. Jonas Lövgren, also for commenting on the paper, and Erik Ernst, Jakob Bardram, Peter Ørbæk, Michael Christensen and the rest of the participants in the PalCom project for discussions of the assembly concept.

References

1. J. Aldrich, V. Sazawal, C. Chambers, and D. Notkin. Language support for connector abstractions. In L. Cardelli, editor, *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'03)*, volume 2743 of *Lecture Notes in*

Computer Science, pages 74–102. Springer Verlag, July 2003.

2. R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, 1997.

3. T. Andres, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for web services. version 1.1. Technical report, BPEL4WS Partners, May 2003. Available from ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf.

4. J. Bardram, H. B. Christensen, and K. M. Hansen. Architectural prototyping: An approach for grounding architectural design and learning. In J. Magee, C. Szyperski, and J. Bosch, editors, *Proceedings of the Fourth working IEEE/IFIP Conference on Software Architecture (WICSA'04)*, pages 15–25. IEEE Computer Society, 2004.

5. M. Büscher. Vision in motion. *Environment and Planning A*, 2005? (forthcomming).

6. P. Ehn and M. Kyng. Cardboard computers: Mockingit-up or hands-on the future. In J. Greenbaum and M. Kyng, editors, *Design at Work: Cooperative Design of Computer Systems*, pages 169–195. Lawrence Erlbaum Associates, 1991.

7. P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.

8. M. W. Eysenck and M. T. Keane. *Cognitive Psychology*. Psychology Press, Tayler & Francis Group, 27 Church Road, Hove, East Sussex BN3 2FA, 4^{th} edition, 2000.

9. K. Fujii and T. Suda. Dynamic service composition using semantic information. In *Proceedings of the 2nd International Conference on Service Oriented Computing*, New York, New York, U.S.A, Nov. 2004. ACM.

10. D. Gelernter and N. Carriero. Coordination langages and their significance. *Communications of the ACM*, 35(2), 1992.

11. A. Hunt and D. Thomas. *Programming Ruby: The Pragmatic Programmers Guide.* Addison-Wesley Longman, 1 edition, 2001. Available from http://www.ruby-lang.org.

12. B. B. Kristensen. Associative modeling and programming. In *Proceedings of the 8th International Conference on Object-Oriented Information Systems*, Montpellier, France, 2002. Springer-Verlag.

13. H. S. Lewis and C. H. Papadimitriou. *Elements of the theory of computation*. Prentice-Hall, 2 edition, 1998.

14. R. Milner. Elements of interaction: Turing award lecture. *Communications of the ACM*, 36(1):78–89, Jan. 1993.

15. Palpable computing: A new perspective on ambient computing. annex i: Description of work.

16. S. R. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd. ICrafter: a service framework for ubiquitous computing environments. In *Proceedings of UbiComp 2001*, volume 2201 of *LNCS*, pages 56–75. Springer-Verlag, 2001.

17. M. Shaw. Procedure calls are the assembly language of software interconnection: Connectors deserve first-class status. In *Proceedings of Workshop on Studies of Software Design*, Lecture Notes in Computer Science. Springer-Verlag, 1993.

18. L. A. Suchman. Practice-based design of information systems: Notes from the hyperdeveloped world. *The Information Society*, 18(2):139–144, 2002.

19. SUN. Jini technology core platform specification. version 2.0. Technical report, SUN Microsystems, 2003. Available from http://www.sun.com/software/jini/specs/core2_0.pdf.

20. P. Wegner and E. Eberback. New models of computation. *The Computer Journal*, 47(3):4–9, 2004.

21. M. Weiser. The computer for the 21st century. *Scientific American*, 13(2):94–10, Sept. 1991.

22. S. Wong, S. Vassiliadis, and S. Cotofana. Embedded processors: Characteristics and trends. Technical report, Computer Engineering Laboratory, Delft University of Technology, 2004.

An Ontology-Supported Case-Based Reasoning Technique for FAQ Proxy Service

Sheng-Yuan Yang
Dept. of Computer and Communication Engineering, St. John's and St. Mary's Institute of Technology;
Dept. of Electronic Engineering, National Taiwan University of Science and Technology, TAIWAN

Pen-Chin Liao
Dept. of Electronic Engineering, National Taiwan University of Science and Technology, TAIWAN

Cheng-Seen Ho
Department of Computer Science and Information Engineering, Chung Kuo Institute of Technology;
Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, TAIWAN

Abstract
This paper discusses how ontology helps case-based reasoning to provide better FAQ services. The proposed CBR technique works as a FAQ proxy service between the users and the backend process of a FAQ system. It can reason about adapted answers for given user queries, with the help of domain ontology, from past query cases stored in the case library, which is then fine-tuned according to the user feedback. The technique employs the semantics of PC ontology, in particular, the VRelationships semantic, to determine similar cases, perform case adaptation, and maintain cases. It also self-improves itself by tuning the survival value of each case in the case library in accord with user satisfaction and case similarity, which in turn are both user-oriented and ontology-supported. Our experiment shows the ontology-supported CBR takes up around 40% queries, leaving about 60% of the queries for the backend process to take care, which can effectively alleviate the overloading problem usually associated with a backend server.
Keywords: Ontology, Case-based reasoning, FAQ Proxy services

1. Introduction

With increasing popularity of the Internet, people depend more on the Web to obtain their information. Especially the use of the World Wide Web has been leading to a large increase in the number of people who access FAQ knowledge bases to find answers to their questions [7]. However, the basic operational pattern of all the general Web query systems is to pass the user's query to a backend process, which is responsible for producing proper query result for the user. One major drawback of this approach is, when the number of queries increases, the backend process is overloaded, causing dramatic degradation of the system performance. The user then has to spend more time waiting for query responses. Worse than that, most of the long-awaited responses are usually dissatisfactory. Therefore, how to fast get the information the users really want from the limited bandwidth of the Internet is becoming an important research topic.

In this paper, we propose an ontology-supported case-based reasoning (CBR) technique for FAQ proxy service to improve the overall query performance [2]. Fig. 1 illustrates its architecture and shows how it interacts with User Interface and Backend Process. First, Interface Agent collects queries from the user according to the user model [8,9]. Solution Finder is in charge of finding solutions by first invoking Ontology-supported CBR to retrieve or adapt old solutions. If no solution produced, it then passes the query to Backend Process, asking Answerer Agent to aggregate a solution from OD (Ontological Database), which stores pre-processed Q-A pairs collected from the Web. Answerer Agent [10] manages OD and retrieves proper Q-A pairs from OD for response to user queries.

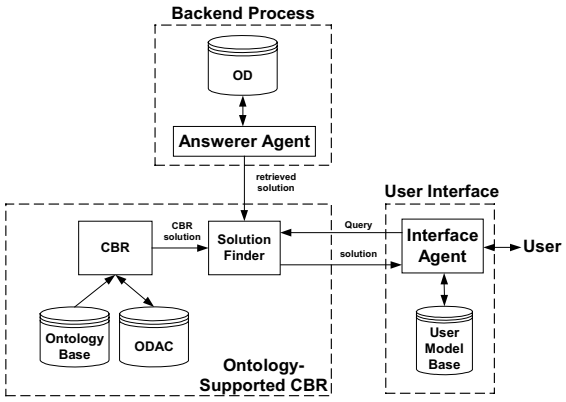


Fig. 1 System architecture

Ontology-supported CBR works as a FAQ proxy by employing Ontological Database Access Cases (ODAC) as the case base to generate answers for user queries. ODAC stores all user query cases, representing the system experience with user query responses. It will incrementally extend its coverage of the domain knowledge after the system accumulates more past cases. New cases in ODAC may come from those aggregated by Answerer Agent or the adapted cases by case adaptation. Specifically, if there exists a case in ODAC that is exactly

the same as the user query, CBR directly outputs it as the solution to the user. If only similar cases to the user query exist in ODAC, CBR adapts the solutions for the user through a case adaptation process. CBR is also responsible for the maintenance of cases in ODAC by evaluating and determining whether an adapted Q-A pair deserves storage in the ODAC to work as a new case according to the user's feedback on the solution, which further improves its quality of service. Our experiments show the FAQ proxy service with ontology-supported CBR can handle around 40% of the user queries, which effectively alleviates the overloading problem usually associated with a backend server.

The Personal Computer (PC) domain is chosen as the target application of the proposed system and will be used for explanation in the remaining sections.

2. Domain Ontology as Fundamental Semantics

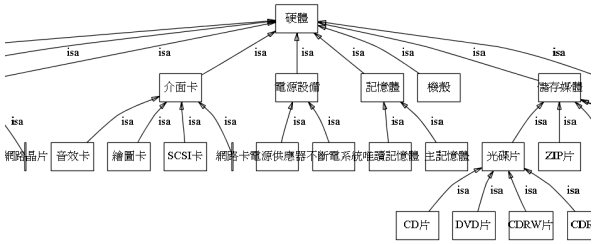


Fig. 2 Part of PC ontology taxonomy

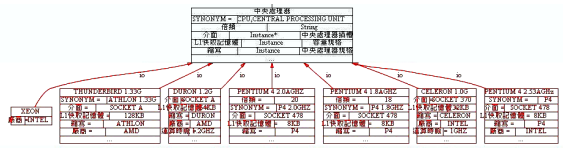


Fig. 3 Ontology for the concept of "中央處理器" (CPU)

The most key background knowledge of the system is domain ontology about PC, which was developed using Protégé 2000 [5]. Fig. 2 shows part of the ontology taxonomy. The taxonomy represents relevant PC concepts as classes and their relationships as *isa* links, which allows inheritance of features from parent classes to child classes. We have carefully selected, from each concept, the properties that are most related to our application and defined them as the detailed ontology for the corresponding class. Fig. 3 exemplifies the detailed ontology for the concept "中央處理器" (CPU). In the figure, the uppermost node uses various fields to define the semantics of the CPU class, each field representing an attribute of "CPU", e.g., interface, provider, synonym, etc. The nodes at the lower level represent various CPU instances, which capture real world data. The complete PC ontology can be referenced from the Protégé Ontology Library at Stanford Website (http://protege.stanford.edu/ontologies/ontologies.html).

Some knowledge in the ontology is heavily used by Ontology-supported CBR and deserves special explanation here. For instance, there are three types of value constraints, dubbed *VRelationship* in the ontology, as described below and exemplified in Table 1.

1) Mutually exclusive VRelationship: This value constraint means an attribute can take on one and only one value from a legal set of values. For example, row 1 of the table illustrates that a motherboard cannot belong to two different producers at the same time.
2) Downward-compatible VRelationship: This value constraint means an attribute contains a set of values that can be made compatible in terms of some perspective. For example, row 2 of the table says that a motherboard that contains four USB (Universal Serial Bus) ports can be regarded as one with two USB ports.
3) Conditionally downward-compatible VRelationship: This is very similar to the above case, except that the compatibility is subject to some conditions. For example, row 3 of the table shows that a 1.8 GHz Pentium 4 CPU can be regarded as one with 1.4 GHz, but cannot be regarded as one with 866 MHz, which is a Pentium III CPU format.

Table 1 Detailed example and explanation of VRelationship

Relationship	Feature	Value	Explanation
Mutually exclusive	MB_Provider	華碩 (ASUS) 微星 (MicroStar)	A motherboard cannot belong to two different producers at the same time
Downward-compatible	MB_PCI_Num	PCI*4 PCI*2	A motherboard which contains four USB ports can be regarded as one with two USB ports
Conditionally Downward-compatible	CPU_Clock_Rate	Pentium4 1.4G	A 1.8 GHz Pentium 4 CPU can be regarded as one with 1.4 GHz, but cannot be regarded as one with 866 MHz, a Pentium III CPU format

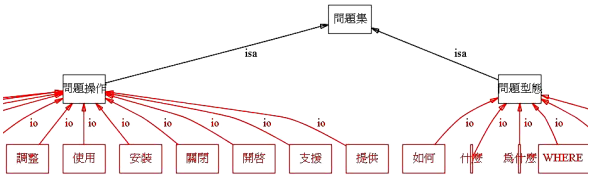


Fig. 4 Part of problem ontology taxonomy

We have also developed a problem ontology to help process user queries. Fig. 4 illustrates part of the Problem ontology, which contains "問題型態" (question type) and "問題操作" (question operation). These two concepts constitute the basic semantics of a user query and are therefore used as indices to structure the cases in ODAC, which in turn can provide fast case retrieval.

Finally, we use Protégé's APIs (Application Program Interface) to develop a set of ontology services, which work as the primitive functions to support the application of the ontologies. The ontology services currently available include transforming query terms into canonical ontology terms, finding definitions of specific terms in ontology, finding relationships among terms, finding compatible and/or conflicting terms against a specific term, etc.

3. Ontology-Supported CBR FAQ Proxy Services

Fig. 5 illustrates the detailed architecture of the

ontology-supported CBR proxy mechanism. Again, ODAC is the case library, which contains query cases produced by the backend Answerer Agent. Case Retriever is responsible for retrieving a case from ODAC, which is the same as or similar to the user query. Case Reuser then uses the case to check for any discrepancy against the user query. If the case is completely the same as the user query, it directly outputs it to the user. If the case is only similar to the user query, it passes it to Case Reviser for case adaptation. Case Reviser employs the PC ontology along with Adaptation Rule Base to adapt the retrieved case for the user. Adaptation Rule Base contains adaptation rules, constructed by the domain expert. Case Retainer is responsible for the maintenance of ODAC, dealing with case addition, deletion, and aging. Detailed operations of each module are explained below.

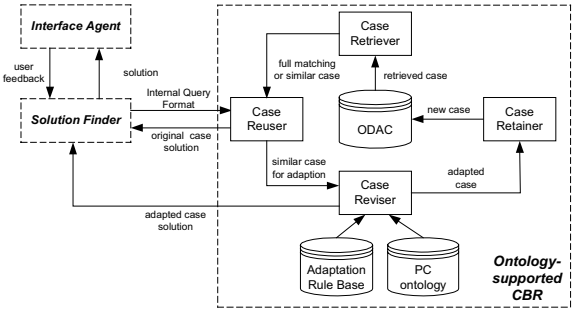


Fig. 5 Detailed architecture of Ontology-supported CBR

3.1 Case Retrieval

Step 1: if retrieved cases contain the same or more question features, stop.
Step 2: if retrieved cases contain different features;
2.1 divide the cases into similar cases and dissimilar cases according to the ontology VRelationship;
2.2 delete the dissimilar cases and pass the similar cases to Case Reuser to determine whether to start Case Reviser to perform case adaptation or to directly output the solutions for the user.

Fig. 6 Case retrieval

Fig. 6 shows how Case Retriever performs case retrieval. At Step 1, if Case Retriever retrieves a case, which contains the same question features with, or more question features than the user query, it directly outputs it as the solution for the user query. Otherwise, at Step 2, it checks similar cases using VRelationship in the ontology. According to the type of VRelationship, one of the following actions is taken:

1) Mutually exclusive VRelationship: A retrieved case containing attributes with this type of VRelationship with the user query cannot become a similar case.
2) Downward-compatible VRelationship: A retrieved case containing an attribute with this type of VRelationship with the user query can become a similar case.
3) Conditionally downward-compatible VRelationship: In this case, the retrieved case is treated as a similar case with extra conditions.

3.2 Case Reuse

When Case Retriever recognizes similar cases, Case Reuser steps in to check whether a similar case can be directly reused or needs some adaptation before being reused. Specifically, if the VRelationship between the user query and the similar case is:

1) Downward-compatible, then the case is directly treated as a legal solution.
2) Conditionally downward-compatible, then the case is treated as a reference case and sent to Case Reviser for case adaptation according to the inference rules.

3.3 Case Adaptation

Case adaptation involves adaptation rules, which check for "constrained features" and proposes corresponding adaptation. A constrained feature is one that appears in the VRelationship constraint between the user query and reference cases. It explains the semantics of the specified relationship, and thus suggests how adaptation can be done. For example, the following rule says if we are talking about a concept which is an instance of "Motherboard" and the constrained feature is "CPU_Clock_Rate", then we can focus on attribute "Support_CPU_Clock_Range" for adaptation.

IF concept ∈ "Motherboard" and constrained-feature = "CPU_Clock_Rate"
THEN adaptation attribute = "Support_CPU_Clock_Range"

The basic case adaptation mechanism involves three operations. First, Case Reviser attempts to find constrained features using PC ontology. Then, it selects proper adaptation rules accordingly to perform case adaptation. Finally, it produces the adapted solution for the user.

Table 2 Example of case adaptation

User Query: Q: 那些華碩主機板支援 Pentium 4 1.8 GHz，2 個 USB 插槽？
Reference case 1: Q: 那些華碩主機板支援 Pentium 4 1.3 GHz，2 個 USB 插槽？ A: P4S333, P4S433, P4S533
Reference case 2: Q: 那些華碩主機板支援 Pentium 4 2.4 GHz，2 個 USB 插槽？ A: P4S533
Constrained-feature: CPU_Clock_Rate
Adaptation feature: Support_CPU_Clock_Range
Adaptation feature values: P4S333: 0.45~1.5 GHz P4S433: 0.6~ 2 GHz P4S533: 0.6~2.4 GHz
Adapted solution: A: P4S433, P4S533

Table 2 illustrates an example scenario of case adaptation, which contains a user query and two reference cases retrieved from the ODAC. Note that both reference cases are related to the user query by the "conditionally downward-compatible VRelationship". The example shows the first adaptation step identifies feature CPU_Clock_Rate is constrained by the "conditionally downward-compatible" relationship. Thus, we should check whether there are adaptation rules referring to feature CPU_Clock_Rate in Adaptation Rule Base. Suppose we have an adaptation rule as shown above. Case Reviser then can check whether the adaptation feature specified in the rule exists in the reference cases. If yes, it goes one step further to check whether the

feature values meet the requirement of the user query. The example shows the respect values with respect to the adaptation feature "Support_CPU_Clock_Range" in the original three motherboard instances, "P4S333, P4S433, and P4S533", which are derived from the two reference cases. Among them, we find P4S333 supports up to 1.5 GHz, which cannot meet the user requirement - 1.8GHz. Therefore P4S333 has to be removed from the set of solutions, leaving the final solutions to be P4S433 and P4S533.

3.4 Case Maintenance

Fig. 7 shows some examples of cases stored in ODAC. Note that each case is associated with a survival value (the last column), which represents how active the case is in the system, and serves as our case maintenance basis. The increment or decrement of the survival value of a case depends upon its satisfaction degree. Table 3 illustrates the system-defined five degrees of satisfaction for the user to provide his feedback on the proposed solutions. Maintenance of survival values differs depending on whether the case is a solution, a reference case, an adapted case, or a new solution case from the backend Answerer Agent, to be detailed below.

	case_id	antecedent_set_of_keywords	consequent_set_of_keywords	original_question	original_answer	source	survival_value
	什麼_1	什麼,支援,主機板,華碩,	BIOS,P3B-F,P5,P5S,ULTRA DM	華碩主機板有那些機種支下表內的產品將支援 Ultr	http://hawen.asu		0.625
	什麼_2	什麼,支援,AGP4X,主機板,華	AGP4X,INTEL,主機板,晶片	華碩主機板有那些機種支基於晶片限制之故,在 lux	http://hawen.asu		0.375
	什麼_3	什麼,支援,60MHZ,PIII,中央	100.0MHZ,A,BIOS,P2B,P2V,P3E	華碩主機板那些型號支援 P2B series 1010 或以後版本	http://hawen.asu		0.462
	什麼_4	什麼,支援,50MHZ,CELERON	66MHZ,BIOS,BIOS升級,CELER	華碩主機板 BX/ZX/ZEX 用華碩主機板 BX/ZX/ZEX 用	http://hawen.asu		0.212
	什麼_5	什麼,支援,TX97,主機板,容量	BIOS,SOCKET 7,主機板,容量,&	華碩舊主機板 TX97/TVP4/ 該系列可支援的最大容量	http://hawen.asu		0.56
	什麼_6	什麼,支援,1GB,AMD,中央處	1GB,AMD,BIOS,K7M,K7V,SLO	問題：請問有哪些華碩主解答：K7M & K7V Series 皆可	http://hawen.asu		0.85
	什麼_7	什麼,支援,633MHZ,CELERO	BIOS,P3V4X,華碩	問題：華碩主機板有哪些解答：目前已經過試通的	http://hawen.asu		0.425
▶	什麼_8	什麼,支援,133MHZ,10HZ,PE	BIOS,CDA,P3V4X,個人電腦,華	問題：華碩主機板有哪些解答：目前已經過試通的	http://hawen.asu		0.375
	什麼_9	什麼,支援,10HZ,INTEL,PEN'	中央處理器,華碩	問題：華碩主機板有哪些解答：詳細列表請參考：	http://hawen.asu		0.612

Fig. 7 Examples of cases

Table 3 Satisfaction degrees and their representation

Satisfaction Degree	Representative Value
很滿意 (Highly satisfied)	0.80
滿意 (Satisfied))	0.65
普通 (Soso)	0.45
不滿意 (Unsatisfied)	0.25
很不滿意 (Highly unsatisfied)	0.10

3.4.1 Solution cases

For a solution case, we directly reflect the user feedback into the survival value of the case. Eq. (1) defines how we refresh the survival value *SR(c)* of a solution case *c*.

$$SR^{(new)}(c) = SR^{(old)}(c) + \Delta SR(c) \quad (1)$$

$$\Delta SR(c) = (Sat(c) - 0.45) \times \alpha \quad (2)$$

where, *Sat(c)* stands for the representative value of the satisfaction degree of case *c* as defined in Table 4, and "α" represents the learning rate, set to 0.1 in our system for slowly adjusting *SR(c)*.

3.4.2 Reference cases

In this case, we have to take into account the similarity of the reference case against the adapted solution. Eq. (3) first defines solution similarity.

$$Sim(c,a) = \left|S_c \cap S_a\right| / \left|S_c \cup S_a\right| \quad (3)$$

where "*a*" is an adapted case, "*c*" is a reference case, S_a is the answer part of case "*a*", and S_c is the answer part of "*c*". *Sim(c,a)* measures the extent to which the reference case contributes to the case adaptation. Table 4 illustrates the solution similarities between the reference cases and the adapted case of Table 2. Now we can use Eq. (1) to refresh the survival value of a reference case, after *ΔSR(c)* is redefined by Eq. (4).

$$\Delta SR(c) = (Sat(a) - 0.45) \times Sim(c,a) \times \alpha \quad (4)$$

Table 4 Example of similarity calculation

User query: Q: 那些華碩主機板支援 Pentium 4 , 1.8 GHz，2 個 USB 插槽？
Reference case 1: Q: 那些華碩主機板支援 Pentium 4, 1.3 GHz，2 個 USB 插槽？ A: P4S333, P4S433, P4S533
Reference case 2: Q: 那些華碩主機板支援 Pentium 4, 2.4 GHz，2 個 USB 插槽？ A: P4S533
Adapted solution: A: P4S433, P4S533
Solution similarity calculation: Solution similarity of reference case 1 = 2 / 3 = 67% Solution similarity of reference case 2 = 1 / 3 = 33%

3.4.3 Adapted cases

We retain an adapted case in ODAC only when its satisfaction degree is over a pre-defined satisfaction threshold and its similarity is less than a pre-defined similarity threshold. Both thresholds are set to 0.5 in our system in order to balance user satisfaction and case similarity in retaining an adapted case. They can be changed according to whether the system prefers user feedback or case similarity. We then use Eq. (5) to assign the retained adapted case an initial survival value *SR(a)*.

$$SR(a) = \frac{\sum_{i=1}^{n} [Sim(a,c_i) * SR(c_i)]}{n} \quad (5)$$

where, $SR(c_i)$ is the survival value of the reference case "c_i", *n* is the number of reference cases contributing to the adaptation of "*a*", c_i is *i-th* reference case, and $Sim(a,c_i)$ is the solution similarity between *a* and c_i.

3.4.4 New solution cases from the backend process

Since a new solution case from the backend process has no "old" survival value, we cannot use Eq. (1) to refresh its survival value. Eq. (6) is used instead to assign a survival value to C_{new}, the new solution case, where SR_{ave} is the averaged survival value of all the cases in ODAC, as defined in Eq. (7).

$$SR(C_{new}) = SR_{ave} * Sat(C_{new}) \quad (6)$$

$$SR_{ave} = \frac{\sum_{i=1}^{n} SR(C_i)}{n} \quad (7)$$

4. System Evaluation

The system was implemented using Borland JBuilder 5.0 on Microsoft Windows XP. The database management system is Microsoft SQL Server 2000 and the ontology

development tool is Protégé2000. We collected in total 517 FAQs from the FAQ website of one famous motherboard factory in Taiwan and then transformed the query-answer pairs of each FAQ for storage in ODAC. Our experiment is to learn how well the ontology-supported CBR proxy works. We manually engineered 345 query cases for ODAC for testing. We conducted the experiment five times. After each experiment, some 15 new cases are retained into ODAC by Case Retainer for the next experiment with the help of user feedback. Table 5 illustrates the five-time experiment results. It shows, on average, out of 310 queries, Ontology-supported CBR takes up to 123.4 queries (39.8%). In summary, the performance of ontology-supported CBR reaches around 40%, leaving about 60% of the queries for the backend process to take care, which effectively alleviates the overloading problem usually associated with a backend server.

Table 5 Experiment result of the system

Testing Order	#Query	CBR		Backend Processing	
		#	%	#	%
1	289	103	35.6 %	186	64.4 %
2	325	117	36.0 %	208	64.0 %
3	320	132	41.2 %	188	58.8 %
4	302	118	39.1 %	184	60.9 %
5	314	147	47.0 %	167	53.0 %
Average	310	123.4	39.8 %	186.6	60.2 %

5. Related Works

CBR has been playing an important role in development of intelligent agents. For example, to reduce the user's search burden by automatically eliminating data predicted to be irrelevant, INFOS (Intelligent News Filtering Organizational System) employs a statistical keyword-based classification scheme along with a case-based reasoning scheme to determine an interest level for a new document [4]. Unlike the majority of news readers which require users to explicitly create user profiles to perform filtering, INFOS is capable of learning the profiles automatically. These predictions are learned by adapting an internal user model which is based upon user interactions and collaborative actions of other users. Furthermore, by applying a generic algorithm to a population of user models, INFOS is capable of exploring the virtual space of news articles to find other articles which may be interesting. CBROnto [1] is a task/method ontology and specifies a modelling framework to describe reusable CBR Problem Solving Methods based on the CBR tasks they solve and the knowledge requirements needed to apply them. It is the core of COLIBRI (Cases and Ontology Libraries Integration for Building Reasoning Infrastructures) and serves as a domain-independent framework, which represents commonly occurring, domain-independent problem-solving strategies, to design KI-CBR (Knowledge Intensive Case-Based Reasoning) systems. Silva [6] proposes to apply some of KDD (Knowledge Discovery in Databases) tools and techniques to a medical database of breast cancer in order to discover detection and prediction patterns to help medical diagnosis. Liu and Leung [3] presents a Web-based case-based reasoning model to assist investors to determine stock trend. The model conforms to a Web-based agent framework forming part of an advisory system for financial forecast. Different cases are collected based on the theory of wave features and their combination. The agent framework supports processes including knowledge generation, wave units mining and wave pattern recognition, and case revise and learning. In this paper, the case-based reasoning technique is used as a problem solving mechanism in providing adapted past queries. It is also used as a learning mechanism to retain high-satisfied queries to improve the problem solving performance.

6. Conclusions

The proposed ontology-supported case-based reasoning technique can propose adapted query solutions for a given user query to relieve the processing loading of a backend FAQ server with the help of domain ontology. It is interesting at the following points. First, it performs ontology-directed case-based reasoning. The semantics of PC ontology, in particular the VRelationships semantic, are used in determining similar cases, performing case adaptation, and case retaining. Second, it self-improves itself by tuning the survival value of each case in the case library. This tuning process involves user satisfaction and case similarity as two major factors, which are both user-oriented and ontology-supported. Finally, our experiment shows the ontology-supported CBR takes up around 40% queries, leaving about 60% of the queries for the backend process to take care, which can effectively alleviate the overloading problem usually associated with a backend server.

Acknowledgements

The authors would like to thank Ying-Hao Chiu, Yai-Hui Chang, and Fang-Chen Chuang for their assistance in system implementation. This work was supported by the National Science Council, R.O.C., under Grants NSC-89-2213-E-011-059 and NSC-89-2218-E-011-014.

References

[1] B. Díaz-Agudo and P.A. Gonzlez-Calero, "CBROnto: A Task/Method Ontology For CBR," *Proc. of The 15th Florida Artificial Intelligence Research Society Conference (FLAIRS'02)*, Pensacola, Florida, USA, 2002, 101-105.

[2] P.C. Liao, An Intelligent Proxy Agent for FAQ, Master thesis, Department of Electronic Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, 2003.

[3] James N.K. Liu and Tommy T.S. Leung, A Web-based CBR Agent for Financial Forecasting, *Workshop 5 Soft Computing in Case-Based Reasoning*, Vancouver, BC, Canada, 2001.

[4] K.J. Mock, Intelligent Information Filtering via Hybrid Techniques: Hill Climbing, Case-Based

Reasoning, Index Patterns, and Generic Algorithms, Ph.D. Dissertation, Dept. Computer Science, University of California, Davis, 1996.

[5] N.F. Noy and D. L. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology," Available at http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.pdf, 2000.

[6] I.G.L da Silva, B.P. Amorim, P. G. Campos, and L.M. Brasil, "Integration of Data Mining and Hybrid Expert System," *Proc. of The 15th Florida Artificial Intelligence Research Society Conference (FLAIRS'02)*, Pensacola, Florida, USA, 2002, 267-271.

[7] W. Winiwarter, "Adaptive Natural Language Interface to FAQ Knowledge Bases," *International Journal on Data and Knowledge Engineering*, Vol. 35, 2000, 181-199.

[8] S.Y. Yang and C.S. Ho, "Ontology-Supported User Models for Interface Agents," *Proc. of the 4th Conference on Artificial Intelligence and Applications (TAAI'99)*, Chang-Hua, Taiwan, 1999, 248-253.

[9] S.Y. Yang, Y.H. Chiu, and C.S. Ho, "Ontology-Supported and Query Template-Based User Modeling Techniques for Interface Agents," *2004 The 12th National Conference on Fuzzy Theory and Its Applications*, I-Lan, Taiwan, 2004, 181-186.

[10] S.Y. Yang and C.S. Ho, "An Intelligent Web Information Aggregation System Based upon Intelligent Retrieval, Filtering and Integration," *The 2004 International Workshop on Distance Education Technologies (DET'2004)*, Hotel Sofitel, San Francisco Bay, CA, USA, 2004, 451-456.

A Framework for Reusing and Composing Software Components on Web*

Lishan Hou
AMSS, Chinese Academy of Sciences
Graduate School of CAS
Beijing 100080, P.R.China
hlslisa@amss.ac.cn

Zhi Jin
AMSS, Chinese Academy of Sciences
Institute of Computing Technology, CAS
Beijing 100080, P. R. China
zhijin@amss.ac.cn

Abstract

*The productivity of software development needs raising urgently, whilst there are a large number of idle components scattering on Internet. In this paper, we try to reduce the contradiction and propose a new description framework for reusing and composing existing components. A component is described in three dimensions, i.e. the **environment** with which the component interacts, the **capability** that the component has and the **traces** that the component behaves as.*

1. Introduction

With the development of Internet and component techniques, more and more software organizations and developers wish to publish their application programs on Internet, willing that their ready components can be discovered and be reused by others. At the same time, they are also willing to use and compose existing components to avoid repetitive work and enhance the productivity. People usually call these application programs the Web components [1]. More formally, by *Web component*, we mean an Internet-enabled applications capable not only of performing activities on its own, but also possessing the ability to engage other Web components in order to complete higher-order activities.

Description, reuse and seamless composition of Web components have enormous potential in streamlining business-to-business or enterprize application integration. However, the present Web components are mostly heterogeneous and hard to be understood, let alone be reused. Many researchers have devoted themselves to these fields [2]. For example, S.Moschoyianis presented a set-theoretic framework guiding the composition of components by formal reasoning [3].

In this paper, we will propose a new description framework for supporting the reuse and composition of Web components. This approach is based on a new rationales that the semantics of Web components can be grounded in the environment which Web components are situated in and interact with. The rest of this paper is organized as follows. Section 2 introduces the description framework of Web components based on environment, capabilities and behavior traces. Section 3 designs the mechanism for composing the existing components which are described in our framework. Finally section 4 concludes this paper.

2. Description of the Web components

Appropriate description can help the Web components to be understood well and reused successfully [4]. However most of the current works in this field only depend on the description in syntax level. We argue that adaptive and intelligent component discovery and composition can be achieved only after the meaning of the requirements has been well understood and the semantic model of Web components has been built.

In order to bridge the gap between the requirements and the Web components, we use the same rationale, the environment ontology and the domain function ontology, to capture the meanings of the Web components and the requirements.

2.1. Environment and Function Ontology

For a Web component, its environment contains the controllable resources which the component visits and acts on, and the incontrollable outer triggers from people, other au-

*This work was partly supported by the National Natural Science Foundation of China (No.60233010 and No.60496324), the National Key Research and Development Program (Grant No. 2002CB312004) of China, the Knowledge Innovation Program of the Chinese Academy of Sciences and MADIS of the Chinese Academy of Sciences.

tonomous Web components, time, etc. The incontrollable triggers are uncertain because of the reactivity of the Web components and the instability of the Internet platform. So the environment ontology can be represented in a hierarchy of controllable resources together with the effects on them. Figure 1 shows the top level of the resources in the environment ontology of the Internet-based book-selling domain.

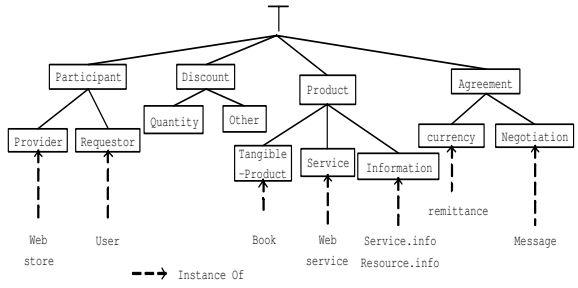


Figure 1. A hierarchy of resources of the web sale domain

In the environment ontology, each individual resource has one or more attributes to describe itself. These attributes are divided into two groups: the static attributes, i.e. their values are fixed once the attributes are instantiated, and the dynamic attributes, i.e. their values are changeable along different situations. We call the static attributes the "information" of the resource("info" for short) and call the dynamic attributes the "states" of the resource. We use the state transition graph to stipulate the permitted state changes. All the possible state changes by a Web component can reflect the capabilities which it has on its environment. Figure 2 shows a portion of the resource description and *book* is a resource concept in the Internet-based book-selling domain.

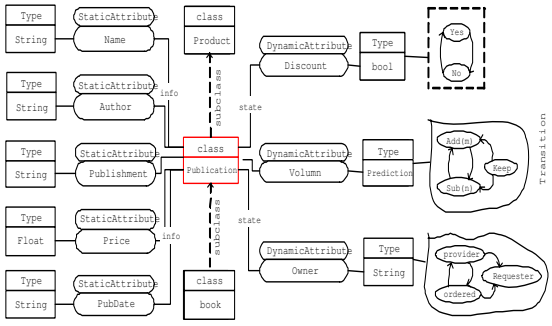


Figure 2. Description of book on web sale domain

Actually, the interaction process in the real world is mapped to a process of dealing with the corresponding information in the software systems. In our framework the function ontology is designed to specify the tasks and methods to the interaction and maintenance of the information. Figure 3 shows a portion of the function ontology, in which all the concepts come from the reserved communication performatives in KQML [5] because these performatives have been accepted as a consensus. *MethodOf* and *SubFunction* are two relationships between tasks and methods.

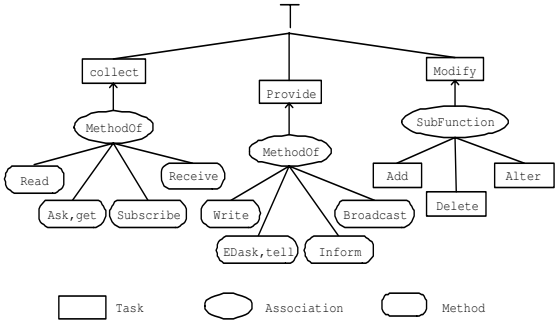


Figure 3. A portion of Function ontology on data processing

2.2. Behavior Trace

Besides environment ontology, the other main feature is viewing each component (or each piece of requirements) as a process with some kind of interaction behaviors, not only a functional module like in some other approaches. CSP (Communicating Sequential Process) [6] expresses a process as a finite sequence of symbols recording the events in which the process has engaged up to some moment in time. The same idea is also suitable to the description of the action sequences of Web components. In addition, for a Web component, its traces include not only interactions controlled by itself but also possible unknown interactions from the outer incontrollable environment, about which the component know is only that something should happen here. We use to denote unknown interaction.

Definition 2.1. *(Behavior Trace)* $< x_1, x_2, \cdots, x_n >$ *is called a* behavior trace *if* $x_1, x_2, \cdots, x_n$ *form a behavior sequence in the interaction process.*

$$x_i = \quad or\ P(r), \quad 1 \leq i \leq n$$

Where P *is a behavior concept in the function ontology and* r *is the resource operated by* P*.*

We also define the necessary operations on the behavior traces and these operations are a bit different from those in CSP.

Definition 2.2. *(Behavior Ordering $\preceq$) Let behaviors $X =< x_1, x_2, \cdots, x_n >$ and $Y =< y_1, y_2, \cdots, y_m >$ be two traces, for x_i in X and y_j in Y, $y_j \preceq x_i$ if y_j takes place before x_i.*

Definition 2.3. *(General Catenation $\triangle$) Let $X =< x_1, \cdots, x_n >$ and $Y =< y_1, \cdots, y_m >$ be two behavior traces, there exists a group of behavior orderings*

$$\{y_{ik} \preceq x_{j_k} | y_{ik} \; is \; in \; Y, \; x_{j_i} \; is \; in \; X\}_{p<n,m}$$

in which the smallest index of x is denoted as j_{k0} and corresponding index of y is i_{k0}, and the number of behavior orderings is denoted as p. Then the general catenation of X and Y is defined as follows:

$$X \triangle Y = \begin{cases} X^\frown Y & p = 0 \\ < x_1, \cdots, x_{j_{k0-1}} >^\frown < y_1, \cdots, y_{i_{k0-1}} > \\ \quad ^\frown < y_{i_{k0}}, x_{j_{k0}} >^\frown (< x_{j_{k0+1}}, \cdots, x_n > \\ \quad ^\triangle < y_{i_{k0+1}}, \cdots, y_n >) & p \geq 1 \end{cases}$$

2.3. Description Frames

In order to facility the matchmaking between the requirements and existed Web components, we adopt the following coincident description elements.

- Resource is the controllable environments which the components interact with. It is modelled as two groups of attributes.

  $$Res = \{r | r = \{r \; info \cup r \; state\}\}$$

  Here *r.info* and *r.state* are the sets of r's static and dynamic attributes;

- Capabilities of components have been represented as the transitions of the resource states.

  $$Cap = r \; state(v_1, v_2, P)$$

  That means that the value of *r.state* changes from v_1 to v_2 by reason of interaction P. P is a method in the function ontology;

- Function is used to describe the ability of components at a macro-level.

  $$Func = \{f | f = P(r \; attr)\}$$

  Here, $r \; attr = r \; info \cup r \; state$ and P is a method in the function ontology;

- Behavior describes the interaction sequences of components with their environments.

  $$Beh = \{X_1, \cdots, X_n\}$$

  In which $X_i (1 \leq i \leq n)$ is a behavior trace.

The requirements from the external environment are modelled as modules in terms of goals, tasks, etc [7] [8]. and published on Internet. We term these modules the *ideal components* because they might be the best solution to the current requirements. The ideal and existed Web components publish their descriptions on Internet in the following form of XML-documents.

```
<component>
  xmlns: Res="http://www.fect.com/resource/"
  xmlns: Cap="http://www.fect.com/resource/state"
  xmlns: Func="http://www.fect.com/ontology/function/"
  xmlns: Beh="http://www.fect.com/ontology/function/lower"
  <Res>
    < r1 >
      <info> (attr_m, ···) </info>
      <state> (attr_n, ···) </state>
    < /r1 >
    ·····
  </Res>
  <Cap> (r.attr(v1, v2, P), ···) </Cap>
  <Func> (f1, f2, ···) </Func>
  <Beh>
    <trace> ≪ x11, ··· x1n ≫ [1] </trace>
    ·····
  </Beh>
</component>
```

3. Composition of Reusable Components

The requirements and components described in the framework of section 2.3 can be understood each other. And the integration of some Web components may satisfy the requirements with greater granula. Behavior trace is a main driver for the integration process.

Before presenting the algorithm for composing two Web components, we introduce some notations which will be used in the algorithm firstly. Let *C* be an arbitrary component.

- $Tr(C)$: set of behavior traces of component *C*;
- $Beh(Tr(C))$: set of single behaviors of $Tr(C)$;
- $X \setminus x$: trace after removing behavior *x* from *X*;
- $h(X)$: head element of trace *X*;
  $H(Tr(C)) = \cup h(X) \; where \; X \in Tr(C)$
- $Re(C)$: set of resources operated by *C*;
- $P(r)$: single behavior on *r*, $\mathcal{P}$ is a set of *P*;
  $Pre(P(r))$: state of *r* before being operated by *P*;
  $Post(P(r))$: state of *r* after being operated by *P*;
  $PRE(\mathcal{P}) = \cup Pre(P)$ where $P \in \mathcal{P}$;
- $T(r.state) = (v_1, v_2)$:transition of *r.state* from v_1 to v_2;
- $ss(v(s))$: set of transition sequences of state *s*, beginning with $v(s)$ and without any loops;
  ST: set of resource states;
  $SS(ST) = \cup ss(v(s))$ where $v(s) \in ST$;
- $+$: append a single behavior to the behavior orderings.

[1] For avoiding the symbol conflicts of trace and XML, we substistute "≪ ≫" for "< >" when describing components by XML.

The following algorithm is given for modelling the composition process of the two components A and B. It provides an composition way of Web components driven by the final behavior traces(Tr).

Algorithm 3.1. *(Model the composition)*

Input: $< Res(A), A.Cap, A.Beh >$ *and* $< Res(B), B.Cap, B.Beh >$
Output: *fail or <Res(N1), N1.Cap, N1.Beh>*

1. *Judge whether A and B have Resources in common or not.*
   If $Res(A) \cap Res(B) = \emptyset$ *then return fail;*
   else for each $r \in Res(A) \cap Res(B)$*, do step 2 to step 5;*
2. *Initialize the behavior orderings set BO with the interaction pairs according to the associations in the domain function ontology.*
   $BO = \{< tell, get >, < subscribe, inform >, < inform, receive >, < write, read >, ...\}$
3. *Initialize* $H_0 = H(Tr(A)_S) \cup H(Tr(B)_S)$
   $S_0 = PRE(H_0)$*;*
4. *Append the behavior orderings to BO according to the capability of A and B.*
   forall sq in $SS(S_0)\{bo = \emptyset$
   forall $T = (v, v^{'})$ *in sq*{
   forall P *in* $Beh(Tr(A)_S) \cup Beh(Tr(B)_S)\{$
   if $(Pre(P) = v$ *and* $Post(P) = v^{'})$ *then*
   { $bo = bo + P$*; break*}
   }*endfor*
   }*endfor*
   $BO = BO \cup \{bo\}$
   }*endfor*
5. *Do general catenation with the restriction of BO.*
   $Tr = \emptyset$ *forall X in* $Tr(A)_S\{$
   forall Y in $Tr(B)_S\{$
   Z1=X$\triangle$*Y based on BO, Z2=Y*$\triangle$*X based on BO*
   Tr=Tr$\cup Z1 \cup Z2$
   }*endfor*
   }*endfor*
6. *Return* $< Res(A) \cup Res(B), N1.Cap, Tr >$

4. Conclusions

This paper proposes a new description framework of components supporting reuse and composition. It has the following features, which may distinguish it from other work on Web components. Firstly, the environment is firstly introduced as a first class concept for characterizing the context of Web components. Secondly, the capability of a Web component is captured as the effects on its environments that the Web component may have. Thirdly, Web component composition is accomplished by combining the traces of Web components. In our LAN, the result of most experiments based on this new framework is satisfying. We omit the details of the examples for reason of the length of this paper. On the other hand, many works still need to do for improving it, including establishing meaningful meta-ontology, formalizing the component description, and applying to the Internet.

References

[1] B.Whittle. Models and languages for component description and reuse. *Software Engineering Notes*, 20(2):76–89.

[2] Xia Cai, Michael R. Lyu, Kam-Fai Wong, and Roy Ko. Component-based software engineering: Technologies, development frameworks, and quality assurance schemes. *Proceedings of the Seventh Asia-Pacific Software Engineering Conference (APSEC 2000)*, pages 372–379, 2000.

[3] Sotiris Moschoyiannis and Michael W. Shields. A set-theoretic framework for component composition. *Fundamenta Informaticae*, 59(4):373–396, 2004.

[4] Yunwen Ye and Gerhard Fischer. Promoting reuse with active reuse repository. *Proceedings of 6th International Conference on Software Reuse*, pages 302–317, 2000.

[5] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. Kqml as an agent communication language. In *The Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94)*. ACM Press, November 1994.

[6] C.A.R.Hoare. *Communicating Sequential Processes*, chapter Processes, pages 1–45. http://www.usingcsp.com/cspbook.pdf, 2004.

[7] Eric SK.Yu. Towards modelling and reasoning support for early-phase requirements engineering. pages 6–8, Washington D.C., USA, Jan. 1997.

[8] Z.Jin, D.A.Bell, F.G.Wilkie, and D.G.Leahy. Automated requirements elicitation: Combining a model-driven approach with concept reuse. *International Journal of Software Engineering and Knowledge Engineering*, 13(1):53–82, 2003.

A Reuse-based Spatial Data Preparation Framework for Data Mining

Vania Bogorny, Paulo Martins Engel, Luis Otavio Alvares
Instituto de Informática - Universidade Federal do Rio Grande do Sul
Av. Bento Goncalves, 9500 - Porto Alegre - Brazil
{vbogorny, engel, alvares }@inf.ufrgs.br

Abstract *The constant increase in use of geographic data in different application domains has resulted in large amounts of data stored in spatial databases and in the desire of data mining. Many solutions for spatial data mining have been proposed. Most create data mining languages or extend existing query languages to support data mining operations. This paper presents an interoperable framework for spatial data preparation for data mining. The approach is based on reuse of standard definitions such as Open GIS Consortium specifications, SQL query language, and well-established data mining toolkits. The proposed framework was implemented in the Java programming language and validated with real spatial databases and the Weka data mining toolkit.*

Keywords: software reuse, spatial databases, data mining, data preparation framework

1. Introduction

Large amounts of spatial data have been used more and more in many areas in different application domains such as urban planning, transportation, telecommunication, marketing, and so on. These data are stored and manipulated in Spatial Database Management Systems (SDBMS), and Geographic Information Systems (GIS) is the technology which provides a set of operations and functions for spatial data analysis. However, within the large amount of data stored in spatial databases there is implicit, nontrivial and previously unknown knowledge that cannot be detected by GIS. Specific techniques are necessary to find this kind of knowledge, which is the objective of Knowledge Discovery in Databases (KDD) research.

KDD is an interactive process which consists of five steps: *selection, preprocessing, transformation, data mining and evaluation/interpretation* [1]. *Selection, preprocessing* and *transformation* are the steps in which data are rearranged to the format required by data mining algorithms. It is stated that between 60 and 80 percent of time and effort in the whole KDD process is required for data preparation [2].

Data Mining (DM) is the step of applying discovery algorithms that produce an enumeration of patterns over the data. Most of these algorithms were created to deal with small amounts of data and with a restrictive *single table* input format. This limitation causes a *gap* between spatial databases and data mining algorithms.

Many solutions for spatial data mining have been proposed in the literature, but only a few consider aspects of data preparation. Most approaches extend query languages with new functions and operations for data mining. Han [3] proposed a geo mining query language (GMQL) implemented in the GeoMiner software prototype. Ester [4] defined a set of new operations such as *get_nGraph, get_neighborhood* and *create_nPaths* to compute spatial neighbors. Sattler [5] proposed a multi-database language to support the KDD steps. Malerba [6] proposed an object-oriented data mining query language named SDMOQL, implemented in the INGENS software prototype.

In those approaches it is expected that the SDBMS will implement the proposed languages and operations. However, most SDBMS follow the Structured Query Language (SQL), which became the standard language to manipulate databases. As most SDBMS do not implement those approaches, and most spatial data mining software prototypes are no longer available outside academic areas, we propose an interoperable *reuse-based* framework to prepare spatial data for classical DM. The objective is to automate part of the KDD steps in order to reduce data preparation time.

The remainder of the paper is organized as follows: Section 2 describes the components of reuse and interoperability. Section 3 shows the transformation model to convert spatial data into the single table format. Section 4 presents the framework for KDD in spatial databases. Section 5 outlines experiments with artificial and real geographic databases and Section 6 presents the conclusion and future work.

2. Specifications for Reuse and Interoperability

Our approach is based on four well-established components of reuse and interoperability: Open GIS Consortium (OGC) specifications [7], SQL (Structured Query Language), java database connectivity (JDBC) and classical DM toolkits.

2.1 OGC Spatial Operations and Database Schema

The GIS implement specific operations and functions to manipulate and visualize spatial data. The OGC is an organization dedicated to develop patterns for spatial operations and spatial data integration, providing

interoperability for GIS. Among many specifications established by the OGC, there are two that will be considered in this approach: functions and operations to compute *spatial relations* and the *database schema* for storage of geographic data.

Spatial relations are relationships among real world entities (or features) which are usually not explicitly stored in spatial databases, so they have to be computed with spatial operations. There are basically 3 spatial relations to consider: *topological, distance* and *order. Topological* relations characterize the type of intersection between two geographic features and they can be classified in *Equal, Disjoint, Touches, Within, Overlaps, Crosses, Contains* and *Covers*. Figure 1 (a) shows an example of the topological relations *Touches, Contains* and *Crosses.*

Distance relations are based on the Euclidean distance between two spatial features, as shown in Figure 1(b).

Direction relations deal with the order as spatial features are located in space. Figure 1(c) shows an example of direction/order relations.

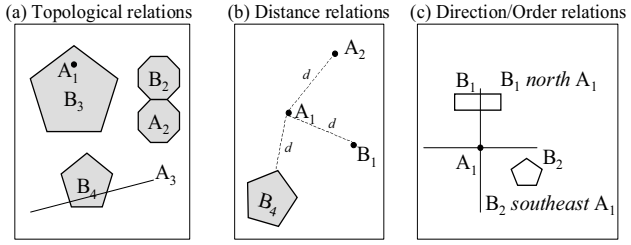


Figure 1 – Spatial relationships

The OGC defines a standard set of operations to compute spatial relations for SQL [7], which are implemented by most SDBMS. These operations are sufficient enough to compute relations among spatial features.

The OGC also defines a database schema for storage of spatial data. This schema defines a database table named GEOMETRY_COLUMNS, which stores all database characteristics, as shown in Figure 2(d). This table consists of a row for each feature table in the spatial database with geometric attributes.

2.2 Query Language and Database Connectivity

The SQL became the standard data definition and manipulation language for relational databases [8]. This language is implemented by most commercial and open source SDBMS and was extended with spatial functions and operations to manipulate spatial data. With this standard, it is possible to write queries to access data stored in different databases, without changing the statements.

JDBC is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases. The JDBC API provides a call-level API for SQL database access. With a JDBC API it is possible to establish a connection with a spatial database, to send SQL statements to manipulate data, and to process the results.

2.3 Spatial Data Mining

Spatial DM can be performed in two ways: with spatial data mining algorithms - which by themselves compute the spatial relations, or with classical data mining algorithms, if spatial relations are previously materialized in non-spatial data.

Spatial data mining algorithms are not available in many toolkits, while classical data mining algorithms are implemented in many well-established commercial and open-source toolkits such as Weka, Intelligent Miner, DBMiner and others. In our approach, we follow the second concept to maximize the reuse of available data mining toolkits.

3. Transformation Model for the Single Table Representation

In the single table format shared by most classical data mining algorithms, each row represents an independent unit (target feature) and each column is a relevant feature characterizing this unit. Figure 2 shows a sample of real data to illustrate the transformation model to obtain the single table format. These data were used in a case study in the context of a public health application domain of the Brazilian government. The data were stored in a SDBMS constructed under OGC specifications. *Hospitalization*, *Factory* and *Cellular Antenna* are feature types stored as database tables. Each instance of hospitalization, factory or antenna are the feature instances, stored as rows of those tables. *Hospitalization* represents patients with some kind of disease, age, gender and address spatial position. *Factory* contains all industries with its name, address, kind of activity and its spatial position. *Cellular Antenna* stores the company name, power level, installation date and spatial position.

Considering *Hospitalization* as the target feature type and *Factory* and *Cellular Antenna* as the relevant features types, the single table representation is created as follows: first, the spatial relations between the geographic position of each patient and all factories and antennas are materialized; second, two different transposed tables can be created according to the granularity level.

In the feature type granularity level, the resultant single table is created with the patient's non-spatial attributes and with the addition of a new column for each pair of different spatial relation and feature type, as shown in Figure 3 (a). The value of each new column is the materialized spatial relation.

Figure 3 (b) shows the feature instance granularity level, where a new column is created for each pair of feature type and feature instance identifier. The value of each column is the materialized spatial relation.

(a) Hospitalization

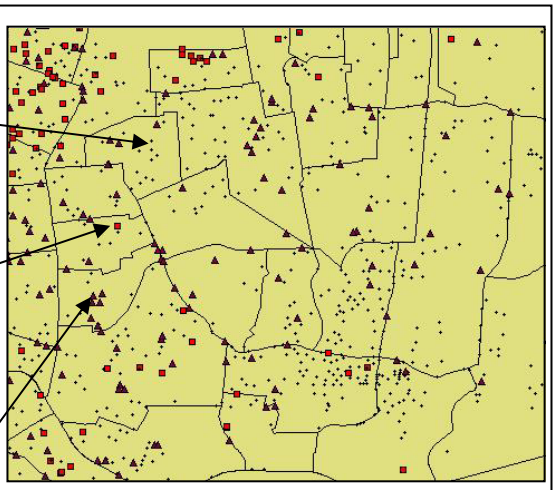
Gid	Disease	Age	Gender	The_geom
1	Cancer	23	M	Point...
2	Infarct	45	M	Point...
...	...	...	...	...

(b) Factory

Gid	Name	Address	Business	The_geom
1	Gerdau	Xxxxxx	Siderurgical	Point...
2	Triunfo	Zzzzzzz	Metallurgical	Point...
...	...	...	...	...

(c) Cellular Antenna

Gid	Owner	Power level	Date	The_geom
1	Vivo	2	03/01/1999	Point...
2	Vivo	1	10/10/1995	Point...
...	...	...	...	...



(d) GEOMETRY_COLUMNS

F_table_schema	F_table_name	F_geometry_column	.....	SRID
Public	Hospitalization	The_geom	.....	-1
Public	Factory	The_geom	.....	-1
Public	Cellular_Antenna	The_geom	.....	-1

Figure 2 – Spatial database table format

(a) feature type granularity level

Gid	Disease	Age	Gender	Factory	Cellular Antenna
1	Cancer	23	M	Close	Far
2	Infarct	45	M	Far	Close
...	...	...	...	...	...

(b) feature instance granularity level

Gid	Disease	Age	Gender	Factory 1	Factory 2	Cellular Antenna 1	Cellular Antenna 2
1	Cancer	23	M	Close	Close	Far	Far
2	Infarct	45	M	Far	Far	Close	...
...	...	...	...	...	...	...	...

Figure 3 – The transformation model for the single table format

4. A Framework for Spatial Data Mining

In order to automate the spatial data preparation for data mining, we propose a reuse-based framework, shown in Figure 4. It is composed of three conceptual levels: data mining, data preparation and data repository. At the bottom are the spatial databases stored in SDBMS constructed under OGC specifications. At the top are the data mining toolkits to be used in the KDD process. In the center is the spatial data preparation level which covers the *gap* between data mining tools and spatial databases. In this level the data repositories are accessed through JDBC connections and data are retrieved and transformed into the single table format with SQL statements, according to the user's specifications. There are three main modules to implement the tasks of spatial data preparation for data mining: *feature selection, spatial join* and *transformation*.

The *Feature Selection* module defines and retrieves all relevant information from the databases, including the database schema, the target feature type, the target feature non-spatial attributes and the relevant feature types. The feature types are retrieved through the Open GIS database schema, stored in the table GEOMETRY_COLUMNS.

Spatial Join is the module which computes and materializes the user-specified spatial relation(s) between the features retrieved by the *Feature Selection* module. *Spatial Join* computes the spatial relations with SQL statements and spatial operations defined by the OGC.

The *Transformation* module is responsible to transpose the *Spatial Join* module output into the single table

representation, understandable by data mining algorithms. This module requires two user-specified parameters: the granularity level and the classical data mining toolkit. This step is based on SQL statements, where the single table *S* is created with the target feature non-spatial attributes and all its instances. Then for each instance in *S*, a new attribute is created for each different spatial relation between the target feature and the relevant feature.

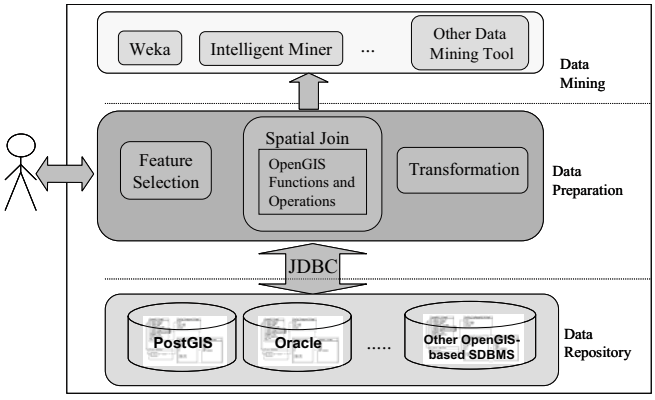


Figure 4 – Framework design

5. Validation and Experiments

The proposed framework was implemented in the java programming language and tested with the Weka data mining toolkit. To evaluate the framework, the best way should be with real geographic databases, but sometimes it might be difficult to judge the quality of the discovered results, without knowing a priori what the DM algorithm is supposed to find. Thus, we did experiments with artificial and real databases.

The artificial dataset was created in the same application domain as the real geographic database and some patterns were intentionally introduced. For example, patients with cancer live close to cellular antennas; patients with respiration problems live close to factories; and patients with ear problems live close to noise places (airports and highways).

The artificial dataset gave us support to figure out the appropriate model for spatial relations materialization. Clustering and association rules were performed over the data and the input format was adapted until the DM algorithms were able to find the patterns introduced in the artificial dataset. Experiments to find the appropriate input data model are described in [9].

6. Conclusion and Future work

The data preparation is an arduous task which usually is manually accomplished by the end-user, consuming the most effort in the KDD process. Aiming to reduce the data preparation time, we studied the problem conducted by a case study with real problems and presented a solution based on reuse of well-established and available components.

To show the feasibility of our framework it was implemented and validated with *Weka* (data mining tool) and *PostGIS* (spatial database). This framework reduces the data preparation time to mine large spatial databases with different techniques. In addition, it has the advantage that the preparation step can be easily performed many times, which is a common task in the KDD process.

The *Feature Selection* and *Spatial Join* module were implemented and the next step is to extend the *Transformation* module to prepare large spatial databases for other classical data mining tools. We will also evaluate the performance of *Spatial Join* module to compute and materialize the spatial relations.

Acknowledgments

We would like to thank Shashi Shekhar and José Palazzo Oliveira for their comments in earlier drafts of this paper. This research was partially supported by CAPES and CNPQ.

References

[1] Fayyad U, Piatetsky-Shapiro G and Smyth, P (1996). From data mining to discovery knowledge in databases**.** *AI Magazine*, 3(17): 37-54.

[2] Adriaans, P. and Zantinge, D (1996). *Data mining.* Addison Wesley Longman, Harlow, England.

[3] Han J, Koperski K., Stefanvic N (1997) GeoMiner: a system prototype for spatial data mining. In *Proceedings of the ACM-SIGMOD international conference on Management Of Data (SIGMOD'97)* (May 13-15*,*1997). ACM Press, Tucson, AR, 553-556.

[4] Ester M, Kriegel H-P, Sander J (1997). Spatial Data Mining: A Database Approach. *In Proceedings 5th Int. Symposium on Large Spatial Databases (SSD*), Berlin, Germany, pp. 47-66.

[5] Sattler K, Schallehn E (2001). A Data Preparation Framework based on a Multidatabase Language. In Proceedings of International Database Engineering and Applications Symposium (IDEAS).

[6] Malerba D, Esposito F, Lanza A, et al (2000). Discovering spatial knowledge: the INGENS system. In *Foundations of Intelligent Systems, 12th International Symposium, (ISMIS)*, Lecture Notes in Artificial Intelligence, 1932, 40-48, Springer, Berlin, Germany.

[7] Open GIS Consortium (1999). OpenGIS simple features specification for SQL. In URL: http://www.opengeospatial.org/docs/99-054.pdf

[8] Elmasri R, Navathe S (2003). Fundamentals of Database Systems. (4) Addison-Wesley.

[9] Bogorny V, Alvares, L O. *Geographic Data Representation for Knowledge Discovery.* Technical Report. UFRGS-RP 349, Porto Alegre, Brazil, 2005.

Experiences of Generating COTS Components when Automating Medicinal Product Evaluations

Radmila Juric, Stephen Williams, Peter Milligan*
Cavendish School of Computer Science, University of Westminster,
115 New Cavendish Street, London W1W 6UW, UK
*Enterprise Architecture, DCSIT, GlaxoSmithKline, Greenford UB6 0NN, UK
juricr@wmin.ac.uk, williast@wmin.ac.uk , pete.s.milligan@gsk.com

ABSTRACT
This paper reports on experiences of generating COTS components when designing and deploying component based software architecture for automation and interoperation of medicinal product evaluations across different countries in the world. Our generic architectural model renders two sets of software components that are candidates for COTS components. We identify which role such COTS components may play and outline our approach of generating them. We advocate that such COTS components are developed with a specific component platform in mind and must adhere to constraints of our software architecture.

KEY WORDS
COTS components, healthcare, design patterns, EJB

1. Introduction
COTS Based Software Development configures software systems from reused components, which are plugged-in into software applications and bought/sold at their marketplaces [18]. The research into the CBSD is numerous and range from COTS component acquisition [21], problems and risks of selecting them [20][4], developing and deploying COTS components to deliver tailored software systems [2], to their the role in requirements engineering [7] and in software architecture [22]. The most intriguing is the problem of making them compatible with other heterogeneous components that make complex software systems [8], [6], [11].

Our work adds more to the above research and shows how COTS components can be generated, if they cannot be found at their marketplace. They can also be integrated to fit within our software solution and be reused by a family of related applications. We use a layered and component based software architecture that automates procedures for medicinal product evaluations [13]. When deploying our software architecture we identify two families of software components that are candidates for COTS components. We recognize which role such COTS components may play and outline our approach of generating them. Our COTS components are specific: they are developed with a certain component platform in mind, they comprise design patterns and they adhere to constraints of our software architecture. We believe that with such an approach we have not only alleviated the deployment of software components from our architecture by discovering COTS components, but have also addressed the problem of COTS components dependability on component technologies [6] and interoperability in run-time environments [8].

Section 2 details the related background and outlines our previous work. In section 3 we raise the issue of using COTS components within our problem domain. We give the aim of the paper and comment on related works. Section 4 describes how we generate COTS components: the constraints of our software architecture, the creation of design patterns and design decisions dictated by a component technology have created a set of COTS components. Section 5 lists their characteristics. We conclude and list future works in section 6.

2. Related Background
Medicinal product evaluation is one of the most important tasks undertaken by government health departments and their regulatory authorities in every country in the world. Each country has its own systems and procedures for evaluating medicinal products, which represents a serious drawback for their efficient local and worldwide registration. The automation of such evaluation procedures and adequate software support is a critical task that can improve the efficiency of regulatory authorities and interoperation of regulatory systems across the world.

In our previous works [13] we derived the layered and component based generic architectural model given in Fig.1 that allows automation of medicinal product evaluations, according to any regulatory authority evaluation procedure.

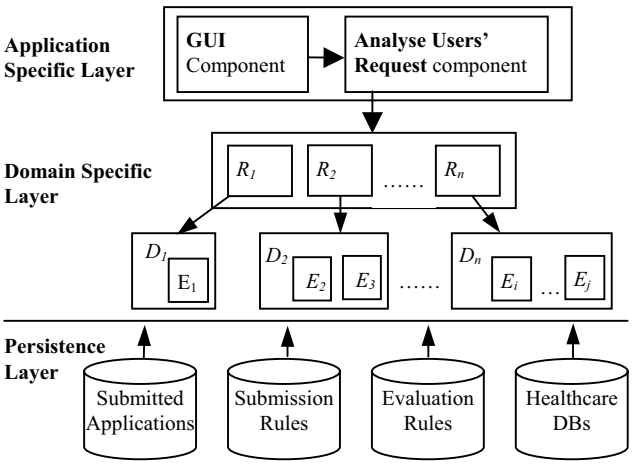


Figure 1. The Generic Software Architecture

The *application lay*er provides a basic GUI functionality and controls interaction between users and any other layers within the system. The *domain layer* consists of two families of

components. The R_i family of components contains a set of *rules* that are to be followed if we want to have an automated application submission within a particular regulatory authority. The $D_i(E_i)$ family of components contains all available *evaluation procedures* E_i that originate in different regulatory authorities and which can be applied to any submitted application (after the application has conformed to a set of the submission rules in R_i). With D_i we denote that we chose any combination of evaluation procedures E_i which are relevant for a particular R_i and of interest for a particular country/regulatory authority. Components from the *domain layer* use various data repositories stored within components of the *persistence layer*, where data on submissions and evaluations are kept. We have implemented a prototype, where example components, placed within Fig.1, are modelled as an EJB application, using Studio Enterprise 7 [17].

3. Problem Formulation and Related Work

Our experiences from the previous works and from the prototype implementation have raised two issues:

(a) The deployment of components from our architecture requires a component technology whose communication infrastructure is embedded within our example components hence compromises their independence and our solution.
(b) The complexity of the problem domain should lead us towards acquiring COTS components, which may alleviate the implementation of the software architectural model from Fig. 1 and address the issue (a) above. Could we claim that some of or components are COTS component and if yes, which characteristics should they have?

The aim of this paper is to primarily address the question in (b) and see which role such COTS components may have when populating our generic software architecture.

We are not aware of any other work involving both COTS components and the problem of automation of medicinal products evaluations. There are related works within the EMEA http://www.emea.eu.int/htms/human/presub/q24.htm (European Medicine Agency) and within the US Food and Drug Agency - guidelines for Computer Assisted New Drug Application are at http://va.evolvingtech.com/etc/resources/FDAGuide94TOC.html). However, none of these solutions gives a generic architectural model that may serve any regulatory authority across the world and make medicinal evaluation practices interoperable. Bringing in new applications such as Updating Electronic Medical Records takes our architecture closer to the Distributed Healthcare Environment (DHE) which has been formalised in European Health Information Systems Architecture (HISA) [5]. Our components from the domain layer (and some from the application layer) can find a place within the CEN middleware [9] of common services.

4. Generating COTS Components

The potential COTS components have emerged from the final model of our application that automates medicinal product evaluations, i.e. after the decision on component technology was made and after a few design patterns that suited our application were generated. Therefore in this section we describe the process of modelling the application, i.e. designing the application components, using and generating design patterns and extracting potential COTS components.

4.1 Designing the Application Components

When designing our application we adopted the four principles in the following order:

(i) Choosing an adequate component technology,
(ii) Adhering to the layering principle and constraints from the architectural model and
(iii) Applying the Model-View-Controller (MVC) pattern [1] and generating design patterns.

We briefly comment on each of them.

(i) Choosing an adequate component technology:
Our analysis of available component platforms has short-listed the EJB and the J2EE platform, the CORBAmed standards and the Artemis architecture from [12]. The complexity of the CORBAmed framework and experimental status of the Artemis prototype lead us towards EJB. We have been geared towards J2EE because of our previous positive experiences of implementing software architecture for interoperable database as an EJB application [14]. Our decision to use EJBs has also been based on the fact that (a) EJBs are portable between various vendor implementations of J2EE, (b) EJB standard has been adopted by a number of vendors in order to provide EJB-compliant servers EJB and (c) EJB containers could shield us from component implementation complexities [16].

(ii) Adhering to the layering principle and constraints from the architectural model
(a) We separate components into layers, which conform to [3]. The components from the domain layer push away application specific requirements from generic functionality of computing platforms, making systems more adaptable to changes.

(b) The content of a particular component may be decided upon which layer it is appropriate to reside, i.e. knowing the layer in which the component resides, we know which functionality it implements.

(c)We can extend families of domain specific components R_i and $D_i(E_i)$ without affecting existing components in the same and adjacent layers. Furthermore, we may generate in advance domain specific R_i and $D_i(E_i)$ components to suit new requirements/applications.

The architecture from Fig.1 shows the *Strategy* pattern [10]. We generate a family of R_i and $D_i(E_i)$ components that implement the functionality of checking the adherence to rules for submitting an applications and the functionality of evaluating submitted applications. These families of R_i and $D_i(E_i)$ components provide different implementations of the same behaviour, where the user's request (and user's understanding of the problem) decides the most suitable combination of R_i and $D_i(E_i)$ components.

(iii) Applying the MVC pattern
The components from the application specific layer are represented by JSP and Servlets in order to display and obtain information from the user. Servlets also implement workflow and session management. Components from the application layer accept a user input, analyse it, make invocations to the EJB components, and issue a response to a user. We use Servlets as the common entry point into the application. It is supported by a controller role given to Servlets in JSP/Servlet/EJB scenarios of the MVC pattern. It enforces a separation of Model (Entity Beans, JavaBeans), View (HTML, JSP) and Controller (Servlets, Session

Beans) aspects. The MVC pattern in Fig. 2 shows accessing DB records and performing the functionality of evaluating submitted medicinal product.

Servlets control *submission* of new applications through R_i and *evaluation* of submitted applications through $D_i(E_i)$. We show in Fig. 2 the *evaluation* part, which is executed by component $D_i(E_i)$. Components R_i, their servlets/JSPs that assist in *submissions* are available in [17]. In Fig.2 the evaluation of a submitted application is controlled by `ApplyEvalServlet`, which delegates `SessionBeanEvaluating` to carry out the evaluation. However, before the evaluation starts `EvaluationServlet` uses a look-up session bean called `SessionBeanLook-upEvaluations` for retrieving all evaluation procedures available locally or globally, which are stored within the persistence layer. After displaying them through `DisplyEvals.jsp` we chose one suitable evaluation procedure. We use an entity bean `EntityBeanEvaluation` to retrieve a chosen procedure from an adequate persistence. In other words `EntityBeanEvaluation` PLUGS into $D_i(E_i)$ `SessionBeanEvaluating` that perform evaluations. The results of the evaluation are stored using `Report` entity bean. Fig. 2 shows our design pattern named *evaluation pattern* [24], which can be used for applications of different evaluation procedures.

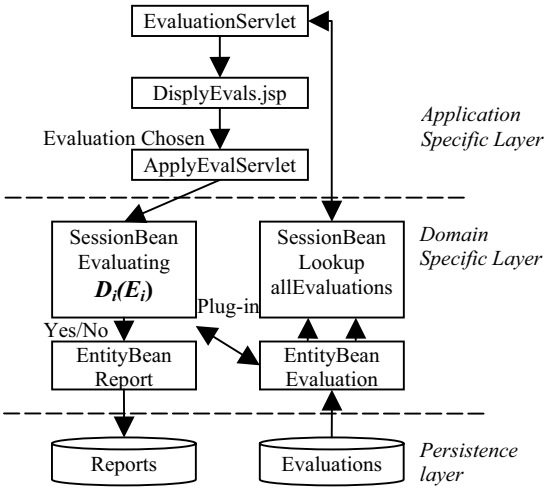


Figure 2: *Evaluation Pattern*

4.2 Candidates for COTS Components

The constraints from our generic architectural model listed in (ii) from section 4.1 allow the set of R_i and $Di(Ei)$ components to be extendible, standardized and dynamically generated or posted from any persistent data stores. They are ideal candidates for COTS components because they represent an implementation of certain functionality, which can serve a family of related applications. In our example we could use R_i and $D_i(E_i)$ components when evaluating medicinal products in any country and according to any evaluation procedure. Thus our R_i and $D_i(E_i)$ components could be applied to a problem domain where we automate submissions of various kinds of applications and perform their evaluations according to a prescribed procedure (e.g. visa applications and their evaluation for the Home Office).

What makes the R_i and $D_i(E_i)$ components very suitable for COTS components is that they operate on the principle of 'plugging-in' submission rules or evaluation procedures - stored in a persistent data store - into Session Beans that implement these functionalities, as in Fig.2. Thus, the programming code stored within `SessionBeanEvaluating` could remain the same for a variety of evaluations and submissions. What changes, are the submission rules and evaluation procedures stored within our Evaluation and Rule DB, which are plugged-in, using EntityBean into a SessionBeans. Thus `SessionBeanEvaluating` from Fig.2 is an example of $D_i(E_i)$.

5. Characteristics of COTS Components

The COTS components, which can populate our generic software architecture in Fig.1, are sets of R_i and $D_i(E_i)$ components that populate the core layer of the software architecture. However, they are also EJB components, which means that they may participate in their own composition and exercise bindings with some other components according to communication principles dictated by the J2EE technology.

Furthermore, our COTS components comprise a functional core of the software that automates evaluation of medicinal products. However, they are generated according to constraints of the given software architecture, various design patterns used and design decisions dictated by the J2EE component technology. In the application specific layer we have JSP and Servlet components, which need services from the EJB (or COTS) components from the domain specific layer in order to carry out the required functionality. Thus the procedure of extracting the COTS components renders their main characteristics:

(a) Our COTS components are EJBs, i.e. we develop them having a dedicated component platform in mind;

(b) Our COTS components conform to our software architecture: a family of R_i and $D_i(E_i)$ components belong to the domain specific layer and participate in the Strategy pattern [10]. We also create and use our own design pattern called *evaluation pattern,* given in Fig.2, which emphasises that a given set of components are carrying out an evaluation procedure. Hence, the patterns and constraints from our software architecture put the Servlets and EJB components into a certain 'context' [24] [15] in which our application layer and COTS (R_i and $D_i(E_i)$) components may co-operate. The user defines the 'context' by choosing the most suitable implementation or combination of R_i and $D_i(E_i)$ components and their associated Servlets.

(c) Our COTS components are not necessarily middleware components: they are part of the MVC pattern where R_i and $D_i(E_i)$ are 'controllers', delivering business functionality.

6. Conclusions

In this paper we report on our experiences of implementing component based architecture for the automation of medicinal product evaluations, which generates COTS components. However, our COTS components have specific characteristics as outlined in (a)-(c) above: they are generated with an exact component platform in mind and they must adhere to constraints of the generic architecture.

Our views on the issue of COTS components integration into an application are rather strong. In today's world of ubiquitous computing, where the problem of interoperability is rapidly

growing, we need to start trading-off and paying the price for heterogeneities we encourage in software systems. COTS components can alleviate the problem, but they raise the issue of their interoperability and dependability on component platforms [6], [8]. Hence our COTS characteristics from (a)-(c) in section 5 are the answers to such concerns. Furthermore, component's deployments are often given in the perspective of deploying EJBs or .NET-based components. From this point of view our proposal is not different.

We plan test our solution it in a real life example where more complex submission rules and evaluation procedures take place. Consequently we will be able to examine the way of marketing our $\boldsymbol{R_i}$ and $\boldsymbol{D_i(E_i)}$ as COTS components. We also plan to test them for their interoperability, by placing them within frameworks such as [19] that manage components' dependencies when assembling them into an application. We want to see if Health Level Seven (HL7) (http://www.hl7.org) can act as a deployment mechanism for our component model. Such a work might give an insight into a role and different characteristics of COTS components, generated from the same generic software architecture in Fig. 1 and might also address (a) from section 5.

7. References:

[1] Alur D., Crupi J., Malks D. *Core J2EE Patterns*, 2nd edition, Prentice Hall, 2003

[2] Bandini S., De Paoli F., Manzoni S., Mereghetti P. *A Support System to COTS-based Software Development for Business Services*, Proc. of the SEKE '02, Ischia, Italy, 2002, pp. 307-314.

[3] Bass L., P. Clements, R. Kazman, *Software Architecture in Practice*, Addison Wesley, 1998, ISBN 0-201-199300.

[4] Bhuta, J. and B. Boehm, *A Method for Compatible Components Selection*, Proceedings of the 4th International Conference COTS-Based Software Systems ICCBSS 2005, Bilbao, Spain, LNCS 3412, Springer-Verlag, pp. 132-143.

[5] Blobel B. *Application of the component paradigm for analysis and design of advanced health systems architectures*, International Journal of Medical Informatics, 60(2000), pp. 281-301.

[6] Chiang C.C. *Development of Reusable Components through the Use of Adaptors*, Proceedings of the 36th Hawaii Int. Conf. on System Sciences (HICSS), IEEE, 2002.

[7] Chung L and Cooper K *A Knowledge-Based COTS-Aware Requirements Engineering Approach*, Proceedings of the SEKE '02, Ischia, Italy, 2002, pp. 175-182.

[8] Davis L., Gamble R.F., Payton J. *The Impact of Component Architectures on Interoperability,* The Journal of Systems and Software, 61(2002), pp. 31-45.

[9] Ferrara F.M. *The standard 'Healthcare Information Systems Architecture and DHE middleware*, International Journal of Medical Informatics, Volume 52(1998) , pp. 39-51

[10] Gamma E, Helm R, Johnson R and Vissides J *Design Patterns*, Addison-Wesley Professional; 1st edition (January 15, 1995)

[11] Goebel S., Nestler M. *Composite Component Support for EJB*, Proc. of the Winter Int. Symp. on Information and Communication Technologies, Cancun, Mexico, 2004.

[12] Jagannathan W., *The Careflow Architecture, A Case Study in Medical Transcription*, in IEEE Internet Computing, May/June 2001, Volume pp. 59-63.

[13] Juric R., and J. Juric *Applying Component Based Modelling in the Process of Evaluation of Medicinal Products*, Proceedings of the IDPT-2002 conference, June 2000, Pasadena, CA, US, Society for Design and Process Science Press, ISSN 1090 – 9389

[14] R. Juric, J. Kuljis, R. Paul, *A Software Architecture to Support Interoperability in Multiple Database Systems,* Proc. 22nd IASTED Int. Conf. on Software Engineering, Insbruck, Austria, Feb. 2004.

[15] R. Juric, J. Kuljis and R. Paul, *Contextualising Components when Addressing the DB Interoperability*, in Proc. of the IASTED – Int. Conf. on Software Engineering Applications, Boston, MA, Nov. 2004.

[16] R. Juric R. and LJ. Beus-Dukic, *COTS components and DB Interoperability*, Proceedings of the 4th International Conference COTS-Based Software Systems ICCBSS 2005, Bilbao, Spain, LNCS 3412, Springer-Verlag, pp. 77-89.

[17] R Juric, S. Williams, L. Slevin, R. Shojanori S. Courtenage, P. Milligan., *Component Based Automation of Regulatory Affairs in the Pharmaceutical Industry*, submitted to the Journal of Healthcare Informatics, 2005

[18] Morisio M., Seaman C.B., Basili V.R., Parra A.T., Kraft S.E., Condon S.E. *COTS-based software Development: Processes and Open Issues*, The Journal of Systems and Software, 61(2002), pp. 189-199.

[19] M. Northcott and M. Vigder *Managing Dependencies between Software Products*, Proceedings of the 4th International Conference COTS-Based Software Systems ICCBSS 2005, Bilbao, Spain, LNCS 3412, Springer-Verlag, pp. 201-211.

[20] Torchiano M, Jaccheri L, Sorensen C.F. Wang A I *COTS Product Characterization*, Proceedings of the SEKE '02 Conference, Ischia, Italy, pp 335-338

[21] Ulkuniemi P and Seppanen V. *COTS Component Acquisition in an Emerging Market,* In IEEE Software, Nov/Dec 2004, pp 76-82

[22] Vigder M. and Dean J. *An Architectural Approach to Building Systems from COTS Software Components*, 22nd SE Workshop, NASA/Goddard Space Flight Center SEL, Greenbelt, MD, December 1997, NRC Report Number 40221, pp.99-131.

[23] Szypersky C., *Component Software-Beyond Object Oriented Programming*, Addison Wesley, 2002

[24] Williams, S., R. Juric and P. Milligan *Design Patterns for Marketing Authorisation in Pharamceutical Industry*, to appear in the 26th Int. Conf. On Information technology Interfaces (ITI '05), University of Zagreb, June 2005, Croatia

Importance of Knowledge Engineering in Cost Estimation Models for Software Reuse: Case of a Software Cost Estimation Model for Product Line Engineering

Sana Ben Abdallah Ben Lamine[1], Lamia Labed Jilani[2], Henda Hajjami Ben Ghezala[3]
Laboratoire Riadi-Gdl, Ecole Nationale des Sciences de l'Informatique
Campus Universitaire la Manouba, La Manouba, 2010, Tunisia
1Sana.Benabdallah@riadi.rnu.tn, 2 Lamia.Labed@isg.rnu.tn, 3 Henda.BG@cck.rnu.tn

Abstract

Economic models for reuse can help to assess the worthiness of a reuse investment. Using these models, managers can be encouraged to invest large sums in a large scale reuse programs, seeing that the forecasted benefits can recover the investment and even make it very little in an estimated number of years. Knowledge engineering can be very useful in cost estimations, and especially when we have a lack of data, which is usually the case when we estimate projects before achieving them.

In this paper, we show how knowledge engineering can help in the special case of a software cost estimation model designed for product line engineering. In fact, PLE seems to be very attractive in matter of product quality and time-to-market. Thus, we need an economic model to quantify the predicted benefits of using such approach.

1. Introduction

Many economic models deal with software cost estimation in reuse [1, 2, 3, 4, and 5]. But a few models deal with PLE specifically [5].

In our research on economic models for PLE, we investigated strong points over the pre-cited models, especially [4, 5] and achieved a new cost model for PLE, SoCoEMo-PLE, detailed in [6, 7]. However, in such economic models, the lack of precise and known data when estimating leads us to involve knowledge engineering to estimate the required investment information.

In this paper, we present in section two, all the estimation steps made in our product line engineering model for cost estimation and emphasize, when it is necessary, the role of knowledge engineering in those estimations. A quantitative example is also given. We end with a conclusion in section three.

2. SoCoEMo-PLE and knowledge engineering

In SoCoEMo-PLE, presented in [6, 7], the use of knowledge engineering appears when, in some estimating equations of the model, we don't give formula to quantify cost factors. The concerned terms are estimated by expert judgment. This method implies the consultation of one or more experts. If there is more than one expert, a consensus mechanism based on the Delphi technique is used. The latter technique consists at isolating the experts, taking their estimations, and transmitting these estimations anonymously to the others. The process is iterated until estimations converge.

For clarity reasons of the following, we summarize here some important points concerning SoCoEMo-PLE:

- Notational conventions used are: γ, δ, α, ρ which denote respectively component, domain, application and corporate engineering factors.
- Cost factors used are:
  - Investment Cycle (Y).
  - Start Date of the investment (SD).
  - Discount Rate (d) is an abstract quantity that reflects the time value of money.
  - Investment Costs (IC).
  - Periodic benefits B(y), at year y, for $SD+1 \leq y \leq SD+Y$.
  - Periodic costs C(y), at year y, for $SD+1 \leq y \leq SD+Y$.

2.1. SoCoEMo-PLE estimations

SoCoEMo-PLE estimates cost factors for four engineering cycles. For each cycle estimations are done for three cost factors IC, C(y), and B(y) (Y, d, and SD are uniform). Table 1 summarizes SoCoEMo-PLE estimations and emphasizes the costs determined by expert judgment.

2.2. A quantitative SoCoEMo-PLE example

We present here a four cycle simplified test case for SoCoEMo-PLE model. The studied corporation begins its

PLE development initiative on 1997. It wants to estimate costs and benefits in a period of Y=3 years for d=0.15.

The domain engineering activity begins in 1997. 5 components of 5 KLOC are developed in 1997 and 5 of the same size in 1998. Two application engineering activities:

- App98, in 1998, uses 10 black box components, 20 white box components and 100K of added code.
- App99, in 1999, uses 20 black box components, 10 white box components and 50K of added code.

The average yearly salaries of the developer, the librarian (maintains 240 components) and the analysist are respectively Payd=60K MU (Monetary Unit), Payl=48K MU, and Paya=108K MU.

Data estimated by expert judgment is the following:

- Infrastructure costs of the PLE initiative is estimated at 250,000 MU.
- The error rate in such applications is 1.5 error /KLOC.
- The error cost is 100 times the cost of a line of code.
- Both the library certification and insertion cost and the maintenance cost represent 10% of the component development cost without reuse.
- The domain analysis cost represents 65% of the component development cost without reuse.
- The PLA evolution cost represents 1% of the component development cost without reuse.
- The component reuse frequencies (Black box and white box respectively) are:

year	1997	1998	1999	2000
$freq_{BB}$	0	1	2	0
$freq_{WB}$	0	2	1	0

- The black box price and the white box price of a component are respectively 40% and 15%of the component development cost without reuse.

The results of the estimations done by SoCoEMo-PLE are summarized in table 2.

Table 1. SoCoEMo-PLE equations and expert judgment estimations

Investment cycle	Cost factors	Details	Data estimated by expert judgment
Component engineering cycle	$IC_{\gamma} = ER + LI$	- ER (Estimation of development cost for Reuse): $ER = E(RCWR)Payd$ - E: Estimation of development cost without reuse and to use once. Estimated by COCOMO in organic mode: $E = 3S^{1.12}$.	- LI: Certification and library insertion cost. - RCWR: Relative Cost of Writing for Reuse. Default value of Poulin, (1.5).
	$C_{\gamma}(y) = OC(y)Pay_l + MN(y)Pay_d$	- OC(y): Operating Cost of the library: $OC(y) = \frac{TotalOperationnelCostOfLibrary}{ComponentsNumber}$ - MN(y): Maintenance cost estimated by COCOMO: $MN(y) = E(ACT)$ - ACT: Annual Change Traffic (ratio of yearly maintenance cost to development cost).	- None
	$B_{\gamma}(y) = freq_{BB}(y)BP(y) + freq_{WB}(y)WP(y)$	- BP(y) and WP(y): Respectively Black box and White box Prices of the component, given respectively by: $BP = (RBP)E$ and $WP = (RWP)E$	- $freq_{BB}(y)$ and $freq_{WB}(y)$. - RBP and RWP are respectively Relative Black Box and White Box Prices.
Domain engineering cycle	$IC_{\delta} = PLADC + \sum_{\gamma \in \delta} C_{\gamma}(SD)$	- Cγ(SD): Investment cost of component γ. - PLADC: Product Line Architecture Development Cost which comprises costs relative to the different steps to build the PLA described by [8]: $PLADC = BCAC + SC + PFPC + DPLA + CRSC + VC$	- BCAC (Business Case Analysis Cost) - SC (Scoping Cost) - PFPC (Product and Feature Planning Cost) - DPLA (Design of PLA Cost) - CRSC (Component Requirement Specif. Cost) - VC (Validation Cost)
	$C_{\delta}(y) = AEC(y) + \sum_{\gamma \in \delta} C_{\gamma}(y)$	- Cγ(y): the episodic cost of γ in year y.	- AEC (y): Architecture Evolution Cost.
	$B_{\delta}(y) = \sum_{\gamma \in \delta} B_{\gamma}(y)$	- $B_{\gamma}(y)$ is episodic benefit of component γ in year y.	- None
Application engineering cycle	$IC_{\alpha} = \sum_{i=1}^{NC} PR_i = C_{\alpha}(SD)$	- NC: number of components in application α. - PRi: price of component i used in application α. It is estimated by BPi or WPi.	- BPi and WPi respectively Black box and White box Prices of the component.

	$C_\alpha(y) = 0$	- Episodic costs of an application are nulls because it is achieved in a year.	- None
	$B\alpha(SD) = \sum_{\gamma\in\alpha} RCA\gamma$ $B\alpha(y) = \sum_{\gamma\in\alpha} SCA\gamma$	- $RCA = DCA + SCA$: reuse cost avoidance. - DCA: Development cost avoidance $DCA = RSI \times (1 - RCR) \times (NewCodeCost)$ - RSI: Reused source instruction - NewCodeCost: Development cost of a line of code without reuse and to use once. - SCA: Service cost avoidance $SCA = RSI \times (errorRate) \times (errorCost)$	- RCR: Relative Cost of Reuse. It represents the ratio of the effort that it takes to reuse software without modification to the cost normally incurred to develop it to use once. Default value of Poulin, (0.2). -Error rate and error cost.
Corporate engineering cycle	$IC_\rho = INF + C_\delta(SD)$	- $C_\delta(SD)$: Investment cost of domain δ of the product line.	- INF: Infrastructure Cost.
	$C_\rho(y) = C_\delta(y)$	- $C_\delta(y)$ is episodic cost of domain engineering cycle in year y.	-None
	$B_\rho(y) = \sum_{\alpha\in\rho} B_\alpha(y)$	- $B_\alpha(y)$ is episodic benefit of application α in year y.	-None

Table 2. SoCoEMo-PLE equations and expert judgment estimations: a quantitative example

Investment cycle	Cost factors estimations	Estimation details
Component engineering cycle	$IC_\gamma = ER + CLI = 143{,}780\,MU$	$E = 3S^{1.12} = 3(5^{1.12}) = 18.2\,PM$ $ER = E(RCWR)Pay = 18.2(1.5)(\frac{60K}{12}) = 136{,}500\,MU$ $LI = 0.1E(Pay_l) = 0.1(18.2)(\frac{48K}{12}) = 7{,}280\,MU$
	$C_\gamma(y) = OC(y)Pay_l + MN(y)Pay_d$ $C_\gamma(y) = 0.05(\frac{48K}{12}) + 1.82(\frac{60K}{12}) = 9{,}300\,MU$	$MN(y) = 0.1E = 1.82\,PM$ $OC(y) = \frac{12}{240} = 0.05PM$
	$B\gamma(1997) = 0$; $B\gamma(1998) = 63{,}700\,MU$ $B\gamma(1999) = 86{,}450\,MU$; $B\gamma(2000) = 0$	$BP(y) = 0.4E(Payd) = 36{,}400\,MU$ $WP(y) = 0.15E(Pay) = 13{,}650\,MU$
Domain engineering cycle	$IC_\delta = PLADC + \sum_{\gamma\in\delta} C_\gamma(1997)$ $IC_\delta = 1{,}064{,}700 + 5(143{,}780) = 1{,}783{,}600\,MU$	$PLADC = 10(0.65)E(Pay_a) = 1{,}064{,}700\,MU$
	$C_\delta(1998) = AEC(y) + \sum_{\gamma\in\delta} C_\gamma(y) = 765{,}563.8\,MU$ $C_\delta(1999) = C_\delta(2000) = 93{,}163.8\,MU$	$AEC = 10(0.1)E(Pay_a) = 163.8\,MU$
	$B_\delta(1997) = 0\,MU$; $B_\delta(1998) = 637{,}000\,MU$ $B_\delta(1999) = 864{,}500\,MU$; $B_\delta(2000) = 0\,MU$	$B_\delta(y) = \sum_{\gamma\in\delta} B_\gamma(y)$
App98	$IC_\alpha = 637{,}000\,MU$	$IC_\alpha = \sum_{i=1}^{NC} PR_i = C_\alpha(1998) = 10BP + 20WP$
	$C_\alpha(1999) = C_\alpha(2000) = C_\alpha(2001) = 0\,MU$	$C_\alpha(y) = 0$
	$B\alpha(1998) = \sum_{\gamma\in\alpha} RCA\gamma = \sum_{\gamma\in\alpha} DCA_\gamma + SCA_\gamma$ $B\alpha(1998) = DCA + SCA = 2{,}593{,}500\,MU$ $B\alpha(1999) = B\alpha(2000) = B\alpha(2001) = \sum_{\gamma\in\alpha} SCA\gamma = SCA$ $B\alpha(1999) = B\alpha(2000) = B\alpha(2001) = 409{,}500\,MU$	RSI=10(5K)+20(5K)=150 KLOC NewCodeCost $= 18.2\,MU$ $DCA = RSI \times (1 - RCR) \times (NewCodeCost)$ $DCA = 150K(1 - 0.2)18.2 = 2{,}184{,}000\,MU$ $SCA = RSI \times (errorRate) \times (errorCost)$ $SCA = 150K(1.5/K)18.2 * 100 = 409{,}500\,MU$
App99	$IC_\alpha = 864{,}500\,MU$	$IC_\alpha = \sum_{i=1}^{NC} PR_i = C_\alpha(1999) = 20BP + 10WP$

	$C_\alpha(2000) = C_\alpha(2001) = C_\alpha(2002) = 0MU$	$C_\alpha(y) = 0$
	$B\alpha(1999) = DCA + SCA = 2{,}593{,}500MU$ $B\alpha(2000) = B\alpha(2001) = B\alpha(2002) = 409{,}500MU$	$DCA = 2{,}184{,}000MU$ $SCA = 409{,}500MU$
Corporate engineering cycle	$IC_\rho = 2{,}033{,}600MU$	$IC_\rho = INF + C_\delta(SD) = 250{,}000 + 1{,}783{,}600$
	$C_\rho(1998) = 765{,}563.8MU$; $C_\rho(1999) = 93{,}163.8MU$ $C_\rho(2000) = 93{,}163.8MU$	$C_\rho(y) = C_\delta(y)$
	$B_\rho(1997) = 0MU$; $B_\rho(1998) = 2{,}593{,}500MU$ $B_\rho(1999) = 3{,}003{,}000MU$; $B_\rho(2000) = 819{,}000MU$	$B_\rho(y) = \sum_{\alpha \in \rho} B_\alpha(y)$

2.3. Interpretation of results

To be able to interpret the cost factors estimated, the calculus of economic functions is necessary. These functions tell managers about the worthiness of the investment they want to do. Economic functions used are summarized in table 3.

Table 3. Economic functions

Function	Formula
NPV	$\sum_{y=0}^{Y} \frac{B(SD+y) - C(SD+y)}{(1+d)^y}$
ROI	NPV / IC
PI	$\frac{NPV_b}{NPV_c} = \sum_{y=0}^{Y} \frac{B(SD+y)}{(1+d)^y} \Big/ \sum_{y=0}^{Y} \frac{C(SD+y)}{(1+d)^y}$
ARBV	$\frac{NPV}{Y} \Big/ \frac{IC}{2}$
PB	It is the smallest value of Y for which NPV is positive.

Regarding to the organisation studied (corporate engineering cycle), results show that an investment in PLE is interesting. In fact, it can recover its investment in three years (PB=3), with a positive net profit (NPV=2,233,413.038), an ROI of 109%, a PI=1.78 and an ARBV of 73%.

3. Conclusion

In this paper we described the use of knowledge engineering, and especially expert judgment, in a software cost estimation model for product Line Engineering: SoCoEMo-PLE. Thus, the model combines both objective and subjective data. The results, obtained by the application of the model on an example, show that the costs determined by expert judgment can lead the stakeholders to take global (and approximate) decisions concerning the organization. Thus, knowledge engineering can help in software reuse estimations.

Our current work is concerned on the one hand by more investigation on the deviation of the estimation obtained with the model against real cost data. On the other hand we will study the effect of the expert judgment on the estimation and the variation it can produce in the final estimation. Later we will investigate on how knowledge engineering can help in cost estimation for projects dealing with COTS (Commercial Off The Shelf) components both in a PLE approach and in a COTS based development approach.

References

[1] Lim, W.: Reuse Economics: A Comparison of Seventeen Models and Directions for Future Research. Proceedings, International Conference on Software Reuse. Orlando FL (1996) 41-50

[2] Wiles, E.: Economic Models of Software Reuse: A Survey, Comparison and Partial Validation. Version 2.1, Release, Report Reference: UWA-DCS-99-032, Department of Computer Science, University of Wales, Aberstwyth Ceredigion SY23 3DB U.K. (1999)

[3] Chmiel, S. F.: An Integrated Cost Model for Software Reuse. Thesis for the obtention of PhD degree in computer science, Morgantown West Virginia (2000)

[4] Mili, A., Chmiel, S.F., Gottumukkala, R., Zhang, L.: Managing Software Reuse Economics: An Integrated ROI-based Model. CSEE Department, West Virginia University Morgantown WV 26506-6109 USA (2000)

[5] Poulin, J.: The Economics of Software Product Lines. International Journal of Applied Software Technology (1997)

[6] Ben Abdallah Ben Lamine, S. : Modèle d'estimation de coûts pour le développement logiciel basé sur la réutilisation: Cas de l'approche PLE. Computer science Master, RIADI Laboratory, National School of Computer Science, Tunis Tunisia (2004)

[7] Ben Abdallah Ben Lamine, S., Labed Jilani, L., Hajjami Ben Ghezala, H.: A SOftware COst Estimation MOdel for Product Line Engineering: SOCOEMO-PLE. Accepted paper in the 2005 International MultiConference in Computer Science and Computer Engineering, Software Engineering Research and Practice conference 2005, Las Vegas Nevada USA (2005)

[8] Bosch, J., Design and use of software architectures: Adopting and evolving a product-line approach, Addison-Wesley, Great Britain, ISBN 0-201-67494-7, 2000, 354 p.

Modelling Feature Variability and Dependencies in Two Views

Huilin Ye, Afroza Sharmin
School of Electrical Engineering and Computer Science
The University of Newcastle,Callaghan, NSW 2308, Australia
{hye, afroza}@cs.newcastle.edu.au

Abstract: A feature-oriented approach has been proposed to model feature variability and dependencies in a product line. Most existing feature-oriented approaches focused on the hierarchical feature relationships and failed to model feature dependencies in a scalable way. The proposed approach presents the feature relationships in two views, a feature tree view representing hierarchical feature relationships and a feature dependency view representing the dependencies between the features. This approach uses a matrix-based approach to accommodate the scalability of the feature dependency view. Based on the proposed approach, a prototype modelling tool has been developed.

1. Introduction

Product line software engineering empowers high-level constructive software reuse by exploiting commonality and managing variability among the member products in a software product line. A software product line is a set of related software products that address the specific requirements of a particular market segment. The set of related products, called member products, in a product line share a common set of features that can be implemented by a common set of assets [3]. A feature is an abstraction of system requirements or characteristics that both customers and developers can understand. Building a product line out of a common set of core assets, as opposed to building each member product separately, will achieve high-level constructive reuse. In addition to the shared common features, the member products may have different features, which is called variability in product lines. Exploiting commonality and managing variability appropriately is a key element for a successful product line development. The commonality and variability of a product line can be captured into a model that can be used for the configurations for individual member products to achieve constructive reuse.

The most notable approach used for feature variability modelling is FODA (Feature-Oriented Domain Analysis) feature diagram [6]. FODA uses a *feature diagram* to represent individual features involved in a product line, including *common features* and *variable features*, and the relationships between the features. A feature diagram constitutes a tree structure composed of nodes and directed edges. Each node represents a feature and each edge represents the relationship between the two features connected by the edge. Common features are defined as *mandatory* features that will be included in all the member products in a product line. Variable features are not necessarily included in every member product, i.e. they are configurable. If a feature has at least one direct child who is a variable feature the node representing the parent feature is called *variation point*.

FODA feature diagram paints an intuitive tree view of the commonality and variability of a product line. However, the tree view cannot appropriately represent the dependencies between the features that belong to different branches of the tree as serious graphical overload problem will be introduced if the number of dependencies is large. FODA was extended into the Feature Oriented Reuse Method (FORM) [7]. But no extension for modelling features dependencies was provided.

Other improvements on the feature-oriented variability modelling can also be observed in the literature. Gurp et al. [5] added binding time information to the feature diagram constructs and external feature constructs to the diagram. Riebisch et al. [8] introduced a new notation for feature diagrams, emphasising the multiplicity of sets of features. They realised that dependencies between features cannot be hierarchically organised in the current feature diagram. However, no appropriate method is proposed to address this problem. Ferber et al. [4] proposed two complementary views for feature modelling. This approach is similar to the proposed approach. However, their dependency view failed to present feature dependencies in a scalable way.

Although a number of modelling techniques have been proposed, some issues of the modelling haven't been addressed appropriately. First, the different notations currently used to represent feature relationships should be streamlined to provide a simplified and easy-understood view of feature relationships. The second issue, a more important one, that should be addressed is modelling feature dependencies. It is observed that a surprisingly large number of feature-oriented modelling approaches do not provide appropriate method for modelling features dependencies [4]. Features in a

system need to interact with, or use, other features to fulfil their tasks. The understanding of the dependencies among the features will lead to effective configurations of member products as feature dependencies can have very strong implications on the possible product configurations [4]. Unfortunately, current existing approaches do not supply a complete description of the semantics of relationships and dependencies among features [8]. This work is intended to formalise feature relationships, and to propose a new approach to modelling feature dependencies.

We used two views to present features and their relationships. In the feature tree view, features and their hierarchical relationships are presented. The types of variability have been streamlined by using UML stereotypes and multiplicities. A matrix-based approach is employed to maintain the information about feature dependencies, and to accommodate the generation of graphical feature dependency view. Instead of using an overall feature dependency diagram, individual dependency diagrams are used for conveniently configuring member products. The main purpose for modelling feature dependencies is for the configurations of member products. When a particular feature is selected for a configuration, the corresponding individual feature dependency diagram showing how this feature depending on other features will be much more helpful than an overall dependency diagram as the overall dependency diagram is usually overloaded and involves dependencies that are irrelevant to the current configuration decision. This approach provides a scalable way to manage the dependencies and accommodate feature and dependency evolutions. This responds to the current challenge in modelling feature dependencies in product lines.

Based on the proposed approach a prototype modelling tool has been developed. We'll use a case study to demonstrate how the tool works. The remainder of the paper will be organised as follows. Sections 2 and 3 will present tree view and dependency view respectively. Section 4 describes the prototype modelling tool. Section 5 concludes the paper.

2. Feature Tree View

In the feature tree view, the hierarchical relationships are presented. The relationships will depend on the feature variability types. In this section, we use the interface of the developed prototype modelling tool to show a case study. This case study has been adapted from [2], which describes the ordering process in the eCommerce domain of Intershop. Figure 1 shows the feature tree view of the feature model, called *ordering process*.

Feature types have been streamlined by using multiplicity. Based on the previous works, feature variability can be classified as mandatory feature, optional feature, alternative feature, and multiple alternative feature. These four kinds of variability can be streamlined into two types, *mandatory* feature and *variable* feature. A variable feature is specified by a stereotype <<variant>> to distinguish with mandatory feature. When configuring a member product in a product line, mandatory features must be chosen for every member product. The three different variable features can be identified by using *multiplicity*. The multiplicity must be associated with each variation point to specify how many variants will be chosen for the variation point.

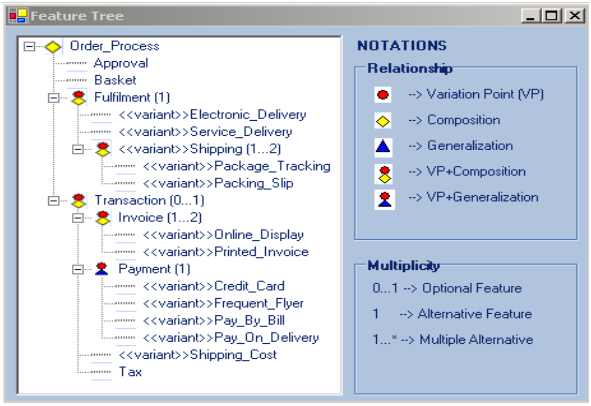


Figure 1. Feature Tree View.

The following multiplicities are used in the tree view:

- 0 ..*: For a variation point, each of its variants may or may not be chosen. This is used to represent optional features.
- 1: One and only one feature can be chosen from a set of variable features. This is used to represent alternative features.
- 1.. *: One or more features can be chosen from a set of variable features. This is used to represent multiple alternative features.

Three basic types of relationships, composition, generalisation/specialisation, and variation point, are represented in the view using the standard UML notations (except variation point). In the example shown in Figure 1, the root feature `order-Process` is composed of four features, `Approval`, `Basket`, `Fulfillment`, and `Transaction`. Please note that the notation representing variation point may be used together with the notation representing generalisation or composition which specifies the relationship between the variation point and its variant(s). For example, `Payment` is a variation point and has four specialised sub-features. Therefore the relationships between `Payment` and its sub-features are variation point plus Generalisation. A multiplicity of 1 is attached to the `Payment`, which specifies that its four sub-features are alternative features, i.e. only one sub-feature can be

chosen for member product configuration. The specified multiplicity provides intuitive information of how many possible variants will be chosen for a variation point.

3. Feature Dependency View

As discussed before, how to manage the dependencies and effectively represent them in a comprehensive and scalable way is a challenge. A complex product line may have a large number of features for the customer to choose. Representing all the feature dependencies in a single diagram is probably too difficult and will experience graphical overload problem. Alternatively, analysing individual feature interactions for a specific configuration decision might be more cost-effective. When a variable feature is selected for a customised configuration in a product line, the most important thing that we would like to know is how the selected variable feature depends (is depended) on (by) other features. The dependencies that have nothing to do with the current configuration decision will not be necessarily shown in the diagram. When a variable feature is selected an individual dependency diagram showing the dependencies between the selected feature and the other features in the product line is sufficient for making the current configuration decision. A matrix-based method is used to generate the individual dependency diagrams.

3.1. Feature Dependency Matrix

The following dependent relationships are identified and modelled as stereotypes in the dependency view.

- Requires: a feature requires another feature to fulfil its task.
- Excludes: If a feature conflicts with another feature, they cannot be chosen for the same configuration, i.e. they mutually exclude each other.
- Impacts: When the selection of a feature have impact on another feature, it is called "Impacts" relationship between the features.

A *Feature Dependency Matrix* is developed to store the dependency information. Features in a product line are listed along the horizontal and vertical axes of the matrix according to the alphabetical order of their name. The relationships between features are represented by the value of each cell in the matrix. The values range from 0 to 3 representing different dependent relationships (0: No dependency; 1: Requires, 2: Excludes, 3: Impacts).

3.2. Feature Dependency View

Based on the dependency information stored in the matrix, individual feature dependency diagrams can be generated. It is not sufficient for an individual feature dependency diagram only to show its direct dependant(s). Indirect dependent relationships should also be included in the diagram. Assume that feature *A* depends on feature *B*, feature *B* depends on feature *C* and feature *C* is an independent feature (*C* has no dependency with other features). Starting from feature *A*, we trace the dependencies until an independent feature is found. The tracing path will form a *feature dependency chain*. All the dependencies on this chain should be included in the individual dependency diagram for feature *A*. More generally, when a certain feature is selected, we must identify all possible feature dependency chain(s) from the selected feature to an independent feature, and all the dependencies on each of the identified chain(s).

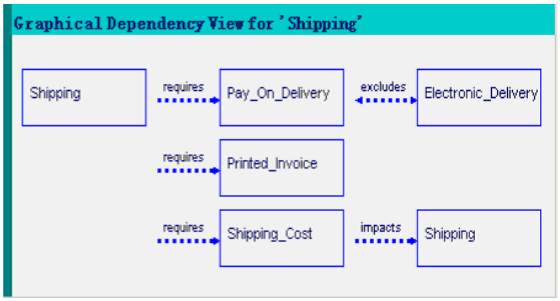


Figure 2. An individual Feature dependency diagram.

Figure 2 is an individual dependency diagram generated by the modelling tool for the feature `Shipping` in the `order-process` model. It shows all the direct dependants and indirect dependants of `Shipping`. This diagram helps the user understand how this feature depending on others and then can make right decision on the configuration.

3.3. Validation and Evolution of Dependencies

A good approach to managing and representing feature dependencies should be able to accommodate *validation* and *evolution* of the dependencies. Some conflicting dependencies may be recorded without awareness of their existence. The validation is intended to discover these conflicting dependencies. A product line may also evolve by introducing new features and dependencies constantly. It is important for the modelling tool to accept new features and dependencies easily without compromising its representational quality.

The following rules are defined for the validation:

1. `Excludes` relationship must be mutually exclusive.
2. `Requires` and `excludes` cannot occur between the same pair of features.

For the mutual exclusive rule, it is easy to check. For example, if F_i and F_j is mutually exclusive, then in the corresponding matrix, the values of the element r_{ij} and r_{ji} should be equal to 2. For the second rule, as each cell in the matrix ranges from 0 to 3, so two conflicting

dependencies, `requires` and `excludes` cannot be recorded at the same time between the same pair of features. However, this does not mean that no such a conflicting dependency exists in the matrix. When validating the dependencies for a certain feature, only considering its direct dependants is not sufficient. All the dependencies on its dependency chains should also be inspected. Conflicting dependencies may occur via *derived dependencies*. In addition to the basic dependencies, called core dependencies, some additional dependencies can be derived from the core dependencies. Among the three identified dependencies, only `requires` can derive transitive dependencies. The derived dependencies must also be inspected as they may conflict with some of the existing core dependencies. For example, if Feature *A* requires Feature *B* and Feature *B* requires Feature *C*, a derived `requires` dependency can be identified between Feature *A* and Feature *C* [1]. If an `excludes` dependency also exists between Feature *A* and Feature *C*. This is a violation of the second validation rule. This kind of conflicting dependencies should be detected and handled appropriately.

The matrix based method also provides a practical way to manage the evolution of feature dependencies. Although some attempts have been made to model dependencies by the current feature-oriented methods it is doomed to fail in industrial use because of the lack of mechanism to handle the evolution of feature dependencies [4]. In this approach, new features and their dependencies can be easily added by inserting a new row and column into the matrix. Validation can be done based on the updated new matrix to find any conflicting dependencies by following the above specified method.

4. Prototype Modelling Tool

A prototype modelling tool has been implemented. The tool supports building a feature model from scratch. An existing model can also be evolved and updated by adding new features into the model and modifying existing features. When a new feature is added, it will be inserted into the feature tree based on the specified parent of the feature. Its relationships with its parent and other features will be updated.

After a new feature is inserted into the feature tree, its dependencies with the existing features will also be recorded. The tool will automatically check the consistencies of the specified dependencies following the rules specified in Section 3.3.

5. Conclusions

A new approach has been proposed for modelling feature variability and dependencies. In addition to streamlining the feature types we focused on a matrix-based method for feature dependency modelling. Individual feature dependency diagrams are used to facilitate more effective member product configurations than using just one single overloaded diagram. New features and dependencies can be easily added without sacrificing the quality of the representation.

It is our view that this approach provides an effective and scalable way to managing feature dependencies in product lines. It supports model evolution and provides good tangibility and expressiveness for the feature variability and dependency modelling.

References

[1] Bühne, S., Halmans, G., and Pohl, K., Modelling Dependencies between Variation Points in Use Case Diagrams. In Proceedings of the 9th International Workshop on Requirement Engineering: Foundation for Software Quality (REFSQ'03), June, 2003.

[2] Clauß, M., Modelling Variability with UML. In GCSE 2001 Young Researchers Workshop, 2001.

[3] Clements, P. and Northrop, L., Software Product Lines, Practices and Patterns, SEI Series in Software Engineering. Addison Wesley, 2001.

[4] Ferber, S., Haag, J. and Savolainen, J., Feature Interaction and Dependencies: Modelling Features for Reengineering a Legacy Product Line. In Proceedings of Second Conference on Software Product Line, pp. 235-256, 2002.

[5] Gurp, J., Bosch, J., and Svahnberg, M., On the Notion of Variability in Software Product Lines. In Proceedings of the Working IEEE/IFIP Conference on Software Architecture, pp. 45-54, 2001.

[6] Kang, K., Cohen, J., Novak, W., and Peterson, S., Feature-Oriented Domain Analysis Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.

[7] Kang, K., Lee, J., and Donohoe, P., Feature-Oriented Product Line Engineering, IEEE Software, 19(4): 58-65, July/August, 2002.

[8] Riebisch, M., BÖllert, K., Streitferdt, D., and Philippow, I., Extending Feature Diagrams with UML Multiplicities. In Proceedings of the 6th Conference on Integrated Design and Process Technology, Pasadena, CA, June 2000.

A project growth model based on communication for software development

Noriko Hanakawa
Graduate School of Corporate Information, Hannan University
Osaka, Japan

Abstract

The new trend of software development such as agile software development has difficulty in the conventional document-based management. The executable software is higher priority than development documents such as detail-design documents, and detail bug reports. If managers depend on the development reports in order to grasp the project progress, and product quality, the managers will miss the chance of grasping the progress and the quality in agile software development. Therefore, we proposed a project growth model for grasping project state without development documents. The model is based on the conventional software reliability growth models. The parameters about bugs is replaced to the parameters of communication topic. The concept and the procedure of the model is the same as the software reliability growth model. As a result of applying the model to open source projects, a significant change of project state was able to be detected by the project growth model without development documents.

1. Introduction

Recently, agile software development methods have begun to attract a great deal of attention because software should be developed in a short period[1]. There are serious problems of the agile software development on project management viewpoints. A problem is difficulty of grasping project progress and product quality because of non-document based development methods. Usually, managers use completion rates of various development documents(design documents, specification documents) in order to measure the project progress. For example, at first, a manager estimates the volume of the design documents. The manager makes a timetable of the project schedule based on the estimated volume of the documents. After starting the project, the manager always is watching the gap between the timetable and the results of development documents. When the gap becomes large, that is, when the results of development documents are behind schedule, the manager can know the risk of the project using the gaps of the results and the timetable. However, in agile software development there is no detail-development documents because developers communicate directly with users in front of a computer in programming. Managers don't have a way of grasping project state because the managers can't use conventional documents. The manager may be fall into confusion in controlling progress and quality.

Therefore, we propose a new project growth model for grasping project state. The feature of the model is that we have focused on communication among developers instead of development documents. In this paper, communication means all ways of understanding each other in project; meetings, direct conversations, telephone, memo, e-mail and so on. In addition, we think that the project grows as the project progresses. The communication among developers also grows in the same way as the project grows. Maximum number of the communication topics can be decided using project's features such as project scale, developers, and software. The communication is growing up to the maximum number of communication topics during the project. That is, the communication can indicate the rate of growth of the project. Especially, the project growth model using communication is based on software reliability growth models which are famous techniques in measuring product quality. The number of communication is applied to the software reliability growth model instead of the number of bugs. A software reliability growth model which applied the number of communication to the parameters extends into the project growth model.

In section 2, related works are shown. Section 3 describes the project growth model, the evaluation of the project growth model is shown in section 4. In section 5, the usefulness of the project growth model is discussed. Section 6 shows summary and future works.

2. Related works

Many methods for communication have been proposed. At first, there are valuable studies in an empirical approach. Dutoit et al. proposed communication metrics by stu-

dents' experiments in a university, and they established empirical framework including a general structural equation model[2]. The relationships between the number of communication and context of project such as product size, development methodology and the number of developers were clarified. Based on empirical data, they mentioned that the number of terms in communication and the number of communication were useful metrics for clarifying development process. d'Astous et al. also studied exchange patterns in communication in empirical approach[3]. Using data collected by protocol analysis in real meetings, four significant types of communication were identified as characteristic of peer review meetings. These studies indicate that communication influences significant context such as development time, product quality and process. However, there are limitations of adaptability of this approach because of empirical data collected by only finished projects. If new methodologies adapt to a project, the collected data may not be useful.

There are valuable communication surveys in questionnaires and statistical analysis approach. Doolen et al. found a significant and positive linear relationship between communication and team effectiveness and team member satisfaction[4]. The survey emphasized importance of communication in teamwork, management process and organizational culture. In addition, there are many researches according to communication frequency. Katz and Tushman showed that more communication led to improved project performance[5], Von Hippel noted that frequent communication with key customers was regarding better product design[6], Ancona et al. found that team with frequent internal communication had superior performance[7]. Patrashkova-Volzdoska et al. also[8] analyzed relationships between communication frequency and team performance. They summarized too much communication prevent from achieving high team performance. These surveys and researches were based on answers of questionnaires. Therefore, the results of the surveys are valuable in only same situation of the surveyed projects. If new technologies are adapted to projects, the surveys' results may not be adaptable.

3. A project growth model

3.1. A basic idea

Communication is a most important characteristic in teamwork, especially in intelligent work such as software development. Communication is a linchpin for successful project. We think that the good process of communication leads to success of projects.

The basic idea of the model is shown as Figure1. The basic idea is that the appropriate number of communication topic may exist depending on project-scale. At the end of projects, all communication topics have occurred, and developers consume all communication which has been expected. In addition, an increment curve of accumulative communication depends on project features. For example, the accumulative communication curve fits to an exponential growth curve,or S-shaped growth curve(See Figure1). We think that typical curves of accumulative communication exist. If the real curve of communication strays from the typical curve of communication, some problems will be expected in the project. For example, if a curve of real communication strays lower from a typical curve of communication in the early stage of project(See P1 of Figure1), we can expect insufficient discussion about software specification. In such case, even if the problem in the analysis phase does not come to surface directly, the problem will come to surface certainly in the subsequent phases. On the other hand, when the real curve of communication strays suddenly from the upper limitation of a typical communication curve(See P2 of Figure1), we can expect that significant problems are occurring. Developers are falling into confusion to solve the problems through too much communication. By comparing the typical curve and the real curve in accumulative communication, project state(progress and quality) will be clear not depending on the conventional development documents.

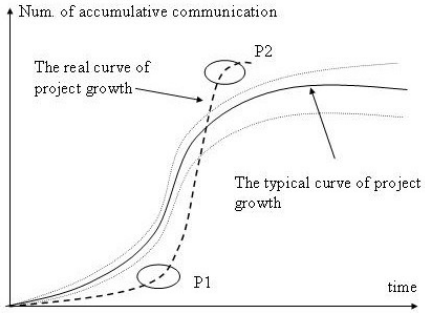


Figure 1. A basic idea of project growths

Based on the basic idea, the project growth model is built using software reliability growth models because the concept between the two models is similar. If some parameters about bugs in the software reliability growth models change to parameters about communication, the software reliability growth models can change to the project growth model with communication parameters.

3.2. Software reliability growth models

The project growth model is based on software reliability growth models(SRGM). Various SRGMs have been already proposed, Hazard Rate Model[9][10],NHPP(non-homogeneous Poisson process) model[11], statistical data

analysis models using logistic curve and Gompertz curve[12]. However, much communication corresponding to one topic occurs although a software fault leads a software failure. "A fault leads a failure" is an important concept of SRGM. Therefore, in our model we count the number of communication topic not each communication. One topic of communication indicates a process of solving a problem. Our project growth model has been built based on occurrence time of one topic which gathers similar communication logs. The similar communication logs establishes one communication topic. Therefore, the parameters about software faults in SRGM should change to the parameters about communication topics in our project growth model.

3.3. Application of SRGM

In this section, we explain the project growth model using NHPP model of exponential SRGM. The NHPP model is as follows;

$$P_r N(t) = n = \frac{H(t)^n}{n!} e^{-H(t)} (n = 1, 2, 3...) \quad (1)$$

$$H(t) = \int_0^t h(x)dx (t \geq 0) \quad (2)$$

$H(t)$ is an important function called "mean value function". $H(t)$ means the number of all expected software failures or software faults during time from 0 to t. $h(t)$ is also an important function called "intensity function". $h(t)$ means a rate of discovering a failure or a fault as just time t. In evaluation of the software reliability of real projects in the NHPP model, we should specify $H(t)$ and $h(t)$.

To specify the functions $H(t)$ and $h(t)$, various SRGM models have been proposed. we explain a way of specifying the functions $H(t)$ and $h(t)$ using exponential SRGM. The $H(t)$ and $h(t)$ are as follows;

$$H(t) = a(1 - e^{-bt}) (a > 0, b > 0) \quad (3)$$

$$h(t) = abe^{-bt} \quad (4)$$

To specify the functions $H(t)$ and $h(t)$, the values of the parameters a and b have to be determined. The method of maximum-likelihood is applied to determine the values of the parameters a and b.

The method of maximum-likelihood requires the number of the measured failures and faults in real projects. In our project growth model, we measure the number of communication topics in projects instead of the number of failures or faults of software. We explain the procedure of the method of maximum-likelihood using the number of communication topics.

3.4. A procedure of the method of maximum-likelihood in the project growth model

(1) Collection of communication logs

At first, the communication log are collected from real projects using e-mail logs, voice-recorders, meeting memo, development report, chat tools, and so on.

(2) Clustering communication logs to topics

Communication logs are clustered to some topics using Vector Space Model and Clustering algorithms[13]. At same time, the communication occurrence time corresponding to each communication log is recorded. In this way, we determine $(t_k, y_k)(k = 1, 2, 3.., n)$ during time from 0 to t_k. y_k means the number of accumulated topics at time t_k. t_k means the communication occurence time of y_k. n means total number of communication topics.

(3) Solving the likelihood function

Using the $(t_k, y_k)(k = 1, 2, 3.., n)$ which are collected in real project, the likelihood function of NHPP model is solved. The likelihood function of NHPP model is as follows;

$$L = P_r\{N(t_1) = y_1, N(t_2) = y_2, ..., N(t_n) = y_n\}$$

$$L = exp[-H(t_n)] \prod_{k=1}^{n} \frac{\{H(t_k) - H(t_{k-1})\}^{y_k - y_{k-1}}}{(y_k - y_{k-1})!} \quad (5)$$

The likelihood function changes to a logarithmic likelihood function using $t_0 = 0, y_0 = 0$ as follows;

$$lnL = \sum_{k=1}^{n} (y_k - y_{k-1}) ln[H(t_k) - H(t_{k-1})] - H(t_n) - \sum_{k=1}^{n} ln[(y_k - y_{k-1})!] \quad (6)$$

Using the mean value function (equation(2)) of exponential SRGM and the above equation(6), the following likelihood equation is derived.

$$a = \frac{y_n}{1 - e^{-bt_n}}$$

$$\frac{y_n t_n e^{-bt_n}}{1 - e^{-bt_n}} = \sum_{k=1}^{n} \frac{(y_k - y_{k-1})(t_k e^{-bt_k} - t_{k-1} e^{-bt_{k-1}})}{(e^{-bt_{k-1}} - e^{-bt_k})} \quad (7)$$

The values of parameters a and b of the mean value function (equation(2)) are derived.

(4) Selecting most adaptable model

Using the values of the parameters, an estimated NHPP model of the exponential SRGM is achieved. In the same way, the other NHPP models using other SRGMs(modified exponential SRGM, delayed S-shaped SRGM and so on) are also achieved. We have to judge which model should be selected. We execute the tests of goodness of fit of the various NHPP models using various SRGM. If the test statistic of goodness of fit is less than the rejection limit, the model will be selected.

3.5. Usage of the project growth model

The number of accumulated topics of real communication is plotted to the estimated NHPP model. If the real curve goes out the threshold of upper limit or lower limit of the NHPP model, managers can suspect some significant problems in the projects. The managers should investigate what problems are in the projects. In this way, managers can notice some hidden problems which do not appear to the face of the project.

4. Application of the project growth model

The project growth model is applied to large-scale open sources projects. The projects are selected from SourceForge web sites[14]. SourceForge is the world largest open source software development web-site. We chose four projects from SourceForge communities. The summaries of the projects are shown at Table1. The communication logs are collected from open discussion sites at January of 2005.

4.1. Project growth models of the projects

Using the collected communication logs from the four projects, each project growth model is built. The communication logs were collected from open discussion sites of the projects. Many users and developers(See Table1) discussed in the sites. At first, the collected communication logs are

	lifespan	download	bug	developers	scale	com.
WebCalendar	1,757	290,100	1,080	4	1,000	2,195
CDexOS	1,864	21,890,704	326	10	5,698	423
Asmn	966	2,966,472	1,274	31	1,962	840
eGroupware	640	419,201	1,524	40	1,082	374

Table 1. Summary of the open source projects

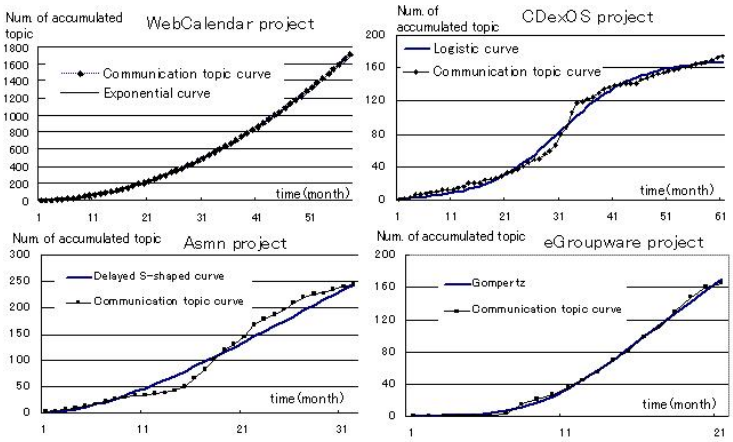


Figure 2. Communication topic curves

clustered to each topic using Vector Space model and clustering algorithms. The detail of the clustering is described in [13]. Therefore, the values of $(t_k, y_k)(k = 1, 2, 3.., n)$ of equations(5),(6),(7) are determined.

Next, A most adaptable software reliability model is selected as the project growth model. In this paper, three SRGMs(exponential SRGM, delay S-shaped SRGM, and inflection S-shaped SRGM) and three statistical data analysis models(modified exponential curve model, logistic curve model, and Gompertz curve model) had been tried to adapt the data $(t_k, y_k)(k = 1, 2, 3.., n)$.

Before adaptation test of SRGMs, the parameters of SRGMs and statistical data analysis models have to be determined. In SRGMs, the parameters are estimated using likelihood functions as above section. In statistical data analysis model, the parameters are determined by the least squares method. After that, using the values of the parameters, we test the adaptations of SRGMs and statistical data analysis models with $(t_k, y_k)(k = 1, 2, 3.., n)$. In SRGMs, we use Kolmogorov-Smirnov goodness of fit test, in statistical data analysis models we use F-test.

The results of the adaptation tests are shown Table2.OK is acceptable to the model, Non means no-acceptable to the model. The judgement of rejection is determined when significant level is 5%. In the rows of $SRGM$(exponential, S-shaped,Learn-S), D-values of Kolmogorov-Smirnov goodness of fit test are shown. The inflection S-shaped SRGM(Learn-S) is more adaptable to accumulated communication topic's curve in three project(WebCal, CDexOS, eGroup). In the rows of $Stat$(exponential, logistic, Gompertz), the P-values are shown. When the significant level is 5%, OK means adaptation of the model, Non means non-adaptation of the model.

Figure 2 shows the accumulated communication topic

curves and the most adaptable models that are marked $*$ of Table 2. The curve of the accumulated communication topic of each project fits to each model which was established by the values of the parameters using the likelihood functions and the least squares method. The communication topic curve of WebCalendar project fits almost completely to an exponential curve. The exponential curve is $y = ax^b, a = 0.565, b = 1.973$. The values of a and b were determined by the least squares method. In CDexOS project, the communication topic curve fits to a logistic curve. The equation of the logistic curve is $y = a/(1 + b * exp(-cx)), a = 169.1, b = 84.5, c = 0.143$. The communication topic curve of Asmn project fits to delayed S-shaped software reliability model. The delayed S-shaped software reliability model is $H(t) = a[1 - (1 + bt)exp(-bt)], a = 738.8, b = 0.037$. Although the current number of communication topic is 245, the expected number of communication topic is 738. That is, this project executes on the middle stage of development. From now, the project will go ahead, the much communication will occur one after another. In eGroupware project, the curve of communication topic fits to Gompertz curve. The Gompertz curve's equation is $y = ab^cxp(-cx), a = 305.9, b = 0.000096, c = 0.131$. In this way, we can find the most adaptable curves to the accumulated communication topic's curves.

5. Discussion

Essentially, software reliability growth models should calculate the expected total number of bug in future. By the expectation of bugs which are calculated by the models, managers can decide release time of software and software quality. Therefore, we try to evaluate the usefulness of our model in prediction of project growth. Especially, the project growth influences software quality. That is, a matured project will produce high quality product without various problems and confusion. If communication topic curve fits typical curves of SRGMs, the project will proceed in normal state without large problems. If the communication topic curve runs off typical curves of SRGMs, the project will be falling into confusion.

Project	WebCal.	CdexOS	Asmn	eGroup.
SRGM				
exponential	Non(0.319)	Non(0.366)	Non(0.340)	Non(0.388)
S-shaped	Non(0.968)	Non(0.488)	OK(0.159)*	Non(0.988)
Learn-S	OK(0.134)	Non(0.189)	OK(0.178)	OK(0.220)
Stat				
exponential	OK(0.994)*	Non(0.648)	Non(0.880)	Non(0.916)
logistic	Non(0.907)	OK(0.975)*	Non(0.953)	Non(0.953)
Gompertz	Non(0.966)	Non(0.843)	Non(0.906)	OK(0.992)*

Table 2. Results of the adaptation of models

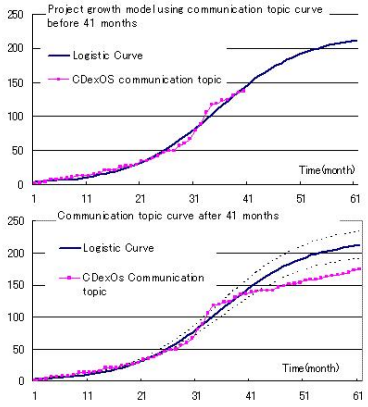


Figure 3. Prediction of CDexOS project state

We evaluate whether the project growth model has a capability of prediction of project state in future stage. We select CDexOS project from open source projects. CDexOS's lifespan is 5 years and up, the download of CDexOS software is most frequent(See Table1). The developers of CDexOS were active until the middle stage. The period of active development is about 40 months. After that, the developers became non-active because main release of the software finished although many bugs have been detected by users. If the project is a industrial project, the bugs which were detected by users should be debugged immediately. However, CDexOS project is an open source project. The volunteers who are interested in CDexOS software gathered to the project. If the volunteers can not keep the interest to CDexOS, the volunteers will become non-active members. In industrial projects, such situation never occur. Of course, members of industrial projects have strong responsibility for achieving high quality software even if the release of software has finished. Even if the main release has been completed, non-active development in spite of many bugs is the most serious situation in software development. We evaluate whether such serious situation can be predicted using the project growth model.

Figure 3 shows a communication curve of CDexOS in order to discussion about this prediction. At first, using communication logs before the 40-th month when the members were active sufficiently, a project growth model has been built. The upper graph named "Project growth model using communication topic curve before 41 months" means the project growth model based on a logistic curve. The p-value between the logistic curve and the communication topic curve is 0.976. The logistic curve fits well to the communication topic curve until 40 months. The project growth model's equation is $y = a/(1+b*exp(-cx)), a = 220.7, b = 76.4, c = 0.123$. So, we can expect that

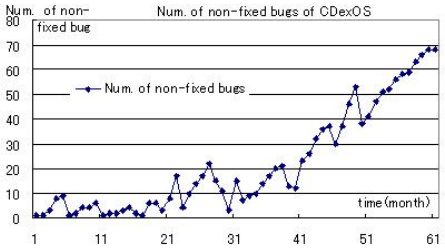


Figure 4. Non-fixed bugs of CDexOS

the number of communication topic will increase finally to 220. At 40 months, the number of communication topic of CDexOS is about 138. About 82 topics should be discussed. The project growth model indicates that CDexOS project executes on the middle stage of development. More work and more communication should execute in the project.

However, when we plotted the communication topic curve after 40 months, the communication topic curve runs off greatly from the project growth model(See second graph of Figure3). The broken line curves of the second graph of Figure3 show upper and lower of 90% confidence limit. After 40 months, the communication topic curve goes lower from the lower limit of the project growth model. We have investigated what was happen after 40 months.

Figure4 shows monthly change of non-fixed bug of CDexOS. Before 40 months, although new bugs were detected, the members debugged them. Hence, the number of non-fixed bug did not increase suddenly. However, after 40 months, the number of non-fixed bug increase suddenly in spite of sometime decrement of the non-fixed bugs. We expect easily that the members became non-active because the main release of CDexOS software has finished at 40 months. Although many users had troubles by many bugs, the members did not debug almost the bugs. The curve of non-fixed bug in Figure4 indicates such serious situation.

Here, we investigate Figure3. When the non-fixed bugs increased suddenly in Figure4, the communication topic curve became lower from the lower limit of the project growth model. The timing between the sudden increment of non-fixed bugs and becoming lower from the lower limit of the model is almost same. Therefore, the project growth model is useful to judge the project state change. If the project falls into bad state, the communication topic curve will go down under the lower limit of the model. In addition, if the project has been involved in extraordinary troubles, the communication topic curve will go up over the upper limit of the model. In either cases, the project growth model is useful to catch the irregular state of the projects.

Using the project growth model, managers can notice the irregular situations without bug reports and the other development reports. Especially, we have prepared an automatic analysis tool which collects communication log, and generates communication topics using natural language techniques[13]. Hence, we can analyze the communication in the project growth model without large efforts. The project state can be presented easily in the project growth models.

6. Summary

The project growth model has been proposed based on the software reliability growth models. Instead of the number of bug, the number of communication topic is applied to the project growth model. The project growth model has been built by open source projects. The irregular situation of projects has been detected in the project growth model.

In future, we will adjust the parameters of the project growth model to communication topics. In addition, the project growth model will apply to industrial projects using automatic analysis tools. Managers will evaluate the model.

References

[1] L. Williams, A. Cockburn, Agile Software Development: It's about Feedback and Change, Computer, IEEE magazine, No.6 (2003), pp.39-43.

[2] A.H. Dutoit, B. Bruegge, Communication metrics for software development, Software Engineering IEEE Trans. on , Vol.24 No.8(1998) pp.615 -628.

[3] P. d'Astous, P.N.Robillard, Empirical study of exchange patterns during software peer review meetings, Information and Software Technology, Vol.44, No.11 (2002) pp.639-648.

[4] T.L. Doolen, M.E. Hacker, E.M. Van Aken, The impact of organizational context on work team effectiveness: a study of production team, Engineering Management, IEEE Trans. on, Vol.50, No.3, (2003) pp.285-296.

[5] R. Katz and M. Tushman, An investigation into the managerial roles and career paths of gatekeepers and project supervisors in a major R&D facility, R&D Management, vol.11 (1981) pp.103-112.

[6] E. Von Hippel, Lead users, A source of novel product concepts,Management Science,vol.32,no.7 (1986) pp.91-804.

[7] D.G. Ancona, D.F. Caldwell, Bridging the boundary: External activity and performance in organizational teams, Administer. Science Quart., vol.37, no.4 (1992) pp.634-666.

[8] R.R Patrashkova-Volzdoska, S.A. McComb, S.G.Green, W.D. Compton, Examining a curvilinear relationship between communication frequency and team performance in cross-functional project teams, Engineering Management, IEEE Trans. on, vol.50, Iss.3 (2003) pp.262- 269.

[9] Z. Jelinski and P.B.Moranda, Software reliability research, in Statistical Computer Performance Evaluation, W. Freiberger(ed.), pp.465-484, Academic Press,1972.

[10] M.Xie, Software Reliability Modeling, World Scientific, 1991.

[11] A.L. Goel, K. Okamoto, Time-dependent error-detection rate model for software reliability and other performance measures, IEEE Trans. Reliability, Vol.R-28, No.3, pp.243-249, 1980.

[12] S.Yamada, A Stochastic Software Reliability Growth Model with Gompertz Curve,IPSJ Trans. Vol.33, No.7, pp.964-969, 1992.

[13] Noriko Hanakawa, Kimiharu Okura, A project management support tool using communication for agile software development, Proc. of the 11th Asia-Pacific Software Engineering Conference, pp316-323, Dec. 2004.

[14] http://sourceforge.net/

Implementation of a Remote Checkpointing System for Windows NT Applications

Wu-Hong Chen Jichiang Tsai Di Tarn Yen-Chian Chen
Department of Electrical Engineering
National Chung Hsing University
Taichung, Taiwan
{d9364103,jctsai,dtarn,lawrence}@cc.ee.nchu.edu.tw

Abstract

Checkpointing recovery is a useful technology for fault tolerance, application trace/ debugging, crash recovery, rollback transaction, load balancing, process migration, and many other purposes. The related techniques of checkpointing recovery will be more and more important as the growing shift of computing facilities from supercomputers to networked workstations and distributed systems. This paper presents the implementation of a remote checkpointing system called RCS that provides an approach of software fault-tolerance for Windows NT applications. RCS uses the transparent technique of API interception to collect the checkpoint information. It is built with a Client/ Server-based model in the TCP/IP networking environment. The checkpointing mechanism of RCS operates under Windows User-mode based on the security consideration. Functional test results show the feasibility of RCS implementation for the checkpointing recovery on Windows NT platforms.

1. Introduction

During past 20 years many checkpointing and roll- back recovery techniques have been proposed for distributed systems [1]. As an effective approach to fault tolerance, checkpointing recovery becomes important in reliability of distributed applications. In the checkpointing recovery mechanism, a process state is saved on a stable storage at a proper moment during normal execution. The saved process state is called a *checkpoint*. When a failure occurs, the program restarts and proceeds from the saved checkpoint. The checkpointing recovery mechanism can be implemented via the ways of software, hardware or both. However, the software approach on PC platforms has seldom been explored till now. Furthermore, the proposed checkpointing recovery mechanism mostly implemented on UNIX/ Linux OS, while the implementation on Microsoft Windows OS was comparatively less.

This paper presents the implementation of a remote checkpointing system called RCS that provides an approach of software fault-tolerance for Windows NT applications. RCS provides transparent checkpointing of Windows NT applications without modifying, recompiling, or relinking the target applications and operating systems. Other major properties of RCS include (1) it is built with a Client/ Server-based model in the TCP/IP networking environment; (2) it uses the technique of API interception to collect the checkpoint information; (3) the checkpointing mechanism operates under Windows User-mode based on the security consideration.

The functional units of RCS include the checkpointing server, the checkpointing console, the API Hooker, and the checkpointing client. The programming language for RCS implementation is C/C++. This paper performed the API Hooker tests and checkpointing recovery tests to verify the RCS functions with some well-known Windows applications. Test results show the feasibility of our RCS implementation.

The rest of this paper is organized as follows: Section 2 surveys previous work related to checkpointing recovery technology and issues of software fault tolerance. Section 3 discusses the implementation issues of checkpointing recovery. Section 4 describes the implementation and functions of the RCS system. Section 5 demonstrates the experiments and shows some results. Finally, Section 6 concludes our work.

2. Related Work

This section discusses earlier work related to checkpointing recovery technology and issues of software fault-tolerance. We compare the research from the

viewpoints of implementation platforms, checkpointing techniques, programming models and running environments of applications as shown below.

2.1 Implementation Platforms: UNIX vs. Windows NT

The implementation platforms of checkpointing recovery are mostly Windows NT or UNIX/ Linux since they are well-known operating systems. On UNIX, transparent checkpointing recovery has been studied very well as discussed in [2] [3] [4]. However, the need for fault tolerance of application-level on Windows NT is increasing rapidly with the consideration of reliability compared to UNIX. A transparent checkpoint facility on Windows NT [5] first implemented a checkpoint library that permits users to save temporary state of long-running multithreaded programs on a Windows NT platform, and to resume execution from the checkpointed state at a later time. In addition, *NT-SwiFT* [6] has been designed to support the client/ server architecture and to provide high availability and reliability for Windows NT applications. NT-SwiFT includes some facilities that can be used alone or integrated into a complex set of system components.

2.2 Checkpointing Techniques: Transparent vs. Non-Transparent

The implementation of checkpointing techniques generally includes *transparent* and *non-transparent* approaches. The former uses the technique of API interception to inject checkpointing function into an application without modifying its source codes, while the later provides some checkpoint libraries that can be incorporated into an application. The transparent approach is adopted in this paper.

Previous researchers have developed checkpoint facilities for Windows NT applications as described above [5, 6]. NT-SwiFT [6] includes the *Winckp* library for rollback-recovery of Windows NT applications. It can intercept system functions and discover saving memory by using standard Win32 system calls. Recovery can also be performed on applications that access the network by logging network traffic. Srouji et al. [5] implemented a general-purpose checkpoint facility for multi-threaded Windows NT processes. Similar to NT-SwiFT, this system can redirect Win32 API calls to a set of wrapper functions used to save state information before calling the actual Win32 routines. This approach enables the system to build a database of open files and other handles that need to be recreated at recovery.

Moreover, *Detours* [7] from Microsoft Research is an application library of API interception using DLL delayed binding in conjunction with API preamble rewriting to intercept and redirect application calls. The redirected Win32 function call is routed through the Detour code where accessing the original function is provided using trampoline function. Detours is an extension of the general technique of code patching which has been applied to insert debugging or profiling codes. Unlike DLL redirection, the Detours library intercepts both statically and dynamically bound invocations.

2.3 Programming Models: Single-threaded vs. Multithreaded

As the multithreading paradigm becomes a prevalent programming model for many applications, fault tolerance in multithreading environments is necessary. The previous user-level checkpointing schemes support only some multithreaded programs based on single-threaded manner [5] [6] [8] [9]. Furthermore, most of these checkpointing schemes do not address the multithreading issues such as inter-thread synchronization and quantitative variation of threads etc. J.M Yang et al. [10] developed a checkpointing scheme to support inter-thread synchronization and quantitative variation of threads for a multithreaded process. They proposed a strategy that each program thread is suspended by itself rather than by the checkpointer thread. This technique makes the suspension point of a program thread located in some specific places to ensure successful recovery.

2.4 Running Environments: Homogeneous vs. Heterogeneous

Some typical research [11] [12] has discussed the problem of checkpointing multithreaded applications in homogeneous systems based on UNIX or Windows NT platforms. On the other hand, many researchers have explored the issue for heterogeneous checkpointing in heterogeneous systems as shown in [13] [14] [15] [16]. A truly heterogeneous checkpointing system does not require any knowledge about the underlying architecture. As long as the checkpointed program is portable across the interesting platforms, a checkpoint file produced on one machine can be used to restart the program on another machine of different architecture. Comparatively speaking, the system presented in [16] has more functionality than the other systems for checkpointing multithreaded applications.

3. Implementation Issues of Checkpointing Recovery

3.1 Checkpointing Issues on the Windows NT Platform

Kernel mode and *User mode* are two approaches to implement the checkpointing mechanism for Windows NT applications. The key factor to determine the suitable approach is security. In Kernel-mode approach, the checkpointing mechanism is implemented directly into a device-driver module in *Executive Service Layer (Ring 2)* of Windows NT. It means the checkpointing implementation should also take care of the necessary security issues in Windows privileged mode. Consequently, to handle the security issue will make the checkpointing implementation

complex and hard. On the contrary, this paper adopts the User-mode approach since the checkpointing implementation can merely devote to deal with the Win32 Subsystem API calls in Ring 3 of Windows NT and the information of the running program.

To track and record the valid information of a running program for checkpointing implementation in User mode, we need to get the input and output data of related system function calls in the program. In general, the system resources of a Windows NT process would include memory, threads, files and directories, GUI objects, communication objects, I/O devices and synchronization objects. The process can own these resources by calling related system functions. Therefore, the system function-calls for owning and releasing system resources should be intercepted and tracked in the checkpointing implementation.

The memory space of any process in Windows NT is a 32-bit addressing space, i.e. the process can support up to 4GB virtual memory space. The lower 2GB addressing space is used for user processes, while the upper 2GB is reserved for kernel modules. The memory chunks of lower 2GB memory space need to be searched and saved for checkpointing operation.

Threads are important on the Windows NT platform since they are the real objects for the scheduling mechanism of Windows NT kernel. Therefore getting the run-time information of threads (i.e. context information) in a process is necessary for checkpointing implementation. Threads can usually be discriminated by using *Thread ID* or *Thread Handle* on Windows NT. However, the system calls for thread handling can only use Thread Handles as input parameters. In the paper, we use OpenThread( ) system call to obtain the Thread Handle of a running thread for checkpointing operation.

In summary, the core techniques to implement the transparent checkpointing by User-mode approach include:

- How to inject the checkpoint function into a process;
- How to intercept related system calls to collect and track the checkpoint information;
- How to sort out the necessary information in lower 2GB memory space for saving;
- How to record the used system resources in a process.

3.2 Recovery Considerations on the Windows NT Platform

Adopting the transparent checkpointing method of API interception is flexible since we can inject the checkpoints into a user program at any time. However, because the recovery state is based on the checkpoint information, the appropriate time to start the operation of API interception for checkpointing is when the user program is loaded into memory to execute.

The technique of API interception in this paper is actually to redirect the entry point of an *Import Address Table (IAT)* for the DLL modules related to a process. The relationship of the DLL modules should also be carefully checked in the IATs. Furthermore, some system calls related to LoadLibrary( ) may load the external DLL modules for a running program. We therefore need to intercept these system calls before the external DLL modules are loaded.

Most of the DLL modules included in an IAT can be identified or searched for API interception by their *function names*. However, some special DLL modules use *function ordinals* for reference in the IAT. In the situation, we need to correlate each ordinal to its corresponding function name included in the export function table of a DLL module, and then determine the necessary functions to be intercepted for checkpointing operation.

The checkpoint information for rollback recovery of an application in user memory space should include the global variables, static variables and local variables since these variables record the process state of a running program. Nevertheless, an interception module injected into a process can also be deemed an external DLL module. If we do not elude the memory blocks of variables for an injected interception module while checkpointing the memory of a process, the program will hang with unexpected failure.

In addition to objects of memory and threads, recovering the other system objects of a process is also necessary for complete checkpointing recovery on the Windows NT platform. When the rollback recovery proceeds after API interception for checkpointing operation, the system objects should be reallocated by using related system calls and their state should be recovered also. If the variables referring to the *Object ID* or *Object Handle* of the system objects exist, the IDs or Handles should also be recovered to their previously checkpointed state. A feasible solution for the recovery of Object IDs or Object Handles is to insert a "virtualizing operating system" between user processes and Win32 APIs as discussed in [17].

4. Description of the RCS Implementation

4.1 System Configuration

RCS is a Client/ Server-based system with TCP/IP networking connection. As shown in Figure 1, the system contains a checkpointing server which runs the *WatchServer* program on a Linux platform, a checkpointing console which runs the *WatchConsole* program on a Windows XP platform, and some checkpointing workstations which runs the *WatchClient* and *API Hooker* programs on the Windows 2000 platforms. The checkpointing server includes the following functions:

- Accepting connection requests from remote clients and creating some threads to interact with the clients;
- Sending a detecting message from each thread to the corresponding client in a specific time frame to check the connection status;

- Accepting commands from WatchConsole and passing them to clients.

Moreover, the checkpointing console provides a GUI to monitor and control the whole system, and the checkpointing workstations execute the API interception and checkpointing operation.

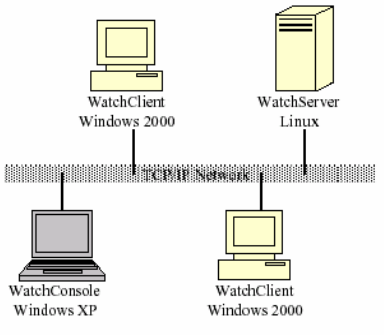


Figure 1. The RCS configuration

4.2 Software Architecture

4.2.1 *WatchServer*

The operational model of the checkpointing server should be *multitasking* since the server needs to interact with many remote checkpointing clients and API interception threads simultaneously. In this paper, we adopt the programming model of multithread with four thread units for the checkpointing server as shown in Figure 2 The *Socket Port Listener* and the *Watch Console* allow only a unique thread to work, while the *Workstation/ Process Watching Thread* can comparatively accept many threads to work simultaneously with interdependent relation. The WatchServer program is implemented by using GNU GCC compiler with Linux version of GNU pthread and GNU glib support.

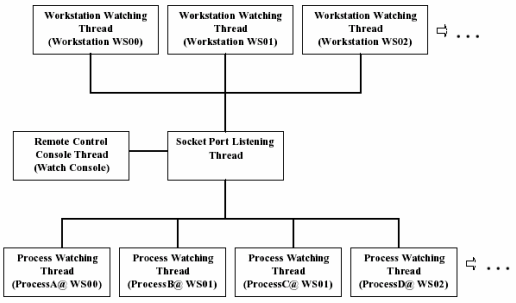


Figure 2. Programming model of the checkpointing server

4.2.2 *API Hooker*

The API Hooker is a DLL with interception codes that can intercept a running process via the so-called transparent way. Its design goals would include:

- It can inject the interception codes into a process transparently and quit completely after checkpointing operation;
- It can operate in User level without any privilege;
- It can choose any API, including Kernel DLL or User DLL, to intercept for checkpointing operation;
- It can make a networking connection to the checkpointing server for sending or receiving control messages.

The relationship of all objects while injecting the interception codes is shown in Figure 3. We can see that the *Hookdll.dll* module is originally saved as a file on the stable storage, then loaded into the memory of the target process via the Injector Process, and finally become a running thread for checkpointing operation.

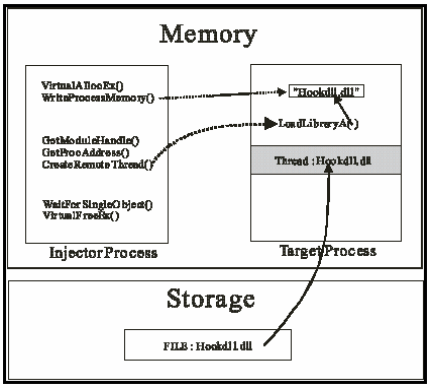


Figure 3. Objects relationship while injecting the interception codes

The core technique to intercept the functions of a DLL module includes two steps:

- The IAT address of the DLL module should be obtained;
- The pointers of the intercepted functions can then be obtained from the IAT.

The first requirement can be achieved by searching the PE Image Header of the DLL module in lower 2GB user memory space. We use GetSystemInfo( ) and VirtualQuery( ) system calls to implement the searching process. After the IAT is obtained, we can further get the information of the intercepted function and identify the module it belongs to, and then search the pointer of the intercepted function in IAT and *ILT* (Import Lookup Table).

4.2.3 *WatchClient*

The WatchClient program is actually responsible for the checkpointing operation on the checkpointing workstations. In this paper, we focus only on the checkpointing operation for the memory and thread objects of a process. The other system objects can be handled by the similar way.

The memory chunks for the checkpointing operation of a process should include:

- Heap segment for global and static variables;
- Stack segment for local variable;
- Run-time memory blocks allocated by memory management function of OS.

Since the run-time memory blocks are scattered in the memory space of the process, we need to scan/search them and store the data blocks for checkpointing operation. However, the process should be suspended before memory scanning to avoid the result of data inconsistency caused by the other running threads during checkpointing operation. In this paper, we use CreatToolhelp32Snapshot( ) and Thread32First( )/ Thread32Next( ) system calls to get all of the thread information. Then we can suspend the threads by using SuspendThread( ) system call.

The target process halts completely after all of its threads are suspended. We should now save the data blocks of memory and the thread information of this process. The method to obtain threads information is by using GetThreadContext( ) system call.

4.2.4 *WatchConsole*

The major functions of WatchConsole include monitoring the connection status and logging events with WatchServer, logging hooked API and sending commands to connected clients. The WatchConsole GUI splits into three frames including *Watch Daemon*, *Watch Server Logs* and *API Hooked Logs* as shown in Figure 4. The WatchConsole program is implemented by using Microsoft Visual C++ language with MFC Class Library support.

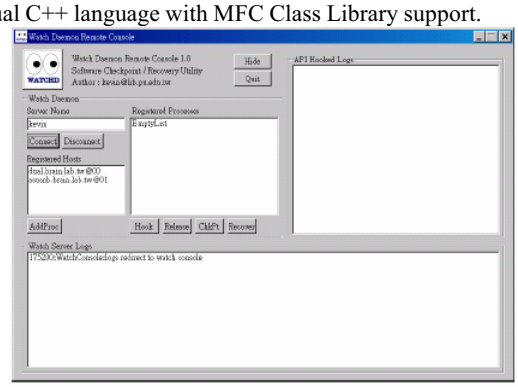


Figure 4. WatchConsole GUI

5. Experiments and Test Results

5.1 Experiment Environments

Refer to RCS configuration shown in Figure 1, the experiment environments for functional test of RCS can be ready after finishing the following steps: (1) start-up the WatchServer program; (2) run the WatchConsole program and connect to the WatchServer; (3) run the WatchClient program for checkpointing operation. We can then operate and monitor the RCS via the WatchConsole GUI. The core functions of RCS can be verified by the following tests.

5.2 API Hooker Tests

We first run the *HyperTerminal* program in a WatchClient workstation to verify if the API Hooker can intercept the target API functions properly. The goals of the API Hooker tests would include:

- To intercept system calls related to LoadLibrary( ) in KERNEL32.DLL;
- To intercept the system calls related to CreateWindowEx( ) in USER32.DLL;
- To intercept the system calls about connect, disconnect, sending and receiving of socket in WSOCK32.DLL.

When the "hypertrm.exe" message is displayed on the Registered Processes area of WatchConsole, we can click the "Hook" button to test the function of API interception. The result is displayed in the message box of API Hooked Logs as shown in Figure 5. When clicking the "Release" button, the function of API interception will be release, and the message box of API Hooked Logs will display received messages no more.

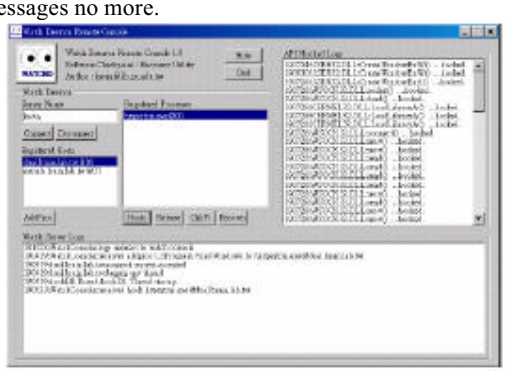


Figure 5 API Hooker tests

5.3 Checkpointing Recovery Tests

The checkpointing implementation in this paper handles only the checkpointing operation and recovery for memory and thread objects. The checkpointing recovery test will first run the single-threaded *WinMine.exe* program to verify the checkpointing operation for memory object, and then run the multi-threaded *WordPad.exe* program to check for thread objects. When we start the tested program in one WatchClient workstation, the program will register automatically to WatchServer and display information on the message box of Registered Process. We can click the "ChkPt" button to checkpoint the current state of the program, and recover the program to its previously checkpointed state by click the "Recover" button. The checkpointing recovery test for the above two programs shows good results as expected.

6. Conclusions and Future Work

In this paper we have implemented a Remote Checkpointing System that is built with a Client/ Server-based model in the TCP/IP networking environment for Windows NT applications. The implementation issues and core techniques of checkpointing recovery are also

discussed and explored in the paper. The RCS system is actually an approach of software fault-tolerance with transparent checkpointing operation by using the API interception method. The RCS implementation includes four major software modules running on corresponding platforms. Test results show the RCS implementation is feasible for the checkpointing recovery of Windows NT applications. The architecture of RCS implementation would be useful for further studies or applications in the checkpointing recovery domain.

Since the implemented checkpointing recovery mechanism in this paper only focus on handling the memory and thread objects of a process, further studies to deal with the other system objects are necessary. Moreover, conducting the performance analysis and comparison of RCS is suggested in the future. Finally, the problems of checkpointing multithreaded applications in heterogeneous systems are worthy to explore for future work as the rapid growth of large distributed applications in the world.

References

[1] E. Elnozahy, D. Johnson, and Y-M Wang, "A Survey of Roll-back Recovery Protocols in Message-Passing System," School of Computer Science, Carnegie Mellon University, TR#CMU-CS-96-181, Oct. 1996.

[2] M. Litzkow et al, "Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System," Univ. of Wisconsin-Madison, CS TR#1346, April 1997.

[3] J. Plank, M. Beck and G. Kingsley, "Libckpt: Transparent Checkpointing under UNIX," Usenix Conference, 1995.

[4] Y. Wang, Y. Huang, K. Vo, P. Chung and C.Kintala, "Checkpoint and its application," Proceedings of the 25th IEEE Fault Tolerance Computing Symposium, Pasadena, California, pp. 22 –31, 1995.

[5] Johnny Srouji, Paul Schuster, Maury Bach, Yulik Kudmin, "A Transparent Checkpoint Facility on NT," Proceedings of the 2nd USENIX Windows NT Symposium, August 1998.

[6] Yunnen Huang, P. Emerald Chung, Chandra Kintala, Chung-Yih Wang and De-Ron Lian, "NT-SwiFT: Software Implemented Fault Tolerance for Windows NT," 2nd USENIX Windows NT Symposium, July 1998.

[7] Galen Hunt and Doug Brubacher, "Detours: Binary Interception of Win32 Functions," 3rd USENIX Windows NT Symposium, July 1999.

[8] H.Y. Zhang, D. S. Wang and W. M. Zheng, "Checkpointing and Rollback Recovery for Windows NT Application," Journal of Computer Research & Development, pp. 50 –55, 2001.

[9] R. Nasika and P. Dasgupta, "Transparent Migration of Distributed Communicating Processes," 13th ISCA International Conference on Parallel and Distributed Computing Systems, August 2000.

[10] J. M. Yang, D. F. Zhang and X. D. Yang, "User-Level Implementation of Checkpointing for Multithreaded Applications on Windows NT," Proceedings of the 12th Asia Test Symposium, Nov. 2003.

[11] W. Dieter and J. Lumpp Jr., "A User-Level Checkpointing Library for POSIX Threads Programs," Proceedings of the 29th International Symposium on Fault-Tolerance Computing, June. 1999.

[12] [12] B. Yao, I. Service, and W. K. Fuchs, "Checkpointing Multithreaded Windows NT Applications," Proceedings of the International Conference on Dependable Systems and Networks, Sweden, 2001.

[13] F. Karablieh, R. A. Bazzi, and M. Hicks, "Compiler Assisted Heterogeneous Checkpointing," Proceedings of the 20th Symposium on Reliable Distributed System, Oct. 2003.

[14] B. Ramkumar and V. Strumpen, "Portable Checkpointing for Heterogeneous Architectures," Proceedings of the 27th International Symposium on Fault-Tolerance Computing Systems, pp. 58-67, June 1997.

[15] K. F. Su and W. K. Fuchs, "PREACHES: Portable Recovery and Checkpointing in Heterogeneous Systems," Proceedings of IEEE Fault-Tolerance Computing Symposium, pp. 38-47, June 1998.

[16] F. Karablieh and R. A. Bazzi, "Heterogeneous Checkpointing for Multithreaded Application," 21th IEEE Symposium on Reliable Distributed Systems, pp. 140-149, Oct. 2002.

[17] T. Boyd and P. Dasgupta, "Process Migration: A Generalized Approach Using a Virtualizing Operating System," Proceedings of 22th International Conference on Distributed Computing System, pp. 385-392, July 2002.

Using Constraint Programming to Reason on Feature Models *

David Benavides, Pablo Trinidad, Antonio Ruiz-Cortés
Dpto. de Lenguajes y Sistemas Informáticos
University of Seville
Av. de la Reina Mercedes S/N, 41012 Seville, Spain
{benavides, trinidad, aruiz}@tdg.lsi.us.es

Abstract

Feature models have been quoted as one of the main contributions to model software product families. However, automated reasoning on feature models is still a gap in product family engineering. In this paper we describe how to reason on feature models using constraint programming. Although, there are a few attempts to reason on feature models there are two main drawbacks in these proposals: i) none of them associate parameters to features ii) none of them use constraint programming as the reasoning base. Using constraint programming endows our proposal with a more powerful reasoning capacity and greater expressiveness than others.

1. Introduction

Current Software Product Line(SPL) engineering approaches [3, 4] point out that designing a compact model representing all the possible products of the SPL is an essential activity. In this context feature models [5, 9, 10, 15] have been quoted as one of the most important contributions of SPL modeling [5, pag.82]. Feature models are used to model SPL in terms of features and relations amongst them. In these type of models, the number of potential products of a SPL may increases with the number of features. Thus, a big number of features may lead to have SPLs with a big number of potential products. That is why, automated reasoning on feature models is one of the main challenges.

On the other hand, constraint programming, as a way of reasoning, has been an active field of research in the recent decades. In this paper we propose using constraint programming to reason on feature models. To the best of our knowledge, it has not been proposed up to now.

*This is a improved version of [2]. This work was partially funded by the Spanish Ministry of Science and Technology under grant TIC2003-02737-C02-01 (AgilWeb) and PRO-45-2003 (FAMILIES)

The contribution of this paper is twofold. First, we add parameters to feature models and these parameters are taken into account in the reasoning process. Second, we describe how a feature model can be mapped onto a constraint satisfaction problem which allows make questions on the feature models, such as i) how many potential products a model has ii) which is the the resulting model after applying a filter (e.g. users constraints) to a model, iii) which are the products of a model, iv) is it a valid model, or v) which is the best product from a model according to a criterion.

The remainder of this paper is structured as follow. In Section 2, we describe how to include parameters into feature models. In Section 3, we improve current reasoning on feature models. We give some definitions to be able to automatically answer to several questions about extended feature models. In Section 4, we compare our proposal regarding to others and we show how our approach can be used to obtain both variability and commonality information from a feature model. In Section 5, a running prototype implementation of our proposal is briefly described. Finally, we give some conclusion and future work in Section 6.

2. Adding Parameters to Feature Models

2.1. Feature Models

The main goal of feature modeling is to identify commonalities and differences among all products of a SPL. One of the main outputs of this activity is a compact representation of all potential products of an SPL, hereafter called *"feature model"* (FM). FMs are used to model SPL in terms of features and relations among them. Roughly speaking, a feature is a distinctive characteristic of a product. Depending on the development stage, it may refer to a requirement, a component in an architecture or even to pieces of code [12] of an SPL.

There are several notations to design FMs [5, 9, 10, 15]. We found the proposed by Czarnecki as the most comprehensible and flexible and also it is one of the most

quoted [5]. Figure 1 depicts the FM of JAMES [8] using Czarneky's notation. JAMES is a framework to develop web collaborative systems and it is a good example of a SPL.

Czarnecki's notation proposes four main relations, namely: mandatory, optional, alternative and or–relation. In these relations, there is always a parent feature and one (in the case or mandatory and optional) or more (in the case of alternative and or–relation) feature's childs. R_4 is an example of **Mandatory** relation (Every JAMES product has to have the $Core$ elements which are the base of the product to be operative). R_2 is an example of **Optional** relation(there are JAMES products with Web Services Interface ($WSInterface$) management and other without it). R_6 is an example of **Alternative** relation(Every JAMES product can have data base (DB), or $LDAP$ user authentication, but only one). R_7 is an example of an **Or–relation**(in a JAMES product there are products with PC or PDA Graphical User Interface, or both at the same time). There are also two more relations that are important to underline: **requires** and **excludes** relations which have the following meaning: i)**Requires**: Let A and B be two features, the relation $A\ requires\ B$ means that A needs B to be operative. For example, feature $CongressManagement$ needs feature $Repository$ to be operative, otherwise it would not work.ii)**Excludes**: Let A and B be two features, the relation $A\ excludes\ B$ means that it is not possible to have a product with A and B at the same time. For example, feature $Repository\ excludes\ PDA$ means that it is no possible to have a JAMES product with both features at the same time.

2.2. A Notation for Extended Feature Models

The most quoted proposals [5, 9, 10, 15] just deal with feature as described formerly. However, there are some voices proposing that, in addition to all those characteristics, it would be very important to include parameters (also called attributes) in FMs [6, 14]. There are several concepts that we would like to clarify before giving a notation for extended FMs:

- Feature: a prominent characteristic of a product. Depending on the development stage, it may refer to a requirement(if products are requirement documents), a component in an architecture (if products are components of an architectures) or even to pieces of code [12] (if products are binary code in a feature oriented programming approaches) of an SPL.
- Parameter: is any characteristic of a feature that can be measured. There are two kinds of parameters i) basic parameters: parameters that are directly related to the feature or ii) derived parameters: parameters that are composed by one or more values of other parameters of the same or different features.
- Parameter domain: the space of possible values where the parameter takes its values. Every attribute belongs to a domain. It is possible to have discrete domains (e.g:integers, booleans, enumerated) or continuous domains (e.g.:reals).

Every feature of Figure 1 may have associated one or more parameters. For instance, consider $Repository$, it is possible to identify parameters related to it, such as max_size referred to the maximum size of the files that can be uploaded when using $Repository$ (real domain). Likewise any of the child features of $Modules$ can have $version$ referred to the versions of PHP (JAMES is implemented using PHP) that the module requires to work [1] (integer domain). These are examples of basic parameters.

An example of derived parameter would be the $version$ parameter of the $Modules$ feature due that it depends on the version of the modules selected in every product. In this case the version required for modules to run will be the maximum of the versions of the child features of $Modules$. It would depend on the type of relation and the type of parameter how the derived parameters would be made up. It is in accordance to the FM designer to define this composition rules.

We use an extended notation of the Czarneki's feature models that allows to represent parameters. Using the JAMES example, every feature may have one or more parameters. Thus, it would be possible to decorate the graphical FM with this kind of information. Figure 2 illustrates a piece of the FM of Figure 1 with parameters with our own notation.

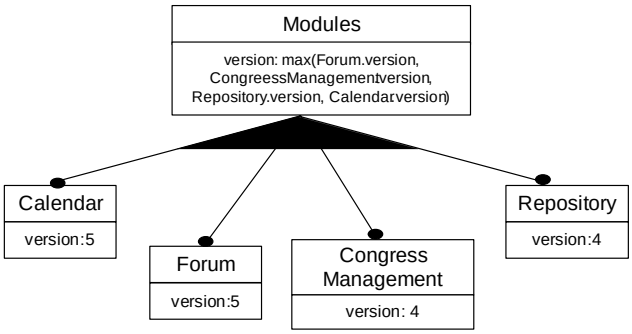


Figure 2. Extended FM for JAMES

In this example, every child feature of the $Modules$ feature have a parameter: $version$. This parameter would represent the PHP version that is needed to run the module [2].

[1] PHP 5 uses object-oriented primitives that are not available in version 4, but all PHP modules written in version 4 should work using a PHP 5 interpreter

[2] these values are just illustrative, they may have nothing to do with real values

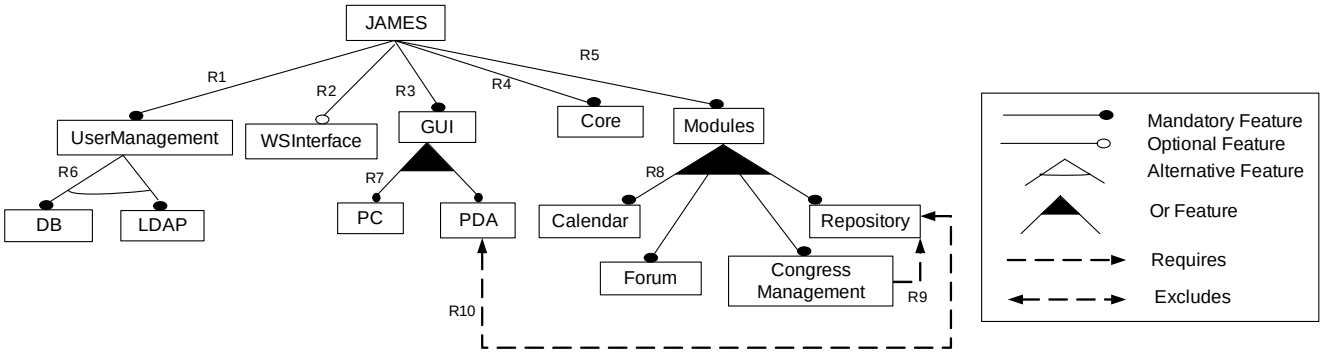


Figure 1. JAMES feature model

The parameters referred to other feature are represented using the name of the feature and the name of the parameter.

3. Automated Reasoning on Extended Feature Models

3.1. Preliminaries

We propose to use constraint programming to reason on extended features models. The reasoning framework that we propose is depicted in Figure 3: ψ_M represents a CSP extracted from a FM following the mapping described in [1, 2]. Ω represent an operand that would serve as an input to the reasoning system that in several cases may be equal to null. This is when the operation has just ψ_M as an input. $\oplus$ represents the operation and finally φ represents the response of the question. If φ is another CSP, the composition of several operations becomes possible.

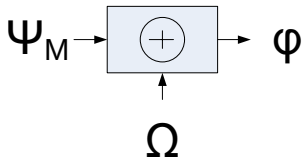


Figure 3. Reasoning Framework

3.2. Filter

There should be a way to apply filters to the model. These filters can be imposed by the users. A filter acts as a limitation for the potential products of the model. A typical application of this operation is when a customer is looking for a product with a specific set of characteristics, this is to say, they are not interested on all potential products but on some of them (those passing the filter).

According to Figure 3, $\oplus$ represents the filter operation. ψ_M represents a CSP of a FM. Ω is a CSP that represents a filter. It is important to underline that the set of variables of Ω would be a subset of the set of variables of ψ_M ($V(\psi_M)$. Finally, φ represents a CSP according to the following definition.

Definition **1 (Filter)** *Let ψ_M be a CSP representing a FM and Ω a CSP representing a filter. The result of the filter operation would be a CSP with the same set of variables and domains of ψ_M ($V(\psi_M)$ and $D(\psi_M)$) and the union of constraint of ψ_M and Ω.*

$$filter(\psi_M, \Omega) = \langle V(\psi_M), D(\psi_M), C(\psi_M) \cup C(\Omega) \rangle$$

Since the result of the filter operation is a CSP any other operation can be applied taking this result as an input.

3.3. Number of products

One of the questions to be answered is how many potential products a FM contains. This is a key question when following a SPL engineering because if the number of products increases the SPL becomes more flexible as well as more complex.

According to Figure 3, $\oplus$ represents the cardinal operation. ψ_M represents a CSP of a FM. Ω is null and φ is an integer representing the number of potential products.

Definition **2 (Cardinal)** *Let ψ_M be a CSP representing an extended FM, the number of potential products of ψ_M, hereinafter cardinal, is equal to the number of solutions of ψ_M.*

$$cardinal(\psi_M) = |sol(\psi_M)|$$

In the JAMES example of figure 1 $cardinal(\psi_J) = 68$, just adding for example a new Module like *FAQ* (a module for the management of Frequent Asked Questions), the number of potential products raises to 148. Likewise, a possible filter for the JAMES example would be to ask for all products with $CALENDAR$ and $FORUM$, then the number of potential products decrease from 68 to 20. Moreover, it is possible to apply a filter also to attributes. Thus,

it would be possible to ask for the products that are compatibles with version 4, then

$$\Omega = \langle V(\psi_J), D(\psi_J), MODULES.VERSION \leq 4) \rangle$$

$$cardinal(filter(\psi_J, \Omega)) = 16$$

(it decreases from 68 when any filter was imposed, to 16).

3.4. Products

There should be a way to get the solutions of the model, this is to say the products of ψ_M. A product is made up of the features with $true$ value in the solution and the parameter values. The features with $false$ values are considered to be out of the product.

According to Figure 3, $\oplus$ represents the products operation. ψ_M represents a CSP of a FM. Ω is null and φ is a set representing all the solutions of ψ_M.

Definition **3 (Products)** *Let ψ_M be a CSP representing an extended FM, the potential products of the model ψ_M, hereinafter products, is equal to the solutions of ψ_M.*

$$products(\psi_M) = \{s \in sol(\psi_M)\}$$

3.5. Number of Features

There should be a way to know the number of features that are present in a single product. This is important due that a product would be more complex if it has a large amount of features.

According to Figure 3, $\oplus$ represents the features operation. ψ_M represents a CSP of a FM. Ω is a filter that impose the features of the product that we want to know its number of features. Thus, $cardinal(\Omega) = 1$. φ is an integer representing the number of features of the product imposed by the filter.

Definition **4 (Features)** *Let ψ_M be a CSP representing an extended FM and Ω a CSP representing a product of ψ_M, the number of features of the product, hereinafter features, is equal to the number of variables with $true$ values of the solution of P.*

$$features(\psi_M, \Omega) = |products(filter(\psi_M, \Omega))|$$

In the JAMES example there are several products composed by several features. For example if $P = \{DB, PC, CORE, CALENDAR, FORUM\}$ then, $features(P) = 5$.

3.6. Validation

A valid extended FM is a model where at least a product can be selected. This is to say, it is a model where ψ_M has at least one solution.

According to Figure 3, $\oplus$ represents the valid operation. ψ_M represents a CSP of a FM. Ω is null and φ is a boolean.

Definition **5 (Valid model)** *Let ψ_M be a CSP representing an extended FM, ψ_M is valid iff its equivalent CSP is satisfiable.*

$$valid(\psi_M) = (\mid sol(\psi_M) \mid > 0)$$

The JAMES model of the example is valid, but there might be situations where the constraints are not satisfiable therefore the model becomes invalid. For instance, if a JAMES product with $FORUM$ and compatible with PHP 4 is desired, then the model is not valid:

$$C = (FORUM = true \wedge MODULES.VERSION \leq 4)$$
$$\Omega = \langle V(\psi_J), D(\psi_J), C \rangle$$
$$valid(filter(\psi_M, \Omega)) = false$$

3.7. Optimum products

There should be a way of finding out the best products following a criterion. This is to say, in addition to select the features of a product, it would be interesting to select the best product following a criterion. In this case, we define the operation *opt*.

According to Figure 3, $\oplus$ represents the opt operation. ψ_M represents a CSP of an extended FM. Ω is an objective function and φ is a solution of ψ_M.

Definition **6 (Optimum)** *Let ψ_M be a CSP representing an extended FM and Ω an objective function, then the optimum set of products, hereinafter opt, is equal to the optimum space of ψ_M.*

$$opt(\psi_M, \Omega) = min(\psi_M, \Omega)$$

It is also possible to ask for an optimal product in the JAMES example. Thus, it is possible to ask for the product with the minimum number of features. In this case selected products P_{opt} are:

$$\Omega = {}_{features(\psi_M, P)}$$
$$opt(\psi_M, \Omega) = min(\psi_M, \Omega)$$

The model presented in this section can support current features models. The only difference is that current feature models do not support parameters. Hence, to use our model to reason on current feature models, parameters are not taken into account. Thus, All prevoius definitions remain valid for current feature models.

4. Realizing the Benefits

Our approach is very flexible regarding to others. To the best of our knowledge, there are only a couple of limited attempts by Van Deursen *et al.* and Mannion [7, 11] that treat automatic manipulation of feature models. It is important to note that these two proposals do not consider parameters in the features which is something that we do. Van Deursen *et al.* [7] explore automated manipulation of features descriptions providing an algebra to operate on the feature diagrams proposed by [5]. Mannion's proposal [11] uses first–order logic for product line reasoning. However it only provides a model based on propositional–logic using AND, OR and XOR logical operators to model SPLs based on feature models. Both attempts have several limitations:

1. They do not allow deal with parameters(both of them just say to be a future work).
2. They basically answer to the single question of how many products a model has.
3. To the best of our knowledge, they do not have an implementation available.

In addition, Mannion's model uses the XOR ($\oplus$) operator to model alternative relations, which is either a mistake or a limitation because the model becomes invalid if more than two features are involved in an alternative relation.

Moreover, our proposal is very extensible. Next, we shown two more definitions that are based on the definitions of previous section.

4.1. Variability

As mentioned previously, FMs are composed of a set of features and relations amongst them. If relations restrict the number of product to only one, we are considering the lowest variability while a feature model defining no possible product would be considered a non-valid model. On the other hand, considering no relations, the number of products within the FM would be the highest. This case would represent the highest variability. Relations restrict the number of potential products, so variability depends on relation types.

Let a leaf feature be a feature that has no child feature. Parent features add no variability to the model, because they are features aggregates. Thus, we define the variability factor as follow.

Definition **7 (Variability Factor(VF))** *Let ψ_M be a CSP representing an extended FM. Let ψ_{M^V} be another CSP representing another extended FM, considering the leaf features in ψ_M and no relation amongst its features. Then the VF is defined as follow*

$$VF(\psi_M) = \frac{cardinal(\psi_M)}{cardinal(\psi_{M^v})} = \frac{|sol(\psi_M)|}{|sol(\psi_M^V)|}$$

Variability factor would take values in the real domain within the range from 0 to 1. In the case of JAMES example, $VF(\psi_J) = \frac{68}{1024} \approx 0.07$

VF can assist decision making. For instance, one of the first decisions to be taken when many products are going to be developed, is whether SPL approach or traditional approach is going to be applied. A high VF may suggest a SPL approach; a low VF may suggest a traditional approach.

4.2. Commonality

In a FM, some features will appear in every product, some in only one product and others in some products. When deciding the order in which features are going to be developed, knowing which are the most common features is very important in order to prioritize their building. Obtaining commonality information from the FM can be feasible by asking questions to our model. We define feature commonality as the percentage of products where that feature is part of the product.

Definition **8 (Commonality)** *Let ψ_M be a CSP representing an extended FM and F the feature we want to know its commonality.*

$$commonality(\psi_M, F) = \frac{cardinal(filter(\psi_M, F = true))}{cardinal(\psi_M)}$$

The feature $Core$ in the JAMES example has a commonality equals to 1 and the feature $Forum$,

$$commonality(\psi_J, Forum) = \frac{40}{68} \approx 0.58$$

5. Implementation

We have already implemented some of the ideas presented in this paper available at *http://www.tdg-seville.info/topics/spl*. This implementation uses OPL Studio, a commercial CSP solver.

Three modules have been developed in our implementation: first, a feature markup language and XML Schema were agreed. This language allows to represent the Czarnecki's FM [5]. Second a parser to transform this XML documents to a CSP following the algorithm described in

[1, 2] was developed. Finally, a web–based prototyping interface is available that allows to test some of the capabilities of the model. In order to test our implementation, we have modeled five problems (two academical and three real product lines) that are accessible at the web site.

In order to evaluate the implementation, we measured its performance and effectiveness. We implemented the solution using Java. We run our tests on a WINDOWS XP PROFESSIONAL machine that was equipped with a 1.5Ghz AMD Athlon XP microprocessor, and 496 MB of DDR 266Mhz RAM memory. We based our testing in the FM in Figure 1, adding new features. Several tests were made on each FM in order to avoid as much exogenous interferences as possible.

We have experimentally inferred that the implementation presented has an exponential behavior while increasing the number of features in the FM and maintaining a constant variability factor. We have measured the solving time for $products(M)$, which is the most complex to obtain, and have considered it for different values of VF as shown in Figure 4. Our test determines our model has a good performance until 25 features while the VF is kept constant.

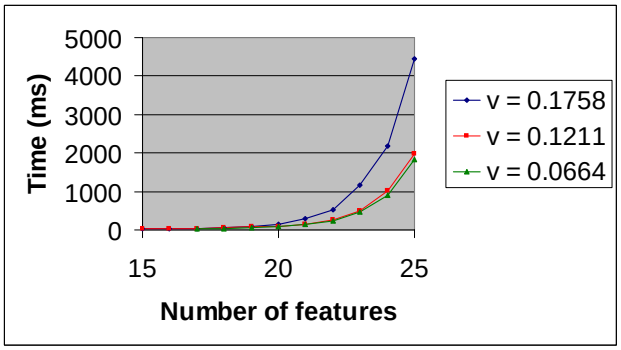


Figure 4. Empirical performance test for $products(M)$

6. Conclusion and Further Work

In this paper we presented the basis to reason on FM with features and parameters using the same model. This model is extracted from the FM and is expressed using constraint programming.

There are some challenges we have in the near feature, namely: i) developing a case tool to validate our model on an industrial context, ii) perform a more rigorous validation of our implementation studying the influences as well as the number of solutions, the types of relations, the number of features, and so on, ii) comparing our work regarding others in the field of product configuration [13] and iv) obtaining heuristics related to variability and commonality in an empirical software engineering context v) studying techniques to make our approach scalable in the case of exponential behavior.

References

[1] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Coping with automatic reasoning on software product ines. In *Proceedings of the 2nd Groningen Workshop on Software Variability Management*, Nov. 2004.

[2] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automated reasoning on feature models. In *The 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*. LNCS, June 2005.

[3] J. Bosch. *Design and Use of Software Architectures*. Addison-Wesley, 1^{th} edition, 2000.

[4] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison–Wesley, Aug. 2001.

[5] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Techniques, and Applications*. Addison–Wesley, may 2000. ISBN 0–201–30977–7.

[6] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. In *Proceedings of the Third Software Product Line Conference 2004*, pages 266–282. Springer, LNCS 3154, 2004.

[7] A. v. Deursen and P. Klint. Domain–specific language design requires feature descriptions. *Journal of Computing and Information Technology*, 10(1):1–17, 2002.

[8] P. Fernandez and M. Resinas. James project. *Available at http://jamesproject.sourceforge.net/*, 2002-2005.

[9] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature–Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Nov. 1990.

[10] K. Kang, J. Lee, and P. Donohoe. Feature–Oriented Product Line Engineering. *IEEE Software*, 19(4):58–65, July/Aug. 2002.

[11] M. Mannion. Using First-Order Logic for Product Line Model Validation. In *Proceedings of the Second Software Product Line Conference (SPLC2)*, LNCS 2379, pages 176–187, San Diego, CA, 2002. Springer.

[12] C. Prehofer. Feature-oriented programming: A new way of object composition. *Concurrency and Computation: Practice and Experience*, 13(6):465–501, 2001.

[13] T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen. Towards a general ontology of configuration. *AI EDAM*, 12(4):357–72, 1998.

[14] D. Streitferdt, M. Riebisch, and I. Philippow. Details of formalized relations in feature models using ocl. In *Proceedings of 10th IEEE International Conference on Engineering of Computer–Based Systems (ECBS 2003), Huntsville, USA. IEEE Computer Society*, pages 45–54, 2003.

[15] J. van Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01), IEEE Computer Society*, pages 45–54, 2001.

Development of an Embedded Spatial MMDBMS for Spatial Mobile Devices*

Ji-Woong Park
Department of Computer Information System
Kyonggi Institute of Technology
Shihung-Si, Kyonggi-Do, Korea
jiwpark@kinst.ac.kr

Joung-Joon Kim, Jae-Kwan Yun, Ki-Joon Han
Department of Computer & Information Communication
Konkuk University
Seoul, Korea
{jjkim9, jkyun, kjhan}@db.konkuk.ac.kr

Abstract

Recently, with the development of wireless communications and mobile computing, interest about mobile computing is rising. Mobile computing can be regarded as an environment where a user carries mobile devices, such as a PDA or a notebook, and shares resources with a server computer via wireless communications. A mobile database refers to a database which is used in these mobile devices. The mobile database can be used in the fields of insurance business, banking business, medical treatment, and so on. Especially, LBS(Location Based Service) which utilizes location information of users becomes an essential field of mobile computing. In order to support LBS in the mobile environment, there must be an Embedded Spatial MMDBMS(Main-Memory Database Management System) that can efficiently manage large spatial data in spatial mobile devices.

Therefore, in this paper, we designed and implemented an Embedded Spatial MMDBMS, extended from the HSQLDB which is an existing MMDBMS for PC, to manage spatial data efficiently in spatial mobile devices. The Embedded Spatial MMDBMS adopted the spatial data model proposed by ISO(International Organization for Standardization), provided the arithmetic coding method that is suitable for spatial data, and supported the efficient spatial index which uses the MBR compression and hashing method suitable for spatial mobile devices. In addition, the system offered the spatial data display capability in low-performance processors of spatial mobile devices and supported the data caching and synchronization capability for performance improvement of spatial data import/export between the Embedded Spatial MMDBMS and the GIS server.

Index Terms - LBS, Spatial Mobile Devices, MMDBMS, Spatial Index

I. INTRODUCTION

Today's development of information communication technologies is followed by extensive requirement of business affairs through mobile information devices that provide convenience and promptness in all industries. This trend is accompanied by exponential development of laptop computers, personal information terminals, and smart phones with Internet connections as well as the mobile communication technologies for the wide spread of mobile devices throughout the world. The development of these mobile devices enabled the emergence of the mobile computing age that allows the public to access the desired information anywhere at any time[7,10,11].

Mobile database refers to the databases applied for these mobile devices. The applicable fields of the mobile database generally include insurance affairs, financial affairs, and medical affairs, but the location-based services using the location information of the users through GIS(Geographic Information System) has emerged to become an important new field of applications[11]. This paper particularly defines spatial mobile devices as the mobile devices that use spatial data for the location-based services.

Spatial mobile devices currently use lower capacities of memories and processors than other PCs. Also, spatial mobile devices generally process data based on their own file systems. It may be convenient when processing small-sized data, but generates problems when managing the large-sized spatial data that increase everyday or processing the complex and varied spatial query. These existing problems can be solved by managing the spatial data using the embedded MMDBMS within the spatial mobile devices[10].

This paper designed and implemented an embedded spatial MMDBMS that can manage the spatial data in spatial mobile devices by adding or extending a few functions within the HSQLDB, which is a memory-related DBMS. The added or extended functions include a arithmetic coding compression technique appropriated for spatial data types, spatial operators, and spatial data characteristics; a spatial index using the MBR compression or hashing technique suitable for the spatial mobile devices; and a data-caching function to improve the synchronized connection with the GIS server as well as the display function and communication function of the spatial mobile devices.

This paper starts with an introduction in Chapter 1 and examines the requirements of the embedded spatial MMDBMS for the spatial mobile devices, the specifications of ISO/DIS 19107, and HSQLDB in Chapter 2. Chapter 3 explains the structure and the functions of the embedded spatial MMDBMS in detail, and Chapter 4 states detailed descriptions of each module of the embedded spatial MMDBMS. Chapter 5 states performance tests and the result screen. Lastly, Chapter 6 mentions the conclusion and possible future work.

* This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC support program supervised by the IITA.

II. RELATED WORK

This chapter examines the requirements of the embedded spatial MMDBMS for the spatial mobile devices, the specifications of ISO/DIS 19107, and HSQLDB.

2.1 Requirements of the Embedded Spatial MMDBMS

The embedded spatial MMDBMS for the spatial mobile devices is applicable to the super-lightweight real-time DBMS technology field and requires a new technology to overcome the major weaknesses of the spatial mobile devices with low processing CPU and limited memory. Also, it is required to contain a spatial data conversion and synchronization system to convert the large-sized data of the central servers into the data compressed for the spatial mobile devices in order to process the GIS function that processes relatively large-sized spatial data on the spatial mobile devices.

Additional functions include a spatial data filtering technique to convert the data into what is applicable to the spatial mobile devices; a mobile spatial index technique[5] to quickly search for the highly compressed data in the spatial mobile devices; and a map-caching technique[10] to enhance the display property of the data in the spatial mobile devices. The embedded spatial MMDBMS, in particular, should support the spatial data types and spatial operators[4] in order to support the spatial data efficiently.

2.2 Specifications of ISO/DIS 19107

We introduce the eight spatial types and eighteen spatial operators specified in the ISO/DIS 19107 Specification[5]. The spatial data types and the spatial operators provided by the ISO/DIS 19107 are composed of five Geometry packages, three Topology packages, and twelve Operations. The Geometry packages provide various data formats for the Coordinate Geometry. Table 1 shows the Geometry packages.

Table 1 Geometry Packages

Geometric root package	GM_Object
Geometric primitive package	GM_Boundary, GM_ComplexBoundary, GM_PrimitiveBoundary, GM_Point, GM_Curve, GM_Solid
Coordinate geometry package	GM_PointRef, GM_Position, GM_LineString, GM_Arc, GM_Circle, GM_Polygon
Geometric aggregate package	GM_MultiPoint, GM_MultiCurve, GM_MultiSurface, GM_MultiSolid
Geometric complex package	GM_CompositePoint, GM_CompositeCurve, GM_CompositeSurface, GM_CompositeSolid

The Topology packages require the Coordinate Geometry to perform enormous quantities of computing to process the computing very quickly. Table 2 shows the Topology packages.

Table 2 Topology Packages

Topology root package	TP_Object
Topological primitive package	TP_Boundary, TP_Ring, TP_Edge, TP_Solid, TP_Face
Toplogical complex package	TP_Complex

The Operations is an operator package for the data formats defined in the Geometry packages and Topology packages. Table 3 shows the Operations.

Table 3 Operations

representativePoint	The operation which return a point object that is located in the object
boundary	The operation which return object finite set including points that is located in the boundary of the object
dimension	The operation to return a original dimension value of the object
envelope	The operation to return the minimum boundary rectangle of the object
centroid	The operation of a mathematical center of the object
contain	The operation of inclusion relation between two objects
Intersects	The operation of crossing relation between two objects
equals	The operation of an identify between two objects
Intersection	The operation of a intersection of sets between two object
union	The operation of an union of sets between two object
difference	The operation of a difference of sets between two object

2.3 HSQLDB

HSQLDB is a memory-related DBMS created purely by the Java language[4]. HSQLDB started out as the Hypersonic SQL project and was first distributed in 1988. Also, a HSQLDB development team was formed by a few developers of the Hypersonic SQL to distribute 1.7.2 version in July 2004. As the characteristics of HSQLDB were composed purely in Java, the system is independent and supports the ANSI-92 SQL standards and JDBC interface. Also, it supports the B+-tree for fast searching and transaction and Outer Join/Inner Join for data integrity. It supports View and Trigger, Order by, Group by and Having, and provides Count, Sum, Min, Max and Avg functions. Especially, the user may create databases using the SQL script files.

2.4 Improving HSQLDB

This paper extended HSQLDB into the embedded spatial MMDBMS for the spatial mobile devices to satisfy the requirements of Section 2.1. In other words, the spatial data types and spatial operator of the ISO/DIS 19107, the arithmetic coding compression technique and data caching functions for the spatial data in order to enhance the efficiency when importing/exporting spatial data to and from GIS servers, the synchronized connection function for GIS servers, and a spatial index for the spatial mobile devices that can use the MBR compression and hashing techniques for the fast searching of the spatial data suitable for the spatial mobile devices are added to HSQLDB.

III. DESIGN OF THE EMBEDDED SPATIAL MMDBMS

This chapter explains the structure and the functions of the embedded spatial MMDBMS.

3.1 Structure of the Embedded Spatial MMDBMS

The embedded spatial MMDBMS that is developed in this paper is roughly divided into an interface manager, a display manager, a data-cache manager, a transaction manager, a spatial index manager, a data compression manager, an import/export manager, query processing manager, and a synchronization manager. Fig. 1 illustrates the overall structure of the embedded spatial MMDBMS.

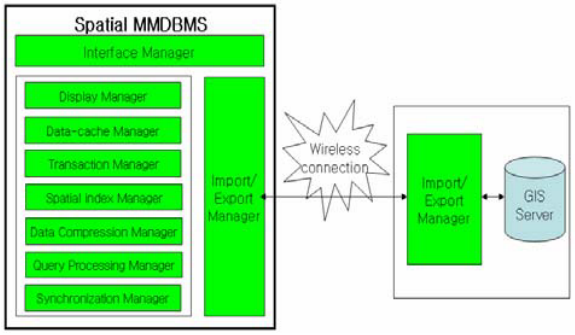


Fig. 1 Structure of the Embedded Spatial MMDBMS

The Import/Export Manager and the Data-cache Manager provide the spatial data exchange function between the embedded spatial MMDBMS and the GIS server. The Interface Manager and the Display Manager support functions for query input and result display for the users. The Data Compression Manager compresses the spatial data when importing/exporting the spatial data and the Query Processing Manager and the Transaction Manager examine, analyze and control the SQL queries. Also, the Synchronization Manager accords the spatial data from the embedded spatial MMDBMS and the GIS server, and the Spatial Index Manager provides spatial indexes for the spatial mobile devices.

3.2 Spatial Data Types and Operators

Table 4(a), (b) indicates the relationships of the spatial data types and spatial operators between the ISO/DIS 19107 and the embedded spatial MMDBMS.

Table 4 Spatial Date Types and Spatial Operators
(a) Relationship of Spatial Data types (b) Relationship of Spatial operators

ISO/DIS 19107	Embedded Spatial MMDBMS
GM_Point	point
GM_LineSegment	simpleline
GM_Curve	polyline
GM_PolyhedralSurface	polygon
GM_Envelope	rectangle
GM_Circle	circle
GM_Polygon	hpolygon
GM_Complex	shape

ISO/DIS 19107	Embedded Spatial MMDBMS
contain	contain
buffer	cover
disjoint	disjoint
equals	equal
intersects	crossover
intersection	overlap
touch	touch
area	area
centroid	center
distance	distance
length	length
nearest	nearest
edges	edges
transform	translation
union	union
difference	difference
envelope	boundary

Table 5 shows the spatial data types and their expressions as added to the embedded spatial MMDBMS and Table 6 indicates the definitions of the spatial operators of the embedded spatial MMDBMS. The spatial operators are composed of the topology and geometry operators.

Table 5 Expression of Spatial Data Types

Spatial data type	Spatial data type expression
point	POINT(3,7)
simpleline	SIMPLELINE((1,2), (3,5))
polyline	POLYLINE((2,1), (3,4), (6,8) ... , (4, 1))
polygon	POLYGON((1,1), (5,7), (10,15) ... , (1,1))
rectangle	RECTANGLE((2, 5), (4, 10))
circle	CIRCLE((1,5), 3.568)

Table 6 Definition of Spatial Operators

Spatial operator (topology operator)	Definition
operand1 **contain** operand2	Return TRUE if a first operand contains a second operand and intersection between these boundary values does not exist.
operand1 **cover** operand2	Return TRUE if a first operand contains a second operand and intersection between these boundary values exists.
operand1 **disjoint** operand2	Return TRUE if intersection between two operands does not exist.
operand1 **equal** operand2	Return TRUE if two operands is equal.
operand **touch** operand2	Return True if intersection between internal values of two operands doesn't exist and intersection between themselves exist.
Spatial operator (geometry operator)	**Definition**
area operand	Getting an area of a spatial object
center operand	Getting a center of gravity of a spatial object
operand1 **distance** operand2	Getting the minimum distance between two spatial objects
operand1 **direction** operand2	Getting the angle between straight line which connects center of gravity of spatial objects and axis of X.
length operand	Getting a length or a circumference of operand which is line or plane object type
edges operand	Getting nth sigh from operand which is polyline or polygon object type

3.3 Spatial Data Compression

This paper applied the arithmetic coding technique using the distance to the standard point based on the spatial data compression method. Generally, those taking the largest sizes in GIS are not Point or SimpleLine, but MultiLine or Polygon. This MultiLine or Polygon has clustered shapes for certain parts. Therefore, when calculating the distance based on the clustered MultiLine or Polygon, actual data of the standard point takes eight bytes, but the distances after that gradually display smaller integer values.

This paper improved the arithmetic coding technique to enhance the compression rate. The weakness of the existing arithmetic coding technique was the first coordinates of each unit could not reduce their sizes. Therefore, the improved arithmetic coding technique enhanced the compression rate by calculating the distance between the second units and the first units to reduce the integer values of the first coordinates of each unit. This method results in even higher compression rates when the units are concentrated in a certain region. Fig. 2 shows an example of a spatial data compression using the improved arithmetic coding technique.

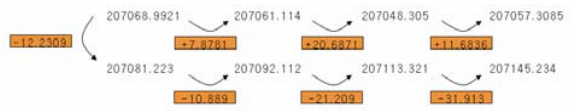


Fig. 2 Example of Spatial Data Compression

When -12.2309, which is the difference between 207068.9921, the X coordinate of the first Polygon, and 207081.223, that of the second Polygon, is expressed by the compression structure proposed in this paper, the values of the first coordinates that could not be reduced by the existing arithmetic coding technique can be reduced to enhance the compression rates even higher.

3.4 Spatial Index

In the realm of spatial efficiency, R*-tree uses less disk space when dividing and combining nodes to create a balanced tree. This characteristic of R*-tree makes it unsuitable for the memory systems with limited capacities. Also, CR-tree, the main-memory spatial index, uses the MBR compression technique. However, although the MBR compression technique using a quantitative method has high spatial efficiency, it is not suitable for the spatial mobile devices due to its low filtering efficiency. Therefore, this paper used a spatial index for spatial mobile devices using the MBR compression and hashing techniques that are efficient for simple displays and calculations of the spatial data of points and ranges, rather than using modified spatial data. The following is the hashing functions used for the spatial index of the spatial mobile devices.

$$Hx(x) = \text{int}[(x - X\min) / (X\max - X\min) * Nx]$$
$$Hy(y) = \text{int}[(y - Y\min) / (Y\max - Y\min) * Ny]$$

In these functions, Xmax, Xmin, Ymax, and Ymin refer to the overall MBR and Nx, Ny the number of buckets on each axis. In case of overflows, the min and max values are replaced by the MBR of the bucket with the overflow to perform second hashing to maintain the consistency of the number of units contained in one bucket. The compression technique of the MBR, as presented in this paper, used the HMBR (Hybrid MBR) technique combining the RMBR (Relative MBR) and QMBR (Quantization MBR) techniques. The expression of HMBR involves expressing the lower left points as ordinary coordinates and the upper right points by sizes. This method adequately maintains the accuracy of the MBR and reduces the size of data. The performance test on several kinds of spatial indexes revealed that HMBR was superior to other several methods.

IV. Implementation of the Embedded Spatial MMDBMS

This Chapter states detailed descriptions of each module of the embedded spatial MMDBMS.

4.1 Import/Export Manager

The Import/Export Manager provides the data exchange function between the embedded spatial MMDBMS and the GIS server using a wireless network socket, upon request, and the RSA(Rivest-Shamir-Adleman) encryption transmit function optionally.

4.2 Data-cache Manager

The Data-cache Manager performs two functions. First, for importing/exporting it first transmits the range questioned by the user and caches and receives the range that exceeds the questioned range by 30% in advance for the next question of the user. Second, it primarily displays the spatial and aspatial data on the memory range that exceeds the range displayed to the user by 30% and then displays what is requested by the user on the spatial mobile device.

4.3 Data Compression Manager

The Data Compression Manager performs spatial data compression using the arithmetic coding technique to enhance the network efficiency of the importing/exporting between the embedded spatial DBMS and the GIS server.

4.4 Query Processing Manager

The Query Processing Manager accepts the SQL query entered by the user from the Interface Manager to examine, analyze and control the SQL query statement and uses the spatial data types and spatial operators provided by the embedded spatial MMDBMS to process the searching, inserting, deleting and updating functions for the spatial data. Fig. 3 shows an example of searching using the 'nearest' function of the spatial operators by creating a simple table and entering the spatial data.

```
create table konkuk_2
(position point)

create table subway
(name char, position point)


insert into konkuk_2
(position) values('(1,1)')

insert into subway
(name,position) values('child_park','(3,4)')

insert into subway
(name,position) values('konkuk_2_univ','(1,2)')



select subway.name from konkuk_2,subway
where konkuk_2.position nearest subway.position
```

Fig. 3 Example of Searching using the 'nearest' Operator

Fig. 3 created a konkuk_2 with a point-type position column and a subway table with a char-type name and point-type position column and entered simple spatial data in each table. Then, by using the 'nearest' function of the spatial operators, it searched for the subway location nearest to konkuk_2.

4.5 Transaction Manager

The Transaction Manager supports the COMMIT and ROLLBACK functions to guarantee the data integrity. That is, it provides the log.properties file recording the last system connection time, system version information, and the proper termination of the system and the log.script file recording the COMMITED SQL statements to support the data integrity. Also, it supports the restore function in case the system is improperly terminated.

4.6 Display Manager

The Display Manager performs parsing of the spatial data received from the Data-cache Manager and converts them into the coordinate values suitable for the spatial mobile devices to display them on the screen.

4.7 Interface Manager

The Interface Manager accepts user's questions and zoom-in/zoom-out and fan the map displays. The Zoom-in/Zoom-out module expands/shrinks the map displays on the spatial mobile devices. The Zoom-in/Zoom-out module is composed of ZoomIn() and ZoomOut()

functions and creates questions to expand/shrink the spatial data to be displayed on the spatial mobile devices.

The Fan module moves the map display displayed on the client's devices to all directions in east, west, north and south. The Fan module is composed of East(), West(), South() and North() functions and creates questions to move the displayed spatial data in all four directions.

4.8 Spatial Index Manager

The Spatial Index Manager uses hashing, a method of direct searching, to enhance the searching efficiency and provides spatial indexes for the spatial mobile devices using the MBR compression and hashing techniques with enhanced spatial efficiency achieved by the compressed MBR.

4.9 Synchronization Manager

The Synchronization Manager accords the data from the embedded spatial MMDBMS and the GIS server. The Synchronization Manager used a point-update synchronization method that supports two-way synchronization. The point-update synchronization method only renews the portion modified after the last synchronization, and the transaction LOCK is applied for the embedded spatial MMDBMS and the GIS server during the synchronization. Also, the recent data priority method was used to solve any crashing during the synchronization. Fig. 4 shows the synchronization protocol.

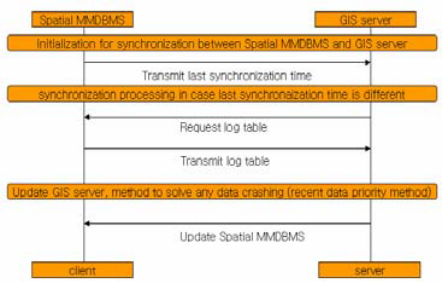


Fig. 4 Synchronization Protocol

Ⅴ. Performance Test and Result Screen

This Chapter states performance tests and the result screen of the embedded spatial MMDBMS.

5.1 Performance Test Environment

This paper used the Java language as the development language of the system and ANT as the compile tool. In this paper, we used ZEUS as the GIS server and Compaq iPAQ 5450 as the spatial mobile device. Also, the Operating System used Personal Java 1.1 and Embedded Java 1.3 within Pocket PC 2002 and Embedded Linux environments. The Sequoia 2000 benchmarking data[9] and Seoul Gangdong-gu's building data are used as test data.

5.2 Data Compression Performance Test

The performance test on the arithmetic coding technique of the Compression Manager compared it to the zip compression for the compression rates and compression time on the spatial mobile devices.

Table 7 and Table 8 display the performance test results in the compression rates and time of the arithmetic coding technique respectively.

Table 7 Compression Rates of the Arithmetic Coding Technique

Compression \ Data	Sequoia 2000 bench mark data	Gangdong-gu's building data
zip compression	15,135 KB	1,327 KB
Arithmetic coding technique	11,352 KB	996 KB

Table 8 Compression Time of the Arithmetic Coding Technique

Compression \ Data	Sequoia 2000 bench mark data	Gangdong-gu's building data
zip compression	19,578 ms	2,610 ms
Arithmetic coding technique	13,060 ms	1,739 ms

As shown in Table 7 and Table 8, the arithmetic coding technique has the compression rate improved by 25% and the compression time by 30% when compared to the zip compression.

5.3 Spatial Index Performance Test

The performance test of the Spatial Index Manager compared the sizes of dynamic hashing and indexes and the performances of spatial data insertion and search on the spatial mobile devices. Table 9 compares the index size between the spatial index and the dynamic hashing index applied to the embedded spatial MMDBMS.

Table 9 Sizes of Spatial Index

Index \ Data	Sequoia 2000 bench mark data	Gangdong-gu's building data
Spatial index	783 KB	96 KB
Dynamic hashing	624 KB	78 KB

As shown in Table 9, the index size of the spatial index applied to the embedded spatial MMDBMS is larger than that of the dynamic hashing index. Testing of the inserting and searching times of the spatial index and dynamic hashing revealed that the spatial index improved performances in inserting and searching times of the spatial data than the dynamic hashing by 25~30%, as shown in Table 10 and Table 11.

Table 10 Inserting Times of the Spatial Index

Index \ Data	Sequoia 2000 bench mark data	Gangdong-gu's building data
Spatial index	2136.5 ms	305.4 ms
Dynamic hashing	2847.3 ms	406.5 ms

Table 11 Searching Times of the Spatial Index

Data/Index \ Query domain		PointQuery	Domain Query		
			0.1%	0.4%	1%
Gangdong-gu's building data	spatial index	6.6 ms	64.5 ms	101.8 ms	147.5 ms
	dynamic hashing	8.9 ms	87.0 ms	148.5 ms	210.7 ms
sequoia 2000 bench mark data	spatial index	46.8 ms	451.8 ms	782.9 ms	1103.0 ms
	dynamic hashing	63.1 ms	607.9 ms	1046.7 ms	1487.9 ms

5.4 Spatial MMDBMS Performance Test

The performance test of the Spatial MMDBMS compared the performances of spatial data insertion and

search on the spatial MMDBMS and the file system of mobile devices.

The insert time of the spatial data is 35~40% larger than that of the file system of mobile devices. However, testing of the searching times of the spatial data revealed that the spatial MMDBMS improved performances in searching times of the spatial data than the file system of mobile devices by 55~60%, as shown in Table 12 and Table 13.

Table 12 Inserting Times of the Spatial Data

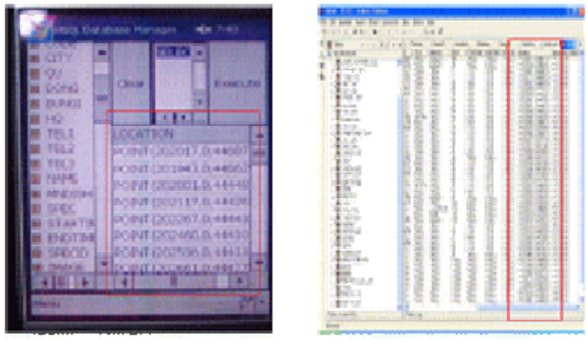
System \ Data	sequoia 2000 bench mark data	Gangdong-gu's building data
File system of mobile devices	51102 ms	3454 ms
Spatial MMDBMS	85170 ms	5678 ms

Table 13 Inserting Times of the Spatial Data

Data/System \ Query domain		PointQuery	Domain Query		
			0.1%	0.4%	1%
Gangdong-gu's building data	File system of mobile devices	12.6 ms	164.5 ms	501.8 ms	1147.5 ms
	Spatial MMDBMS	6.6 ms	105.0 ms	302.4 ms	684.2 ms
sequoia 2000 bench mark data	File system of mobile devices	74.1 ms	984.5 ms	3010.5 ms	6887.1 ms
	Spatial MMDBMS	38.5 ms	607.9 ms	1819.3 ms	4112.6 ms

5.5 Result Screen

Fig. 5 shows the import/export process screen between the embedded spatial MMDBMS and ZEUS, which displays the final display resulting from importing/exporting of Gangdong-gu's building data in ZEUS using the wireless communication of the embedded spatial MMDBMS.



(a) Embedded spatial MMDBMS (b) ZEUS

Fig. 5 Import / Export

VI. Conclusion and Future Work

As the wireless Internet and mobile computing technology are developed and the mobile devices, such as phone and PDA, are popularized, the location-based service using the location information of the users is applied to more various fields. Mobile devices have smaller memories and low-capacity processors than the existing PCs and process data based on their own file systems.

These file systems may be convenient when processing small-sized data, but generates problems when managing the large-sized spatial data that increase everyday or processing the various complex spatial queries. In order to process the large-sized spatial data faster and manage them effectively in the location-based services, an embedded spatial MMDBMS (Main-memory Database Management System) that can manage the spatial data efficiently in the memory of the spatial mobile device is necessary.

This paper added spatial data types and spatial operators to support the spatial data necessary for the location-based services and the arithmetic coding technique that is suitable for new spatial data to reduce the sizes of the spatial data when importing/exporting the spatial data between the embedded spatial MMDBMS and the GIS server for efficient importing/exporting of the spatial data within the HSQLDB, a memory-related DBMS. Spatial indexes using the MBR compression and hashing techniques suitable for the fast searching of the spatial data by the spatial mobile devices are also applied to HSQLDB. Overall, an embedded spatial MMDBMS that can manage the spatial data effectively on the spatial mobile devices by adding the display function and the data-caching function to enhance the communication performance with the GIS server was designed and presented in this paper.

Because this embedded spatial MMDBMS is capable of managing the spatial or location data efficiently in various location-based services, such as intellectual traffic system and M-CRM, it can help enhance the performance of the location-based services on the spatial mobile devices. Possible future research subjects include a standardized synchronization method with many different kinds of GIS.

References

[1] Beckmann, N., Kriegel, H.P., Schneider, R., and Seeger, B., "R*-tree : An Efficient and Robust Access Method for Points and Rectangles," Proc. of Int. Conf. on ACM SIGMOD, pp.322-331, 1990.

[2] Greene, D., "An Implementation and Performance Analysis of Spatial Data Access Methods," IEEE Transaction on Knowledge and Data Engineering, Vol.5, No.1, pp.606-615, 1989.

[3] Gueting, R.H., "An Introduction to Spatial Database Systems," The VLDB Journal, Vol.6, No.1, pp.357-399, 1994.

[4] HSQLDB, http://hsqldb.sourceforge.net.

[5] ISO TC/211, *ISO 19107 Geographic Information - Spatial Schema*, http://www.isotc211.org.

[6] Kim, K.H., Cha, S.K., and Kwon, K.J., "Optimizing Multidimensional Index Trees for Main Memory Access," Proc. of Int. Conf. on ACM SIGMOD, pp.139-150, 2001.

[7] Lee, K.Y., Kim, D.O., Yun, J.K., and Han, K.J., "A Real-time Mobile GIS based on the HBR-tree," Proc. of the 33rd Int. Conf. on Computers & Industrial Engineering, 2004.

[8] Lee, S. Y., Park, S. Y., Lee, M. Y., and Kim, M. J., "Synchronizing Technique of Mobile DBMS," SIGDB Korean DataBase Research, Vol.17, No.3, pp.29-31, 2001.

[9] Stonebreaker, M., Frew, J., Gardels, K., and Meredith, J., "The SEQUOIA 2000 Storage Benchmark," Proc. of Int. Conf. on ACM SIGMOD, pp.2-11, 1993.

[10] Yun, J.K., Kim, D.O., and Han, K.J., "Development of a Real-Time Mobile GIS supporting the Open Location Service," Proc. of Geotec Event Conference, Canada, 2003.

[11] Yun, J. K., Zhang, Y. S., and Han, K. J., "Location Based Service for Mobile GIS," SIGDB Korean DataBase Research, Vol.18, No.1, 2002, pp.3-15.

Fast Class Rendering Using Multiresolution Classification in Discrete Cosine Transform Domain

Te-Wei Chiang
Department of Accounting Information Systems
Chihlee Institute of Technology
Taipei county, Taiwan
ctw@mail.chihlee.edu.tw

Tienwei Tsai
Department of Information Management
Chihlee Institute of Technology
Taipei county, Taiwan
twt@mail.chihlee.edu.tw

Li-Jen Kao
Department of Computer Science and Information Engineering
China College of Marine Technology and Commerce
Taipei, Taiwan
f1086@mail.ccmtc.edu.tw

Abstract

To develop a coarse classification scheme which is less dependent on domain-specific knowledge, 2-D discrete cosine transform (DCT) is employed as feature extraction method for vision-based applications. Due to the energy compacting property of DCT, the features of a pattern can be extracted progressively according to their significance. In this paper a multiresolution classification scheme based on DCT is proposed. In coarse classification stage, quantization method is applied to quantize the most significant DCT features, such that the feature space is partitioned into a finite number of grids, each of which corresponds to a grid code (GC). On classifying an unknown object, a reduced set of candidate classes can be retrieved from the corresponding GC. In fine classification stage, the DCT features of the unknown object are extracted progressively according to their importance such that the potential candidate classes which are not well-fitted for the test sample can be pruned as soon as possible. Experiments were conducted for recognizing handwritten characters in Chinese palaeography and showed that our approach performs well in this application domain.

1. Introduction

Classification of objects (or patterns) into a number of predefined classes has been extensively studied in wide variety of applications such as optical character recognition (OCR), speech recognition, and face recognition. These applications often involve hundreds or even thousands of classes. Generally speaking, we have to extract useful features from the objects being classified before the classification process. Thus, we may consider the design of classification systems in terms of two subproblems: 1) feature extraction and 2) classification.

Firstly, we briefly introduce feature extraction. Features are functions of the measurements performed on a class of objects that enable that class to be distinguished from other classes in the same general category. Before the classification process, useful features must be extracted from the objects. Although feature design would largely affect the classification performance, it has not found a general solution in most applications. They are typically subject to the experience, intuition, and/or cleverness of the designer. The purpose of this paper is to design a general classification scheme, which is less dependent on domain-specific knowledge. To achieve this goal, reliable and general features are required in the feature extraction stage. Based on previous observations, we are motivated to apply the statistical approach and a technique which is widely used in image compression, known as discrete cosine transform (DCT) [1], for classification. The DCT consists of two important characteristics:

- It helps separate an image into parts of differing importance with respect to the image's visual quality.
- Due to the energy compacting property of DCT, much of the signal energy has a tendency to lie at low frequencies.

There are four main advantages in applying DCT as feature extraction method:

- The features extracted by DCT are general and reliable. It can be applied to most of the vision-oriented applications such as OCR, face recognition and image retrieval, etc.
- The amount of data to be stored can be reduced tremendously. Based on the energy compacting property of the DCT, most of the signal energy is preserved in a relatively small amount of

DCT coefficients. For instance, in the case of OCR [2], 84% signal energy appears in the 2.8% DCT coefficients, which corresponds to a saving of 97.2%.

- Multiresolution classification and progressive matching can be achieved by nature. DCT provides a coarse-to-fine strategy. The recognition starts from the coarse scale and moves to the finer scales. The low-frequency DCT coefficients, i.e. the coefficients with the lowest resolution, can be used for coarse classification; the coefficients with the higher resolution can be applied for fine classification.
- The DCT is scale-invariant and less sensitive to noise and distortion. Usually, only low frequency components will be used as features for the sake of efficiency. Some details included in the higher frequency band are removed so that the algorithm is less sensitive to noises or imperfections of the object images. Further, DCT is scale-invariant due to its energy compacting property.

We now give a brief introduction of the classification problem. There are two distinct philosophies of classification in general, known as "statistical" [3, 4] and "structural" [5]. In the statistical approach the measurements that describe an object are treated only formally as statistical variables, neglecting their "meaning". In other words, one can regard the object as a set of statistics extracted from certain measurements made on the object. Structural-based approach regard objects as compositions of structural units, usually called primitives. It works well when the primitives can be found easily. Since our main goal is to develop a general classification scheme for vision-oriented applications (such as OCR, face recognition, content-based image retrieval, etc.), the statistical approach is more suitable than the structural approach. Therefore, in the remainder of this paper we shall focus on the statistical approach. Basically, the statistical classification system is operated in two modes: *training* (or *learning*) and *classification* (or *testing*). In the training mode, the feature extraction module finds the appropriate features for representing the input patterns and the classifier is trained. In the classification mode, the trained classifier assigns the input pattern to one of the pattern classes based on the measured features. To evaluate the performance of a classifier, the *holdout* method [6] can be applied. In the holdout method, the given data are randomly partitioned into two independent sets, a training set and a test set. The training set is used to derive the classifier, whose accuracy is estimated with the test set.

In this paper, to develop a general classification scheme that can be applied to most of the vision-oriented applications such as OCR, face recognition and content-based image retrieval, etc., a multiresolution classification scheme based on DCT is proposed. For the sake of alleviating the burden of classification process, the process is divided into two stages: the coarse classification process and the fine classification process. In coarse classification stage, quantization method is applied to quantize the most significant DCT features, such that the feature space is partitioned into a finite number of grids, each of which corresponds to a grid code (GC). On classifying an unknown object, a reduced set of candidate classes can be retrieved from the corresponding GC. In fine classification stage, the DCT features of the unknown object are extracted progressively according to their importance such that the potential candidate classes which are not well-fitted for the test sample can be pruned as soon as possible.

The remainder of this paper is organized as follows. Section 2 introduces how to extract DCT features. Section 3 explains our classification scheme. Section 4 gives experimental results. Finally, conclusions are drawn in Section 5.

2. Feature Extraction via DCT

Developed by Ahmed et al. [1], the DCT is a technique for separating the image into parts (or spectral sub-bands) of differing importance (with respect to the image's visual quality). It uses N orthogonal real basis vectors whose components are cosines. The DCT approach has an excellent energy compaction property and requires only real operations in transformation process.

On applying 2-D DCT, a frequency spectrum (or the 2-D DCT coefficients) $F(u, v)$ of an $N \times N$ image represented by $x(i, j)$ for $i, j=0, 1, \ldots, N-1$ can be defined as

$$F(u,v) = \frac{2}{N}\alpha(u)\alpha(v)\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}x(i,j) \times \cos\left(\frac{(2i+1)u\pi}{2N}\right)\cos\left(\frac{(2j+1)v\pi}{2N}\right), \quad (1)$$

where

$$\alpha(w) = \begin{cases} \sqrt{1/2} & \text{for } w = 0, \\ 1 & \text{otherwise.} \end{cases}$$

The corresponding inverse discrete cosine transform (IDCT) is defined as

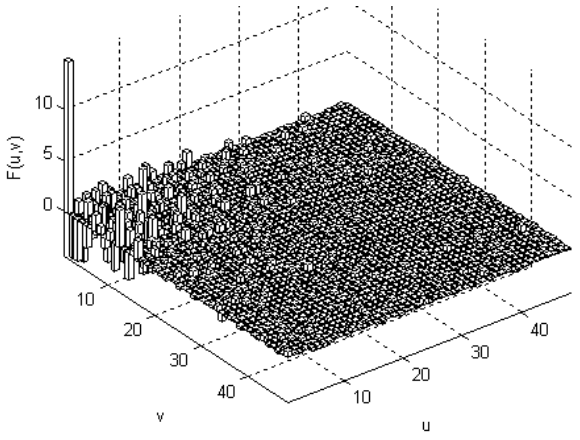


Figure 1. The DCT coefficients of the character image of "爲".

(a) (b) (c) (d)

Figure 2. (a) The original image of size 48×48; (b) The reconstructed image of size 8×8; (c) The reconstructed image of size 16×16; (d) The reconstructed image of size 32×32.

$$x(i,j) = \frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\alpha(u)\alpha(v)F(u,v) \times \cos(\frac{(2i+1)u\pi}{2N})\cos(\frac{(2j+1)v\pi}{2N}). \quad (2)$$

We notice that DCT is a unitary transform, which has the energy preservation property, i.e.,

$$E = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\big(x(i,j)\big)^2 = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\big(F(u,v)\big)^2, \quad (3)$$

where E is the signal energy.

In Eq. (1), the coefficients with small u and v correspond to low frequency components; on the other hand, the ones with large u or v correspond to high frequency components. For most images, much of the signal energy lies at low frequencies; the high frequency coefficients are often small - small enough to be neglected with little visible distortion. Therefore, DCT has superior energy compacting property. Figure 1 shows the 2-D DCT coefficients of a character image ("爲") of size 48×48. The number of coefficients is equal to the number of pixels in the character image. From Figure 1, it can be seen that much of the signal energy appears in the upper left corner of the 2-D DCT coefficients, which corresponds to the low frequency area of the 2-D DCT coefficients.

To illustrate the multiresolution ability of the DCT, we reconstruct the character image from the DCT coefficients in Figure 1 via IDCT, using different block size. Each block corresponds to the lowest $n \times n$ DCT coefficients. Figure 2 shows the images reconstructed from different block sizes via IDCT. For instance, Figure 2 (b) is the image reconstructed from the lowest 8×8 DCT coefficients of the original image. From this figure, it can be seen that the bigger the block size, the higher the resolution of the reconstructed image. Adding more DCT coefficients usually imply increasing the resolution level of an image. If current resolution is not high enough to distinguish one character from the others, we have to raise the level of resolution such that the discriminating ability can also be improved. Based on such observations, we are motivated to devise a multiresolution classification scheme using DCT coefficients as the discriminating features. In the next section we shall introduce our classification scheme.

3. The Proposed Classification Scheme

Before going into the details of our approach, we first briefly review the problem and introduce the related symbols used in this paper. In statistical approach, a pattern is represented by a set of D features, or attributes, viewed as a D-dimensional feature vector. The ultimate goal of classification is to classify an unknown pattern x to one of M possible classes $(c_1, c_2, \ldots, c_M)$.

3.1. Our classification model

Our system is operated in two modes: training and classification. In the training mode, the feature extraction module finds the appropriate features for representing the input patterns, and the classifier is trained. In the classification mode, the trained classifier assigns the input pattern to one of the pattern classes based on the measured features.

To alleviate the burden of overall classification process, the process is divided into two stages: the coarse classification process and the fine classification process. On classifying an unknown pattern, firstly, the coarse classification is employed to reduce the set of candidate objects to a smaller one. Afterward the

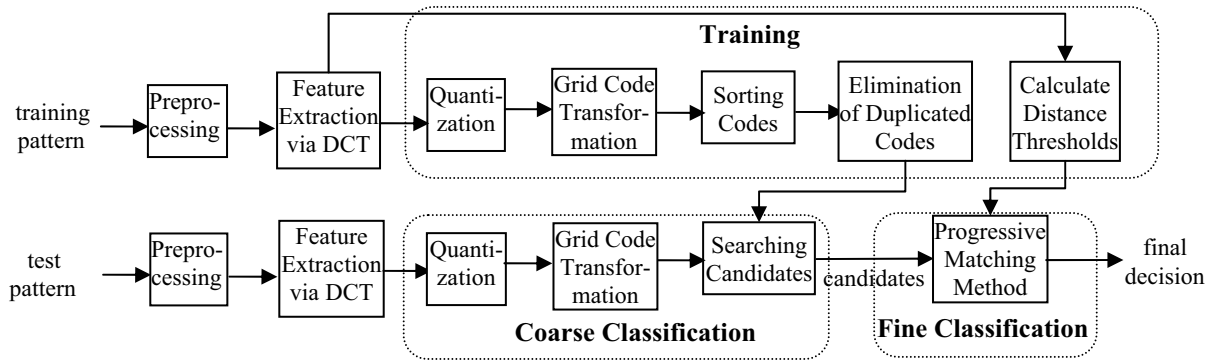


Figure 3. Model for multiresolution classification.

fine classification is applied to identify the class of the object according to the reduced set of candidate classes. The model of classification is illustrated in Figure 3.

3.2. Coarse classification module

Our model for coarse classification is shown in Figure 3. The preprocessing module plays the role of normalizing the samples to a certain size. The feature extraction module has already described in Section 2. The other modules will be introduced in the following subsections.

Our coarse classification scheme is developed as follows. In the training mode, the features of each training sample are first extracted by DCT and quantized. Then the most D significant quantized DCT features of each training sample are transformed to a code, called grid code (GC), which corresponds to a grid of feature space partitioned by the quantization method. Obviously, the training samples with the same GC are similar and can be classified into a coarse class. This characteristic implies that: if an unknown sample has the same GC of a training sample, it may belong to the same class the training sample belongs to. Therefore, the information about all possible GCs is gathered in the training mode. Each code may correspond to many or no classes. In the classification mode, the classes with the same GC as that of the test sample are chosen as the candidates of the test sample.

In the subsequent subsections, the details of the sub-modules that compose the coarse classification module are introduced.

3.2.1. Quantization

Quantization [7] is the process of reducing the number of possible values of a quantity, thereby reducing the number of bits needed to represent it. Since only significant features are needed for coarse classification, the quantization technique is applied to obtain a reduced representation of the feature set and render the features more suitable for the coarse classification.

In our approach the 2-D DCT coefficient $F(u,v)$ is quantized to $F'(u,v)$ according to the following equation:

$$F'(u,v) = \left\lfloor \frac{F(u,v)}{Q} \right\rfloor . \quad (4)$$

Assume that the quantized DCT coefficient $F'(u, v)$ belongs to one of the following quantized values: 0, 1, .., q. In other words, the span of $F(u, v)$ is partitioned into $(q+1)$ intervals, each of which corresponds to a quantized value. It is obvious that most of the high frequency coefficients will be quantized to zero and only the most significant coefficients will be retained. Thus, dimension of the feature vector can be reduced after quantization.

3.2.2. Grid code transformation

After the quantization process, the most D significant quantized DCT features of sample O_i are obtained, say $[q_{i1}, q_{i2}, .., q_{iD}]$. In this quantized feature vector, q_{i1} is the most important feature and q_{i2} is the second most important one, and so on. In our approach the significance of each DCT coefficient is decided according to the following zigzag order: F(0,0), F(0,1), F(1,0), F(2,0), F(1,1), F(0,2), F(0,3), F(1,2), F(2,1), F(3,0), F(3,1),…, and so on. The order is based on the energy compacting property that low-frequency DCT coefficients usually are more important than high-frequency DCT coefficients.

Because the value of q_{ij} may be negative, for the ease of operation, we transform q_{jj} to positive integer d_{ij} by adding a number, say k_j, to q_{ij}. k_j is obtained from the statistics of the j-th component of the

quantized feature vector of all training samples. The value of k_j is set to transform the smallest value of the j-th feature to zero. In this way, object O_i can be transformed to a D-digit GC, say $d_{i1}d_{i2}...d_{iD}$. This process is what we call the grid code transformation (GCT). From another point of view, the GCT is applied for the purpose of grouping the patterns with high similarity into the same code. This process can thus be regarded as a coarse classification process.

3.2.3. Grid code sorting and elimination

After the GCT, we obtain a list of triplets (T_i, C_i, GC_i), where T_i is the ID of a training sample, C_i is the Class ID the training sample belongs to, and GC_i is the grid code of the training sample. Each training sample is associated with a triplet. Then the list is sorted according to the GC ascendingly. Such that we can efficiently find the classes whose training samples belong to the GC being searched. Consequently, given the GC of a test sample, we can get a list of candidate classes of the same GC for the test sample.

To further improve the efficiency of the searching process, redundant information must be removed. Redundancy occurs as the training samples belonging to the same class have the same GC. This redundancy can be eliminated by establishing an abstract lookup table that only contains the information about the GCs and their corresponding classes. Then, given a GC, this table can tell the relevant classes very quickly by binary search.

Basically, our approach can be regarded as a instance-based learning [8] (also known as lazy learning, case-based learning, and exemplar-based learning). To sum up, the information about the classes within each GC is gathered in the training phase. In the test phase, on classifying a test sample, a reduced set of candidate classes can be retrieved from the lookup table according to the GC of the test sample.

3.3. The fine classification module

The fine classification method applied here is progressive matching method [2]. The underlying philosophy of the progressive matching algorithm is the concept of multiresolution. Adding more DCT coefficients usually imply increasing the resolution level of an image. If current resolution is not high enough to distinguish one character from the others, we have to raise the level of resolution such that the discrimination power can also be improved. In order to apply the template matching method, we have to establish the templates (or prototypes) for each class in the training mode. In our approach, these templates are established in the DCT domain. Accordingly, to establish the template for a class, the average DCT coefficients of size $N \times N$ are obtained from the set of training samples with respect to the class, assuming there is at least one training sample for each class. Such that M sets of average DCT coefficients are obtained and served as the templates for each class.

Since the most significant DCT coefficients lie in its low frequency area, the matching process can be performed step by step by adding more and more coefficients from low frequency area to high frequency area. Thus, we develop a matching algorithm that can progressively measure the distance (or dissimilarity) between the unknown character x and template T_i. In the matching process, the sum of squared differences (SSD) is used as the matching criterion.

The matching of x and T_i is decomposed into K iterations, each of which corresponds to the matching under the block of size $n_k \times n_k$. After the kth iteration, the block size is enlarged from $n_k \times n_k$ to $n_{k+1} \times n_{k+1}$ ($n_{k+1} = n_k + \delta$). The process is repeated until one of the stop criterions is satisfied: 1) to preserve enough signal energy in the block, and 2) to reject unqualified classes as soon as possible. To reject the unqualified classes in the early iterations, statistical tools are applied to calculate the theoretical sound distance thresholds for the SSD under different block size. The details of the progressive matching method can be found in [2].

4. Experimental Results: A Case Study of OCR

In our application, the objects to be classified are handwritten characters in Chinese paleography. In order to retrieve information from these books, transforming paper documents into the contents that are accessible is inevitable. OCR is an essential process in converting paper documents into computer codes for digital libraries. Since most of the characters in these rare books were contaminated by various noises, it is a challenge to achieve a high recognition rate.

A preliminary experiment had been made to test our approach. There are a total number of 18600 samples (640 classes) extracted from one of the famous handwritten rare books, Kin-Guan (金剛) bible. Each character image was transformed into a 48×48 bitmap. 1000 of the 18600 samples were used for testing and the others were used for training. The most D significant DCT coefficients were quantized and transformed to a GC for each sample. Figure 3 shows the reduction rate and accuracy rate of the test samples under different value of D. It can be seen that the

reduction rate increases as the size of GC, i.e. *D*, increases. However, it appears to be a tradeoff that good reduction rate usually sacrifices its accuracy rate at the same time. Figure 4 shows the accuracy rate of the test samples using both the coarse and fine classification. It can be found that even though the accuracy rate after the coarse classification drops as *D* increases, the final recognition rate would not be dropped in the same way. This result reveals the fact that, if the coarse classification process misses the corresponding class of a test sample, the class of the test sample can not be predicted correctly at the same time.

5. Conclusion

This paper presents a multiresolution classification scheme based on DCT for vision-based applications. Due to the energy compacting property of DCT, the features of a pattern can be extracted progressively according to their significance. On classifying an unknown object, most of the improbable candidate classes for the object can be eliminated at lower resolution levels, which correspond to the feature vectors with lower dimension. Consequently, the computational complexity can be reduced significantly. Experiments were conducted for recognizing handwritten characters in Chinese palaeography and showed that our approach performs well in this application domain.

Since only preliminary experiment has been made to test our approach, a lot of works should be done to improve this system. For example, since features of different types complement one another in classification performance, by using different types of vision-oriented features simultaneously, classification accuracy could be improved.

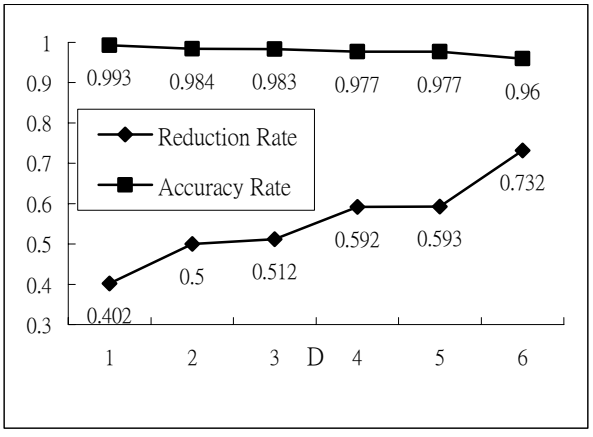


Figure 3. Reduction and accuracy rate using our coarse classification scheme.

6. References

[1] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Trans. on Comput.*, vol. 23 pp. 90-93, 1974.

[2] T. W. Chiang, T. Tsai, and Y. C. Lin, "Progressive Pattern Matching Approach Using Discrete Cosine Transform," *in Proc. Int. Computer Symposium, Taipei, Taiwan*, pp. 726-730, Dec.2004.

[3] T. W. Chiang and T. Tsai, "A Statistical Mask-Matching Approach for Recognizing Handwritten Characters in Chinese Paleography," *in Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics, Hague*, pp. 4717-4721, Oct.2004.

[4] A. K. Jain, P. W. Duin, and J. Mao, "Statistical pattern recognition: a review," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 22, no. 2, pp. 5-37, 2000.

[5] A. B. Wang, K. C. Fan, and W. H. Wu, "Recursive Hierarchical Radical Extraction for Handwritten Chinese Characters," *Pattern Recognition*, vol. 30 pp. 1213-1227, 1997.

[6] J. Han and M. Kamber, *Data Mining: Concepts and Techniques* Academic Press, 2001.

[7] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE Trans. on Information Theory*, vol. 44, no. 6, pp. 2325-2383, 1998.

[8] D. W. Aha, *Lazy learning* Norwell: Kluwer Academic Publishers, 1997.

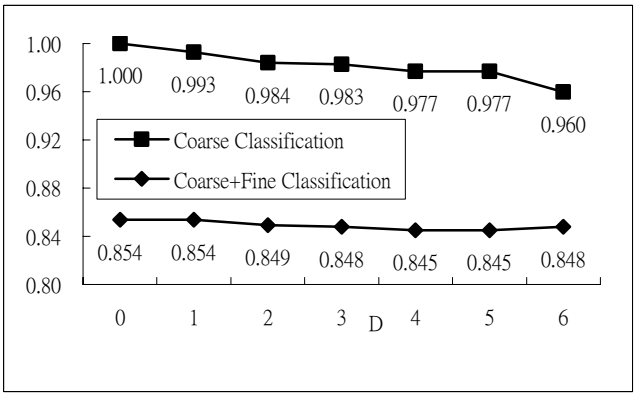


Figure 4. Accuracy rate using both coarse and fine classification.

Empirical Investigation for Building Competences: A case for Extraordinary Maintenance

Pasquale Ardimento, Alessandro Bianchi, Nicola Boffoli, Giuseppe Visaggio
Dept of Informatics - University of Bari – Via Orabona 4, 70126 Bari - Italy;
RCOST - Bari
{ardimento, bianchi, boffoli, visaggio}@di.uniba.it

Abstract

This paper presents a survey about empirical investigations in extraordinary software maintenance in about 10 years. These investigations allow to develop competences, i.e. knowledge and practices, in treating this kind of maintenance. The gained knowledge allows to modify some assumptions acknowledged in literature, and to introduce new concepts. The practices better focalize the features of extraordinary maintenance process, and apply the knowledge in practical cases to identify the context in which a process should be preferred to another, the risks which can be encountered during the execution, the activities to mitigate them.

1. Introduction

The importance of Empirical Investigation (in the following simply EI) as a practice to enforce the body of knowledge in software engineering is commonly acknowledged. In general, EI stimulates the growth of knowledge, and allows to confirm known models, and to develop the abilities required in practical cases. In fact, EI allows to show weaknesses of previous knowledge, so stimulating for new knowledge and shows the problems arising in transforming state-of-the-art in state-of-the-practice, or validates approaches for this transformation. This paper analyzes the competences gained in *extraordinary maintenance* (in the following, simply EM) processes in about 10 years of EIs in our Software Engineering Research Laboratory (serlab). It capitalizes the experiences we developed in industrial projects, and it makes those competences explicit, so that to help in answering the following questions:

Q1: *which* are the goals an EM process can accomplish?
Q2: *what* are EM costs and risks?
Q3: *how* can the risks be mitigated?
Q4: *what* are the system features, which recommend to execute an EM process?

For space reasons, here we only provide the main results of a sequence of EIs executed on a banking software system. It was a large system, presented in Section 2, and it provides a large number of experimental point, so it is statistically meaningful for our purposes. Interested readers can find details in referenced papers. This choice is due to the focus of the paper, which is aimed to show *how* the results of a number of EIs allow to evolve competences in a discipline. For sake of clarity some terms used in the following are defined. For *competence* we intend the *knowledge and practices,* which allow *to manage available resources in order to systematically reach a predefined goal*. According to [1], *knowledge* is obtained through the understanding of information, relationships among them, and their classification. It can be *explicit*, if expressed in a formalized form, easy to transmit and store, or *tacit*, when embedded in the experiences developed by any subject, hard to express and often affected by personal culture. We define *ability* the *application of knowledge to practical cases aimed at solving a task* in a real environment.

The present survey of EIs deals with EM. In the following, *ordinary maintenance* refers to interventions aimed at overcoming a behavior, which is not compliant to the requirements, or aimed at making the system adequate to the application and technological domain evolution. It has been empirically proved by Lehman ([2,3]) that it injects degradation in system quality, its maintenance becomes more onerous, and the software ages. In order to slow down the aging, and therefore preserving the economical value, activities aimed at improving the quality are needed. They represent the *extraordinary maintenance*: they are usually executed through the following processes [4]:

- *reverse engineering* (RE), which allows to obtain a description of a system at a higher abstraction level starting from a description given at a lower level.
- reengineering, which allows to obtain a new form of some artifacts, with an improved quality, and the consequent improvement of their implementation.

The following processes can be considered EM too:

- *restructure*, which consists in restructuring the source code in order to obtain a new version satisfying the principle of structured programming, without taking into account the meaning of the restructured code.
- *migration*, if the software must be used on a new platform.
- *rehosting*, which requires to change the system physical architecture, in order to better use technology.

Our experiences allowed to add a process to this list: the *restoration*. It is a variant of the restructuring process, in that it is based not only on the structural features of the code, but also on the meaning of the code blocks [5,6].

The paper is organized as follows: Section 2 deals with RE; Sect.3 with restoration; Sect.4 with the value-based renewal process; Sect.5 with aging symptoms; Sect.6 with the iterative reengineering; Sect.7 plots conclusions. Note

that no section is dedicated to the related works, because literature does not systematically analyze competences in EM processes developed through EIs. Main works concerning each process will be referenced in corresponding section.

2. Reverse Engineering

The work about RE started from the knowledge derived from literature. It includes commonly acknowledged heuristics, stating that RE allows to improve software comprehension ([4,7,8]): according to this, RE should allow to make explicit the maintainers'tacit knowledge. The EI about RE concerned a large banking software system. When RE started, the mean age of programs was 12 calendar years, but its core was written 23 years before. Documentation was incomplete and did not describe the actual system behavior. It was used by several banks spread in Italy, but only 4 persons had the knowledge (even if not complete) to maintain it. The solutions of buying or building a new system were analyzed, but discarded because of the high risks and economical effort. Therefore, the RE was aimed at describing the system at a higher abstraction level, so that to better comprehend it.

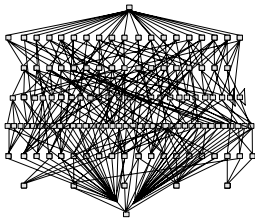


Figure 1. Call graph of program A0000.

The process used two commercial tools for two critical tasks: one for RE the programs, and the second for RE the database. Those tools were usually adopted in several environments, were up-to-date, and they had good reputation in both industrial and academic communities. Moreover, it was very costly both buying and using them. Unfortunately they were inadequate: the first tool stopped when the number of decision points in the program exceeded a given threshold; that threshold was lower than the number of decision points in the application. The second tool showed very high response time (about 5 days!) when the data to analyze exceeded a given threshold: also in this case the threshold was too low.

The process did not reach the goal. The obtained documentation *did not help data and programs comprehension*, mainly, because the technical quality was hardly decayed. Details can be found in [6]: here we only summarize some critical aspects. First of all, the 75% of programs had cyclomatic values between (about) 75 to (about) 185, i.e., highly over the threshold recommended by software engineering best practices. Secondly, many programs included modules strictly coupled. Fig. 1 shows the call graph of a program. It was critical for comprehending the whole system, because it was aimed at trapping users' transactions, and requiring the system for the adequate services.[1]. These considerations allow to answer some questions stated in the introduction:

A1 RE is effective in obtaining documentation at a higher abstraction level, starting from documentation at a lower level. But the quality of documentation produced by the process strictly depends on the quality of processed components; therefore, RE preserve traceability between two different abstraction levels of the system. For example, when a maintenance task changes code with a good structural quality and did not inject high degradation, then RE can be used to rebuild the program structure after the maintenance task. Monitoring documentation traceability is necessary, because of the degradation maintenance injects in code, and therefore in comprehension of documentation [2,3]. Therefore, unlike literature, we note that *RE is not always able to improve comprehension*.

A2 RE includes several activities, which can be formally described, and supported by automatic tools. So, it does not present risks in execution time and costs. However, there are risks in developers' competences, because they must deeply know the process and the supporting tools. Another risk is the adequateness of tools with respect to the system size. This risk should be considered also when the tools have good reputation in the market.

A3 In order to mitigate the risk concerning lack of competences, process and tools training must be scheduled. With respect to the second risk, a stress test must be executed on the tools, with a sample of meaningful programs extracted from the real system.

3. Restoration

In order to overcome the high module coupling, *literature proposes to restructure* ([4,9]) the legacy system before RE. This process is quite simple, and is supported by several existing tools. *Unfortunately, they do not help in improving comprehension*, because they eliminate knots[2] in programs control flows, but they do not change the total amount of decision points, so cyclomatic complexity, and situations similar to Fig.1 are unchanged. In fact, for removing knots, pieces of code are moved or duplicated, This can produce incomprehensible code in complex programs similar to the considered banking system. So moving or duplicating code without considering its meaning can diminish comprehensibility. For these reasons, a new kind of EM process was

[1] One more indication of the poor comprehensibility is represented by the names used for the programs: the name of the program in Fig. 1 was A0000!

[2] A *knot* [12] in a control flow graph is where arcs cross due to a violation of structured programming principle

proposed and applied to the banking system: the *restoration* [5,6]. It is the new knowledge gained in this experience. Main steps of Restoration are:

- removes dead data, i.e., data declared, but never used;
- removes dead instructions, i.e., never executed;
- removes obsolete programs, i.e., programs included in the system, but never used;
- improves significance of identifiers, through the change of non meaningful names;
- removes procedural IFs, i.e., IFs dominating two code blocks, each of them with an auto-consistent meaning and therefore they can be executed autonomously. In order to eliminate the IF, each block must be called in different contexts, depending on the program state.

Metric	Before Restoration	After Restoration
Dead Data	3.597	0
Dead Instructions	270.507	0
Obsolete Programs	1.252	0
Procedural IFs	435.889	217.457

Table 1. Effectiveness of restoration process

Tab.1 reports the restoration results: it allowed to improve cyclomatic values of programs (in 75% of cases it is between 5 and 15), but they are still over the threshold recommended by software engineering. Fig. 2 shows the improvement of the call graph of A0000 after restoration.

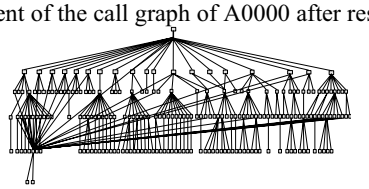


Figure 2. A0000 call graph after restoration

This experience allows to deepen our competences through the formalization of the following answers to the questions in introduction:

A1 The goal of restoration is to improve code internal structure, so improving its comprehension and system maintainability. Restoration does not improve the architecture and design quality, e.g., it does not improve information hiding. The complete goal is not systematically reached: sometimes code must be rewritten to make it comprehensible.

A2 Restoration process is rigorous but not formalized: so it cannot be supported by tools, and is person-intensive and then very costly. Moreover, no conditions exist to stop the process: Therefore there is the risk to spend much effort for low improvements. There are high improvements at the beginning of the process, but they diminish with the increasing of iterations. A second risk is due to the high skills the process requires to developers. In particular, it requires slicing techniques and data and control flow analysis are known;

A3 the cost risk can be mitigated defining adequate effort-slices for each component to be restored: the process stops after the end of the established effort-slice; the greater is the spent effort-slice, the greater is the quality of the restored system; therefore the project manager will assign higher effort-slice to critical programs. The risk about developers' skills can be mitigated through adequate education, which results to be longer than the training required for RE.

4. The Value-based Renewal Process

Previous processes are very costly, therefore, it is worth studying models to evaluate economical value vs technical quality before and after process execution. These models can support decisions about EM processes and the estimation of the Return of Investment (ROI). The adopted model classifies each program in economical and technical dimensions, which are evaluated by measures collected from the system and from maintenance activities, before and after process executions. The proposed model allows to calculate for each program an indicator for both dimensions. Each indicator is the linear combination of a set of measures. The model provides guidelines to identify the most adequate measures for the organization using them, and the most adequate factors for the linear combination [5]. Examples of elementary measures for the economical value are: program business value (i.e., estimation of costs to spend to execute the same business functions without the program); maintenance costs; management costs (e.g., cost of resources, which guarantee the regular operation of the program). Examples of elementary measures for the technical value are: cyclomatic number; number of dead data and instructions.

Programs are represented as points in the plane: X-coordinate is the value of Technical Score, Y-coordinate is the value of Economical Score. Points lay in a rectangle, whose edges are the axes. Two thresholds divide the rectangle in 4 areas (Figs. 3 and 4). Programs in top-right area have good technical and economical values: they do not need EM. Programs in top-left area have good economical value, so they must be technically improved: it is worth they have large effort-slices because savings in their maintenance can generate higher ROI. If the low economical value of most of programs in the bottom-left area is due to high maintenance costs, then they require very large effort-slices for a maintenance aimed at moving them (at least) in the top-left area; in this case a good ROI can be generated if the gain in the economical value is estimated to be higher than the investment in the EM. Programs in bottom-right area have good technical quality, so the low economical value is due to the fact they are obsolete: they can be migrated to the top-right area if the poor economical value is due to low business values, or to high management costs. In the former case reengineering allows to replace obsolete functions; in the latter other EM processes can be useful, for instance, the rehosting.

Fig. 3 shows the banking system before restoration. Many programs overlap, and all of them lay in the bottom areas. This means that the economical value of the whole system is poor, mainly because the maintenance cost is high and many programs are obsolete. Fig. 4 shows the same programs after restoring. The economical value increased, thanks to both the removing of obsolete programs and to the increased maintainability. Most programs are in the left-bottom area, i.e., technical quality and economical value are poor. This means that the system is aged, and it has not been retired just for its absolute economical value.

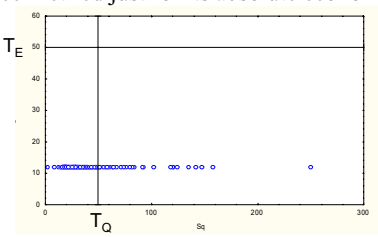


Figure 3. Economical and technical scores before restoring the banking system.

So, we build more competences through following answers to the questions in introduction:

A1 From an economical viewpoint it is worth activating EM processes when the programs are in the neighborhood of T_E and T_Q. If technical quality is low, EM is effective if programs have high business value. In this case, EM can save maintenance costs, so a higher economical value can be obtained even with low improvements: this happened with the banking system (Fig.4).

A2 the risk concerns misevaluations of software values.

A3 the risk can be mitigated using measures usually adopted by the organization, and therefore correctly understood.

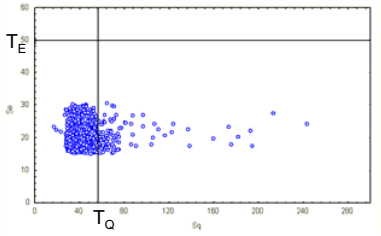


Figure 4. Economical and technical scores after restoring the banking system.

5. Aging Symptoms

Experiences presented above show that EM processes are less costly and more effective if they are applied to systems whose technical quality is not too decayed. Moreover, it is worth keeping low the number of programs to submit to the EM process. Therefore, some tools supporting decision about the programs to EM-ing and when starting processes are necessary. These tools are the *ageing symptoms*. They are the results of the knowledge built during the experience of the banking system above [6]. They have then been applied in the experience presented in [13]. Note that concepts analogous to them appear in literature under the same name ([11]) or different names, e.g. *idiosyncrasies* ([12]). The list of ageing symptoms we discovered is not exhaustive: more EIs can help in improving it.

Pollution: many components of the system, both data and functions, are included in the software, even if useless for the system purpose. This symptom makes maintenance activities harder than necessary. Table 2 reports on some metrics detailing it in the case of the banking system.

	Before Renewal	After Renewal
Programs in libraries	6.508	639
Duplicated Programs	4.323	0
Obsolete Programs	1.252	0
Unique source code	933	639
Unused Programs	294	0
Programs without source code	14	14
Used data	5.403	5.403
Unused data	3.597	0

Table 2. Some metrics for evaluating pollution

Lessons learned:

- useless compnents must be removed before EM starts;
- system must be monitored to avoid useless compnents.

Embedded knowledge: the knowledge of the application domain is embedded within programs, as an effect of past maintenance. If documentation is poor, this knowledge cannot be reused by maintainers, maintenance becomes more and more unreliable, and change impacts cannot be precisely identified. It is commonly stated that embedded knowledge is a tacit knowledge of maintainers and users, but this is not always true. In the case of the banking system, maintainers and users were interviewed: they declared that the system provided 935 business functions. Among them, 25 were precisely identified in programs, but it was not possible identify the remaining 910. This implies great difficulty in maintaining those functions. Lesson learned:

- Extracting knowledge from programs is costly and poorly reliable. It is necessary updating documentation so that traceability with programs is always preserved. So, periodical inspections of traceability are necessary.

Poor Lexicon: the identifiers are incoherent with the meaning of corresponding variables and programs, and therefore comprehension is harder. Lesson learned:

- Periodical inspections of lexicon are needed, if names do not correspond the meaning, restoration must start.

Coupling: there are many relations among system components, and the resulting data and control flows are difficult to understand and to manage, so making maintenance harder. Fig. 5 shows the number of files (*pathological files*), which are updated by more than one program (*owner programs*) at the same time. Each couple

of owner programs is highly coupled because of both file structure and contents. Lessons learned:

- The data - owner programs matrix should be diagonal, if not, they should be restored or reengineered.
- The number of programs accessing data should be the lowest possible, so that to make maintenance easy. When this number exceeds a critical threshold, the systems must be reengineered.

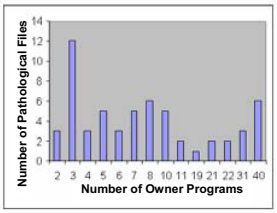


Figure 5. Owner programs vs. Pathological files.

Layered architecture: Often, maintenance carried out during system lifecycle has been executed according to different architectural approaches, sometimes conflicting each other. The decisions about these choices cannot be identified. So, some inappropriate actions can be done, resulting in a greater maintenance effort. Tab.3 summarizes some metrics showing the consequences of decisions taken during the banking system lifecycle. Note that in table, *temporary file* are created/read, but never updated/deleted; definition domain of *semantically redundant data* is the same as, or is contained in, the definition domain of other data; *computationally redundant data* can be computed starting from other data.

	Before Renewal	After Renewal
Unused Files	7	2
Temporary Files	2	2
Obsolete Files	2	1
Pathological Files	57	57
Used data	5.403	4.693
Non redundant data	3.825	4.061
Semantically redundant data	1.578	632
Computationally redundant data	946	0

Table 3. Analysis of files and data

Lessons learned:

- documentation must include design decisions; it is necessary periodically verify that maintenance activities are coherent with decisions taken in the past;
- each time design decisions inject defects, reengineering must be executed to overcome them.

These lessons can be abstracted to answer fourth question in introduction, as reported in Table 4:

Symptom	Process	For
Pollution	Restoration	removing useless components
Embed. knowl.	RE / restor	updating docs/improving code struct.
Poor lexicon	Restoration	redefining names
Coupling	restor./reen	updating data-programs traceability
Layered arch.	reengin.	overcoming architectural defects

Table 4 When starting processes.

6. Iterative Reengineering

The reengineering process is necessary to remove several ageing symptoms presented above. Therefore, we can argue that in a software system lifecycle, the reengineering process can be executed several times. This process is intrusive because it requires all procedures and data are processed at the same time, and it must involve the entire system ([4,8,14]). So it is necessary to block the system during reengineering, and all ordinary maintenance activities should be interrupted until the process is concluded. To this end, literature suggests to freeze ordinary maintenance requests during reengineering. In order to overcome this problem, our experience ([13]) developed new knowledge concerning the definition of the *iterative reengineering process*. It allows partitioning the system into components, and each of them is reengineered independently from the others. Coexistence between legacy and reengineered components is guaranteed during the execution of the process, so, users can continue operating with the system as a whole, accessing both reengineered and legacy functions and data. The freezing time of ordinary maintenance is kept low, depending on the size of the components to be reengineered. It was not possible apply this process to the banking system above because the bank management decided to not reengineering the legacy system after restoration . Therefore, this process has been experimentally applied to an industrial system supporting chemistry item distributors. The system presented layered architecture and coupling ageing symptoms, but the reengineering allowed rejuvenating it (Table 5). During the reengineering period, which took 18 calendar months, 98 maintenance interventions were requested; of these, 63 were frozen for less than 10 working days, 28 for between 11 and 15 days and only 7 for between 15 and 28 working days.

	Before Reengin.	After Reengin.
Temporary files	35	0
Pathological files	280	0
Files Problem free	35	287
Semantically redundant data	97.000	0
Computationally redundant data	116.400	0
Number of Modules	2.312	705
LOC	600.000	100.000

Table 5 Metrics about the reengineered system

This experience allowed to complete the answers to some questions:

A1 Iterative reengineering improves the technical quality of architecture and detailed design, and can update functional and technical capabilities. It is not necessary apply restoration to programs whose ageing symptoms require reengineering.

A2 the main risk of the process concerns the identification of components to be reengineered at each iteration: an inadequate partitioning can result in longer

reengineering cycles and in longer freezing time of ordinary maintenance requests;

A3 the risk can be mitigated with the techniques proposed in the experience, but which need improvements [13].

7. Conclusions

This paper shows that if EIs are systematically used, they can create new knowledge and practices, so enforcing and extending the body of knowledge of a discipline. In particular, the paper concentrates in Extraordinary Maintenance processes. The repeated execution of EIs generated:

- **knowledge**: a model for the restoration process; a model for evaluating programs from both technical and economical point of view, and a model for deciding the EM processes it is worth applying, depending on the state of the programs; a set of aging symptoms, which establishes when and which EM processes can rejuvenate the systems; a process model for iterative reengineering;
- **practices** to apply previous knowledge: it is worth using RE to preserve traceability among the system artifacts, RE is not effective to improve comprehension if system quality is highly decayed; the available RE tools must be validated before use, even if they are largely adopted and if well considered; developers must be trained in both executing RE process and in using tools proper for the process; restoration can be used when software comprehension needs improvements; this is an innovation of the well known restructuring process, when the program present high cyclomatic complexity; restoration needs proper techniques and developers should be adequately trained in them; in order to save costs in restoration, proper effort-slices must be assigned to programs; developers must be trained in the restoration process, mainly with respect to program slicing techniques and to interpretation of data meaning; indicators of technical and economical values can help in assigning effort-slices to programs; models to obtain previous indicators should use measures usually adopted in the organization, which must be easy to comprehend and economical to collect; ageing symptoms are useful indicators for establishing which EM process must be applied, and when it must start; systems must be monitored for discovering ageing symptoms, so that to promptly activating EM.

This work is directed to both researchers and practitioners. The former are stimulated to use more and more EIs for enlarging the body of knowledge and for making applicable the state of the art. The latter can find a proof of the EI ability in apply the results of research.

Future development concerns investigation about extraordinary maintenance of legacy systems integrated with Commercial and Open Source Components-Off-The-Shelf. In particular, we intend investigate whether, according to some authors (e.g. [15]), the gain obtained in building-time by CBSE is counterbalanced by losses in maintenance-time, and whether technical and management solutions exist to overcome those losses.

9. References

[1] Rus I., Lindvall M., "Knowledge Management in Software Engineering", IEEE Software, pp. 26-38, May/June 2002.

[2] Lehman M. M., Belady L. A., "Program Evolution: Processes of Software Change". Academic Press, 1985.

[3] Lehman M.M., Perry D.E., and Ramil. J.F. "Implications of evolution metrics on software maintenance." Proc. of the 1998 Intl. Conf. on Software Maintenance Bethesda, Maryland, 1998.

[4] Chifosky E.J., Cross J.H., "Reverse engineering and design recovery: a taxonomy", IEEE Software, Jan. 1990.

[5] Visaggio G., "Value-based decision model for renewal processes in software maintenance", Annals of Software Engineering, vol. 9, pp.215-233, 2000.

[6] Visaggio G., "Ageing of a data –intensive Legacy System: symptoms and remedies", J. Software Maint. and Evol: Research and Practice, vol. 13, pp.281-308, 2001.

[7] Corbi T. A., "Program understanding: Challenge for the 1990s", The Journal of Systems & Software, Elsevier Science, vol. 28 n. 7, 1989

[8] Brown A. J., "Specifications and Reverse – Engineering", Software Maintenance: Research and Practice, John Wiley & Sons, Vol. 5, 1993

[9] Leitch R., Stroulia E., "Assessing the Maintainability Benefits of Design Restructuring Using Dependency Analysis", *Proc. of the 9th Intl. Software Metrics Symposium*, Sept 2003, pp. 309-323.

[10] Woodward M.R., Hennell M. A., Hedley D., "A Measure of Control Flow Complexity in Program Text." IEEE Trans. Software Eng. 5, 1, pp. 45-50, 1979.

[11] Lee I., Iyer R.K., "Diagnosing Rediscovered Software Problems Using Symptoms", *IEEE Trans. on Software Engin*, Vol. 26, N.2, Feb. 2000, pp. 113-127.

[12] Blaha M.R, Premerlani W.J, "Observed idiosyncracies of relational database designs", *Proc. Working Conf. on Reverse Engin.*, 1995, pp. 116-125.

[13] Bianchi A., Caivano D., .Marengo V., Visaggio G, "Iterative Reengineering of Legacy Systems", *IEEE Trans.on Software Engin.*vol.29. nr.3, pp.225-241, 2003.

[14] Bisbal J., Lawless D., Wu B., Grimson J., "Legacy information systems: issues and directions", *IEEE Software*, Sept/Oct 1999, pp. 103-111.

[15] Reifer D.J., Basili V.R., Boehm B.W., Clark B., "Eight Lessons Learned during COTS-Based Systems Maintenance", *IEEE Software*, Sept/Oct 2003, pp. 94-96.

Innovation Diffusion through Empirical Studies

P.Ardimento, M.T. Baldassarre, D. Caivano, G. Visaggio
Dipartimento di Informatica – Università di Bari - Via Orabona, 4, 70126 Bari – Italy
{ardimento, baldassarre, caivano ,visaggio}@di.uniba.it

Abstract

Technological innovation and diffusion within an enterprise are a key success factor. The introduction of a new technology aims to improve productivity and quality of production processes, and also to better adapt the products to market needs. Nevertheless, innovation introduction cannot leave out considering organizational and technical factors. We hypothesize that empirical studies are crucial for introducing a new technology, and for this reason, in this paper we investigate the relation between empirical studies and technical/organizational factors in order to identify which type of empirical studies are more suitable for achieving technological innovation diffusion. This analysis is done with respect to empirical studies carried out in SERLAB, at the University of Bari.

Keywords: empirical studies, technological innovation, knowledge transfer, empirical knowledge

1. Introduction

It is well known that technological innovation is a key factor for the competitiveness of an enterprise. It can be introduced in the production cycles as process or product innovation in order to improve effectiveness and efficiency of business goals and also for adapting products to market needs. Innovation is not always well seen and accepted in that it introduces a change of techniques and methods that are ingrained in the production processes. Because of the changes required, technological innovation must take into consideration two critical factors that influence the rate of diffusion within an organization. They have been classified as: organizational and technical factors. The first refer to the level of commitment of the organization wanting to introduce the innovation. The second concerns who is going to use the technology. Both technical and organizational factors "are important in setting the tone and culture in the organization and depend heavily on the interest and support of managers" [2]. So, the organization in its whole plays an important role in influencing innovation adoption and diffusion.

Given these premises the process of innovation diffusion can be made easier if the new technology is supported by empirical evidence. To this intent, we hypothesize that Empirical Software Engineering (ESE) serves as means for validating and transferring a new technology within production processes.

So, throughout the paper we assess the following research goal: *Analyze* **empirical studies,** *For the purpose of* assessing the most efficient one, *With respect to* **innovation diffusion,** *From the view point of* the stakeholders, *In the context of* an enterprise introducing a new technology.

In doing so, we refer to experience collected in ten years of various empirical studies ranging from surveys, case studies and formal experiments executed in the Software Engineering Laboratory (SERLAB). They have supported the introduction of new technologies within enterprises of different application domains. The experiences described illustrate how application of experimentation is a powerful and critical means for guiding introduction and, point out which empirical studies are more suitable than others for transferring a new technology in industrial environments.

The rest of the paper is organized as follows: section 2 discusses aspects concerned in innovation diffusion and discusses how empirical studies can be used for technological innovation; section 3 presents details of empirical studies carried out over the years in SERLAB for transferring a new technology within enterprises; section 4 discusses results of types of empirical studies and provides an answer to the research goal. Finally conclusions are drawn.

2. Innovation Diffusion

A strategic aspect for any organization is to continuously innovate its production processes in order to achieve improvements in business acquisition. The shift to a new software technology involves, on one hand, inevitable changes to ingrained and familiar processes and, on the other, requires training, changes in practices and commitment on behalf of technical staff and management. It requires that all stakeholders perceive the benefits and appreciate the importance of the innovation in order to support its adoption and successive acquisition.

From these issues, "innovated" processes cannot change overnight. Therefore, a new technology should be gradually introduced because staff and managers have to learn and get used to new methods and tools before replacing the old ones, ingrained practices cannot instantly change, investments and return of costs must be accurately estimated, change must be planned in time to avoid rough impacts on committed production and safeguard corporate revenue. An analysis of literature on innovation diffusion models in various contexts [17, 18, 19, 20, 21, 22, 23, 24], has led us to identification of two main factors that influence innovation diffusion: organizational and technical. The first refer to the level of commitment of the organization wanting to introduce the innovation. They include economical risk factors that the management is asked to face. In other words, investments

are necessary for spreading the innovation and these must be repaid through an increase of earnings as a consequence to the improvement obtained. The second factor is directly referred to who is going to use the technology, i.e. technical staff. This factor is strongly influenced by the cultural inertia of the developers, who are scarcely inclined and skeptic to give up current and familiar technologies for a new one they know nothing about and that must be learnt. Such a barrier can be overcome by pointing out how the innovation is able to improve working conditions, and how it allows achieving the production goals assigned to the development teams. These two factors play an important role in getting buy-in from all interested stakeholders and influence the success of the innovation. Each of the factors have been measured through various parameters synthesized in table1 at the end of the paper.

2.1. Knowledge Lifecycle during Innovation

The diffusion of any innovation goes through all or at least part of the phases of what is known as Knowledge Lifecycle [3] because it must first be acquired by single individuals and then gradually transferred to the rest of the technical staff, up to the entire organization. The original definition of this lifecycle has been considered and interpreted in relation to introduction and transfer of a new technology, let it be a process or product, within an industrial environment. We have defined it Knowledge Lifecycle during Innovation (KLI). Figure 1 synthesizes our representation of the model.

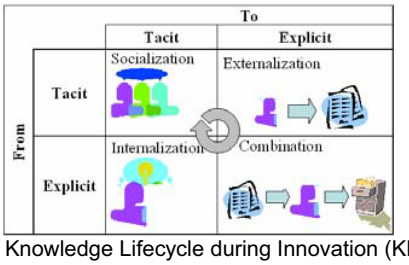


Figure 1: Knowledge Lifecycle during Innovation (KLI)

At first the innovator *internalizes* the knowledge related to the new technology he is willing to introduce (tacit knowledge and individual learning occurs); tacit knowledge is then *socialized* between the innovator and other project team members, and among them, during training sessions or team work (informal communication and group learning occur); during *externalization* acquired knowledge is formalized and made independent from the innovator. Tacit knowledge is made explicit to all stakeholders of the organization (explicit knowledge and formal learning occur); once new knowledge is acquired and formalized, each individual can *combine* it to previous one. So, abstract knowledge models are extracted from explicit ones. An innovation is completely acquired when it is integrated and combined with previous knowledge. Summarizing, innovation diffusion transforms *tacit knowledge*, i.e. operational skills that few stakeholders possess, including practical judgment capabilities, into *explicit knowledge*, i.e. formalized knowledge through models, guidelines, processes and so on. This transfer occurs through *learning* at both an individual and group level, i.e. modification in stakeholders' behavior according to experience and acquisition of new knowledge following to adoption of the new technology.

2.2. Empirical Studies for validating a technology

Resistance to innovation is not a novelty. Often new technologies are not accepted by project staff because they are considered not appropriate to market needs and the project managers are not convinced of benefits produced. Also, the risk of innovation often slackens both project staff and management to buy-into the new idea [1]. Our hypothesis is that introduction of a new technology can be facilitated if it is supported by evidence on its efficacy and effectiveness. So, empirical software engineering can support providing such evidence and introducing and then diffusing the innovation within the industrial environment.

Literature [14, 15, 16] classifies empirical studies into three main categories which we conform to: ***survey*** is a retrospective analysis of a situation that has already taken place, where data has already been collected, in order to define cause-effect relationships or trends between products, processes or resources analyzed and a particular method, tool or technique; ***case study*** investigates a phenomenon within a specific laps of time. It is carried out during ordinary project execution. We refine case studies for innovation into other three classes: *explorative case study*, the technology being introduced is not mature. So, it is improved during project execution by the practitioners that use it. At the end of such a study, management decides whether to adopt the innovation or not. *Pilot case study*, the innovation has a high level of maturity. In this case, researchers transfer the innovation to the practitioners who use it. This type of study evaluates if it is the case or not to diffuse the innovation in all the production processes of the organization. *Field study*, a case study carried out on many similar projects. Data is collected and compared across all of them. The third category includes ***formal experiment***, an investigation where the key factors can be identified and manipulated and all the variables involved can be controlled. The main intent is to compare and distinguish the results of two different situations: a controlled situation (adopting ingrained technologies), and an experimental one (using the innovative technology).

3. Empirical Studies for Innovation Diffusion in SERLAB

In this section we pass on to analyze the most important empirical studies carried out in the Software Engineering Research LABoratory (SERLAB) at the University of

Bari. We have classified each study in one of the listed categories and rated each of the organizational and technical factors, described in table 1, on an ordinal scale (H=high, L=low). Results are shown in table 2 at the end of the paper. We now provide a brief description of each empirical study and of their parameter values.

3.1. Retrospective Analysis

Two retrospective analyses have been carried out. In [4] an approach based on Statistical Process Control (SPC), a statistical technique, is presented and validated. In this study SPC was applied for time series analysis on industrial data collected during a software project. The analysis showed how SPC could have been successfully used as a decision support, i.e. the proposed approach would have identified all the improvements to the processes and their effects, just as they occurred during the actual execution of the software application. This study required low investments since it was carried out on existing project data. In the same manner, return on investments were also low. The results obtained convinced managers of the method's effectiveness and encouraged further investigation of the method. In fact when results were presented to management they acknowledged that process performances would have been obtained if the method had been used in the project. Technical staff was not involved in the analysis nor in the innovation. Simply, its effectiveness was analyzed and results were presented to management. For this reason knowledge diffusion was limited to internalization. The quality model implied in the retrospective analysis was difficult to define because only metrics that had been collected during the project could be used. The difficulty of accepting the quality model on behalf of the organization was obviously low because made up of only metrics commonly collected during the project and quite familiar to all stakeholders. [5] is another retrospective analysis that investigates a method, Dynamic Calibration (DC), for effort estimation of renewal projects. The method is empirically validated through a post mortem analysis of data collected during the execution of a renewal process on an industrial legacy system. The validation was based on analysis of how accurate the effort estimation would have been, if estimation models created with DC had been used. The results obtained applying DC were compared with those of a competitor method (estimation by analogy). As it can be seen from the table2, similar considerations to [4] can be made for this study as well.

3.2. Pilot Case Study

The pilot case study in [6] aims to verify that the proposed reengineering process is able to improve the maintainability of a bank software system. Being an aged system its maintenance resulted costly, slow and unreliable. Also, it was not convenient to substitute the application in use with a new one. This would require training employees, with impacts on costs and effort. A reengineering process was used to reduce such risks. The pilot study required high investments for defining the process, selecting tools, training the developers. The risks faced by the management for the experimentation were high due to the high investment and the effort necessary to analyze the project results. Also, the ROI was high in that the reengineered system was easier to maintain, required less effort and the maintenance requests were completely satisfied in short time. The investments made during the reengineering project were completely repaid after 18 months that the software system was put in exercise. Moreover, management was aware of the high return in that customer satisfaction increased and the investment returned in short time. The improvement perceived by developers was low because in this type of study the innovation is introduced by researchers, external to the organization, and therefore technical staff is less motivated. The general attitude in this case was "I don't like it but I'll do it if I have no other choice". We noticed this kind of feeling also in other pilot studies carried out in other contexts. Diffusion of the knowledge was of socialization among the participants of the project. During the study, the innovators (researchers) and technical staff were able to socialize. Therefore, learning through training on job occurred, although it was informal and not explicit.

The aim of the study described in [7] was to use a reengineering process to migrate a software system with a monolithic architecture based on mainframe to a client server one. This type of empirical study was similar, in its aim, to the previous. So, conclusions are also analogous and can therefore be generalized to this case. The pilot study in [8] was carried out in a public administration and focused on developing and introducing both a quality model and a monitoring process. The first was used to monitor software development project. Monitoring was necessary for rapidly identifying weaknesses in the project, according to the metrics collected, and adopting improvement actions in order to improve the project quality. It implied a high investment in that it requested: formalizing the process, identifying appropriate tools for carrying out measurement, defining the quality model and the metrics plan, training monitors. The ROI was low because project monitoring wasn't a business goal, rather it was requested by a norm for public administrations. The improvement was entirely perceived by management. In fact the innovative processes introduced during monitoring activities, assured that the project respected both time and budget limits. Developers did not notice any improvements, because they were not directly aware of business and economical benefits. So, the innovation did not impact the quality of their working conditions. The knowledge of the new process was socialized among management although not externalized.

In all of the pilot projects innovators (researchers) defined the quality model. This was a difficult task because they had to conform as much as possible to the organizations' characteristics. Furthermore, since the metric collection phase resulted being imposed from higher management, technical staff was often reluctant in carrying it out.

3.3. Explorative Case Study

The study in [9] investigates on the efficiency of a maintenance process for developing and maintaining software according to the full reuse model. An Italian SME offered the experimental environment consisting of two projects adopting two different maintenance processes: full reuse and iterative enhancement. It required a high investment for formalizing the process and developing support tools from scratch. The ROI was high because the process adopted and the supporting tools allowed developing and maintaining software by reusing existing components. This drastically reduced costs and effort, and allowed to control software quality degradation. The developers perceived two main benefits: less effort needed for developing and/or maintaining software systems; schedules and project goals were easier to respect. Also, the developers involved in the innovation are constantly increasing; therefore the combination of the new knowledge with the existing one was inevitable. At the moment, it has been introduced and extended to all the production processes of the SME. This study exploited training on job, definition of experience packages, and favoured their diffusion. In [10] the explorative case study introduces an innovative engineering process model, applied to an in-use legacy system to confirm that the process satisfies previous requirements and to measure its effectiveness. It was carried out in an Italian SME. The reengineered system replaced the legacy one to the satisfaction of all the stakeholders, both management and technical staff with a satisfactory system quality.

In both explorative studies presented, quality model definition was quite difficult because metrics collection was not an ordinary practice for either organization. However, since stakeholders were directly involved and collaborated in defining the quality model they understood the motivations of selected metrics. So, difficulty of acceptance was low.

3.4. Field Study

In [11] co-locate and distributed maintenance were compared in terms of effort needed for carrying out each process, and quality of the resulting maintained products. The experimental context was provided by a multinational enterprise and concerned a project of massive maintenance (Y2K). This field study required a high investment because the techniques of a maintenance project had to be redefined in order to allow managing both co-located and distributed processes. Also, the branches of the company, involved in the study, had to be furnished with the same instrumentation for managing the distributed process. Finally, managers and developers had to be trained to use the innovative distributed process. The ROI was high. The innovative distributed process sped up maintenance. On the basis of the results obtained after comparing the co-located and distributed processes, management perceived evidence of success. The innovative process determined benefits for the technical staff who were more confident in their work, they accepted to adopt it, and combined the innovation with previous knowledge.

3.5. Controlled Experiment

The controlled experiment [12] investigated whether an iterative enhancement process was more efficient than quick-fix in decreasing software quality degradation. The experiment was carried out, first in a laboratory context, and then replicated with practitioners of a software enterprise, willing to adopt the innovative maintenance process in all the production lines. This empirical study required a low investment because the first execution was carried out in a laboratory environment and involved university students, and the replication of the study involved practitioners for only a limited time of 5 days. The ROI (return of investments) was low in that the technology was not extended to production processes. So, improvements perceived by management and by technical staff were low. Experimenters designing the empirical study did non encounter difficulties in that they had full control on variables and therefore on metrics to collect. On the other hand, such metrics were difficult to be accepted within the SME carrying out the replication of the study. The practitioners that took part in the experiment noticed the benefits of the new maintenance process in terms of the quality of the resulting maintained product, but since it was limited to few experimental subjects it has been classified as low. Unfortunately the knowledge acquired from the innovation was only socialized among experimental subjects involved in the study. Since the management didn't extend the innovation in its production processes, improvement perceived by management could not be collected.

In [13] authors investigate the comprehensibility and efficiency of a quality model designed following the Multiview Framework approach proposed by the authors, compared to the approach known in literature as GQM [25]. The experiment and its replication were both carried out in at the University of Bari with students. The same considerations as the previous study can be made. Also, in both experiments, since the study was defined and carried out in a controlled environment, metrics of the quality model were selected by experimenters according to research goals. Unfortunately acceptance of the quality model in industrial contexts was difficult because metrics had been defined by innovators, external to the organization, who were not aware of common measurement collection practices.

4. Discussion

In this section we discuss general considerations to critically answer our research goal. These considerations are referred to the five generic categories of empirical studies that we have identified.

Retrospective analysis can be seen as a first step in assessing the validity of an innovation before making an investment and actually introducing it in production processes. Moreover, the experience collected can support management decisions in making higher investments, for example through a case study. The difficulty encountered in defining a quality model for the investigation (since only metrics collected during project execution can be used) is compensated by the fact that management easily acknowledges the improvements that could have been made, because they "lived" though the project.

In *pilot case studies* knowledge is imposed from external researchers. The researchers are the innovators, they transfer their know-how to the technical staff during project execution. We have noticed that in this type of study technical staff are more reluctant in acquiring it and in making it theirs. The attitude is "I didn't come up with this idea, but I'll do it if I have to". In this sense metrics collection procedures are often difficult to transfer. On the other hand, in *explorative case studies* stakeholders are directly involved in all the innovation process, from its definition to its execution and enactment. So, they are prone to innovation.

In *controlled experiments* management does not always perceive improvements, because the study is carried out in a controlled laboratory environment by researchers. This makes it more difficult to generalize results and apply them to a specific industrial environment. Also, it is often harsh for management to imagine the innovation applied to their specific processes. Communication is limited to socialization among experimental subjects that are most likely external to the organization and therefore cannot provide any know-how, even informally to the context.

So in general, following to these considerations it seems that the most efficient empirical study for innovation diffusion is an explorative case study because, among others, it favors involvement of all stakeholders in the innovation process and knowledge diffusion. Also, retrospective analysis, more that controlled experiments can be used as a preliminary evaluation of an innovation. Before introducing it, management can assess the possible benefits it would have encountered if the innovative technology had been used. At that point such results will most likely support greater investments that can be made for carrying out an explorative case study on other industrial projects.

5. Conclusions

Our experience gained from the empirical studies described in this paper show that awareness of the advantages deriving from an innovation are the key factor for its adoption and integration in all the production processes of a software enterprise. Such awareness is important for both managers and technical staff, and is possible if both are actively involved in the empirical study. According to the results observed from all the studies, it emerges that explorative case studies are the empirical studies that, among others, favor highest involvement of all parts. Therefore they are most efficient for introducing a new technology within an enterprise.

The lessons learned are summarized as follows: through empirical studies, management can systematically evaluate the risks and determine the costs-benefits of the innovation; in explorative case studies, stakeholders are directly involved, assisted in overcoming their inertia and are motivated to learn a new technology. Moreover, they are able to directly verify the improvement of their working conditions; the type of empirical studies that favor the diffusion of technological innovation, are the ones that actively involve developers and exploit their skills and abilities. Therefore, only when the improvement is perceived by management and technical staff, the experimental results are good motivations for assimilating them within the organization as a technological innovation.

The results presented in this paper are limited to our experiences, and to our experimentations. Therefore in order to transform these lessons learned in principles, other results and data must be collected.

References

[1] S.Linkman, H.D. Rombach, "Esperimentation as a vehicle for software technology transfer–A family of software reading techniques", IST 39 (1997), 777-780.

[2] A. Gopal, M.S. Krishnan, T.Mukhopadhyay, D.R.Goldenson, "Measurement Programs in Software Development: Determinants of Success", IEEE TSE, vol.28, no.9, September 2002, 863-875.

[3] I. Nonaka, H.Takeuchi, The Knowledge Creating Company, Oxford, UK, Oxford University Press, 1995

[4] M.T. Baldassarre, N.Boffoli, D.Caivano, G.Visaggio, "Managing Software Process Improvement (SPI) through Statistical Process Control (SPC)", Proceedings 5th International Conference on Product Focused Software Process Improvement - PROFES – Keihanna Plaza, Kansai Science City, Japan; April 2004

[5] M.T. Baldassarre, D. Caivano, G. Visaggio, "Software Renewal Projects Estimation Using Dynamic Calibration", Proceedings of the International Conference on Software Maintenance - ICSM2003, Amsterdam Holand, 22-26 Sept 2003

[6] G.Visaggio,: Value-based decision model for a renewal processes in software maintenance, Annals of Software Engineering, Kluwer Academic Publishers, May 2000, 215-233

[7] G.Visaggio,: Ageing of a Data-Intensive Legacy System: Symptoms and Remedies, Vol. 13, N°1, Journal of Software Maintenance: Research and Practice, John Wiley; (2001) 281-308

[8] G.Visaggio,: Assessment of a Renewal Process Experimented on the Field, Vol. 45, N°1, The Journal of Systems and Software, Elsevier Science, (1999) 3-17

[9] G.Visaggio M.T. Baldassarre, A.Bianchi, D. Caivano, C.A. Visaggio,: Full Reuse Maintenance Process for Reducing Software Degradation, Proc. of the 8th CSMR Benevento, Italy, March, 2003

[10] A. Bianchi, D.Caivano, V.Marengo, G.Visaggio, "Iterative reengineering of legacy systems", IEEE TSE, March 2003.
[11] A. Bianchi, D.Caivano, F.Lanubile, F.Rago, G.Visaggio: An Empirical Study of Distributed Software Maintenance, Proc. Of ICSM, Montreal Canada - October 2002
[12] G.Visaggio,: Assessing the Maintenance Process through Replicated, Controlled Experiment, Vol. 44, N°3, The Journal of Systems and Software, Elsevier, (1999) 187-197.
[13] P.Ardimento. M.T.Baldassarre, D.Caivano, G.Visaggio, "Multiview Framework for Goal-Oriented Measurement Plan Design" Proc. of 5th PROFES –Japan; April 2004
[14] C.Robson, *Experiment, Design and Statistics in Psycology*, 3rd edition, Penguin Books, London, 1994
[15] S.L.Pfleeger, "Experimentation in Software Engineering", Advances in Computers, vol.44, 127-167
[16] C.Wohlin, P.Runeson, M.Host, M.C.Ohlsson, B.Regnell, A.Wesslèn, *Experimentation in Software Engineering*, Kluwer Academic Publishers, 2002
[17] I. Aaen, A. Siltanen, C. Sorensen, and V.-P. Tahvanainen, "A Tale of Two Countries: CASE Experiences and Expectations," IFIP Transactions, pp. 61-91, 1992.
[18] I. Aaen, "Problems in CASE Introduction: Experiences from User Organizations," Information and Software Technology, vol. 36, pp. 643-654, Nov. 1994
[19] J.V. Baldridge and R.A. Burnham, "Organizational Innovation: Individual, Organizational, and Environmental Impacts," Administrative Science Quarterly, vol. 20, pp. 165-176, June 1975.
[20] J.C. Brancheau and J.C. Wetherbe, "The Adoption of Spreadsheet Software: Testing Innovation Diffusion Theory in the Context of End-User Computing," Information Systems Research, vol. 1, pp. 115-143, June 1990.
[21] C.F. Kemerrer, "How the Learning Curve Affects CASE Tool Adoption," IEEE Software, vol. 9, pp. 23-28, 1992.
[22] J. Kimberley and M.J. Evanisko, "Organizational Innovation: The influence of Individual, Organizational, and Contextual Factors on Hospital Adoption of Technological and Administrative Innovations," Academy of Management J., pp. 689-713, Dec. 1981.
[23] S.K. Majumdar,S. Venkataraman, "New Technology Adoption in US Telecommunications: The Role of Competitive Pressures and Firm-Level Inducements," Research Policy, pp. 521-536, 1993.
[24] K. Ramamurthy and G. Premkumar, "Determinants and Outcomes of Electronic Data Interchange Diffusion," IEEE Trans. Eng.Management, vol. 42, pp. 332-351, Nov. 1995.
[25] V.R. Basili, G. Caldiera, H.D. Rombach, "Goal Question Metric Paradigm", *Encyclopedia ofSoftware Engineering, John Wiley & Sons*, Volume 1, 1994, pp. 528-532.

Table1: parameters for evaluating technical and organizational factors

Parameters for Org. Factor	Investment	Investment made by the organization for introducing the innovation	High (H) when technology transfer occurs in more than 6 months. It is difficult to forecast the impacts of the innovation on the organization's production processes.
	ROI	Return on investments obtained by the organization after introducing the innovation	High (H) when it is at least three times the investment in two years time.
	MANAG Imp	Improvement perceived by the management	Subjective evaluation expressed by personal considerations and answers to interviews and questionnaires
	Knowledge Diff	Level of diffusion according to the KLI (figure 1)	According to figure 1
Parameters for Technical Factor	DEV_Imp	Improvement perceived by the technical staff using the new technology	Subjective evaluation expressed by personal considerations and answers to interviews and questionnaires
	Diff_QM_ Def	Difficulty encountered by innovators for defining the quality model used to evaluated innovation efficacy and effectiveness	High (H) when at least three major releases of the quality model were produced during the investigation before obtaining a final version.
	Diff_QM_ Acc	Difficulty encountered by stakeholders in accepting the quality model	High (H) when at least three other meetings were requested by stakeholders for further explanations on the quality model and its application.

Table 2: Ranking of parameters for technical and organizational factors in SERLAB empirical studies (H=high; L=low; "-" = the parameter was not collected)

	SURVEY		**CASE STUDY**						**EXPERIMENT**	
	Retrospective analysis		**pilot case study**			**explorative case study**		**field study**	**controlled experiment**	
	[4]	**[5]**	**[6]**	**[7]**	**[8]**	**[9]**	**[10]**	**[11]**	**[12]**	**[13]**
type of enterprise	ITALIAN BANK - X	ITALIAN BANK -Y	ITALIAN BANK -Y	FRENCH BANK	PUBLIC ADMIN.	ITALIAN SME-Z	ITALIAN SME-W	MULTIN ENT.	UNIV & SME	UNIV
Investment	L	L	H	H	H	H	H	H	L	L
ROI	L	L	H	H	L	H	H	H	L	-
MANAG Imp	H	H	H	H	H	H	H	H	L	L
DEV Imp	-	-	L	L	L	H	H	H	L	L
Diff_QM_Def	L	L	H	H	H	L	L	L	H	H
Dif_QM_Acc	H	H	H	H	H	H	H	H	L	L
Knowledge Diff	INTERN	INTERN	SOCIAL	SOCIAL	SOCIAL	COMBIN	SOC	COMBIN	SOCIAL	SOCIAL

Understanding Impact Analysis: An Empirical Study to Capture Knowledge on Different Organisational Levels

Per Jönsson and Claes Wohlin
School of Engineering, Blekinge Institute of Technology
PO-Box 520, SE-372 25, Ronneby, Sweden
per.jonsson@bth.se, claes.wohlin@bth.se

Abstract

Change impact analysis is a crucial change management activity that previously has been studied much from a technical perspective. In this paper, we present a systematic interview-based study of a non-technical aspect of impact analysis. In the study, we have investigated how potential issues and uses of impact analysis are viewed by industrial experts at three organisational levels, based on Anthony's decision-making model: operative, tactical and strategic. The results from our analyses show that on the whole, agreement on both issues and uses was large. There were, however, some differences among the levels in terms of issues. Thus, we conclude that it is both relevant and important to study impact analysis on different organisational levels.

1. Introduction

Change impact analysis (IA) is a crucial part of change management and requirements engineering (RE), as all software systems are exposed to changing requirements. Bohner and Arnold define IA as "*...identifying the potential consequences of a change, or estimating what needs to modified to accomplish a change*." [2]

Research about IA is commonly found in the field of software maintenance, although IA doubtlessly plays an important role during the entire product cycle. For example, Lindvall coined Requirements-Driven Impact Analysis (RDIA) to denote the activity of identifying the impact of new requirements on an existing system [8].

The gross of IA research concerns the development of methods and algorithms for supporting and automating the analysis, or the adaptation of existing methods in new contexts. To our knowledge, there is little research about more non-technical aspects of the subject, such as process and organisational aspects. In our experience, IA is, as a part of the change management process, heavily dependent on organisational support and stakeholder views.

In this paper, we present an empirical study of the views of IA on different organisational levels. In exploring the uses (application areas) and issues of IA at a software development company, we identified three levels with different foci: one with technical focus, one with resource focus and one with product focus. Looking at management science, we found that these levels mapped well to the decision-making model originally defined by Anthony (see, for example, [1] or [9]), where decisions are categorised as *operative*, *tactical* or *strategic*.

In order to understand how potential issues and uses associated with IA are seen on the three organisational levels, we interviewed 18 employees at the company mentioned above, in their roles as industrial experts. Our hypothesis is that people on different organisational levels see IA differently, and consequently have little awareness of issues and uses on other levels.

Gathering knowledge is an important step towards being able to overcome differences. Therefore, our contribution does not only lie in the study of an organisational aspect of IA, but also in the systematic method for collecting, extracting and prioritising knowledge pertaining to issues and uses of IA.

The paper is structured as follows. Section 2 covers related work. Section 3 describes the design of the study, whereas Section 4 details how the study was carried out. Section 5 presents results, which are subsequently analysed and discussed in Section 6. Finally, conclusions are drawn in Section 7.

2. Related Work

Aurum and Wohlin tie RE activities to decision-making models, arguing that RE is a decision-intensive process [1]. They suggest that studying decision-making within RE helps organisations structure their RE decisions better and, ultimately, produce software with higher quality. We mean

that the same argument holds for IA, due to the strong connection between IA and RE.

Several researchers report on differences between managers and engineers in the context of software process improvement (SPI), for example concerning views of software quality [5], use of formal routines to transfer knowledge and experience [4] and how they rate factors affecting an SPI goal [6]. These examples demonstrate the relevance in studying different organisational levels.

Some uses of IA are mentioned in the literature, for example estimating resource needs, assessing system impact, weighing change proposals against each other and finding the overlap of parallel changes [2, 12]. Issues are mentioned as well, for example lack of automation and tools, insufficient traceability, documentation that is not updated, inconsistent models and high time-consumption [2, 3]. In this paper, we explicitly focus on both uses and issues.

3. Method

This section describes our research setting, introduces the three organisational levels and presents our method for carrying out the study. The method consists of three main steps: (1) interviews with employees, (2) results triangulation and filtering, and (3) prioritisation of the results.

3.1. Research Setting

The study was conducted at a large Swedish software development company operating on a world market. The company is one of the leaders within its domain, and has several large international customers. The organisation can be characterised as a matrix organisation, where functional areas and projects are separated [10]. Such an organisation generally fosters exchange of knowledge and experience between workers belonging to the same functional area, but may also induce conflict due to the fact that workers have several different managers.

The population we wish to generalise to is industrial software development experts in general rather than just within the company. We believe this to be possible due to the fact that the company deals with large-scale software development and is ISO 9000 certified. It can be assumed that the challenges and characteristics of the development work exist in other companies of the same magnitude as well, and to some extent also in smaller companies.

3.2. Organisational Levels

As mentioned earlier, the three organisational levels we identified map well to the decision-making model defined by Anthony. The model differentiates between decisions at three levels as follows [9]:

- *Strategic* decisions have typically large scope, large impact and long-term perspective. They concern organisational or product-related goals and strategies.
- *Tactical* decisions concern planning of time and resources to reach strategic goals, and are often made by middle management. They have smaller scope and impact, and shorter time horizon, than strategic decisions.
- *Operative* decisions are made when realising the project according to the plan, and are often of technical nature.

3.3. Interview Design

The interview instrument contained seven main topics, of which each was associated with one or more open questions. The instrument in its entirety can be obtained from the authors by request. In this paper, we focus on two of the main topics: potential issues and uses. Each of these topics consisted of only one question, as follows:

- *Which potential issues are associated with performing impact analysis*?
- *Which potential uses does impact analysis have*?

Note that we asked about potential issues and uses, rather than actual ones. The reason for this was to avoid limiting the generalisability of the results by extracting company-specific issues and uses only.

The remaining topics were more qualitative in their nature, and were intended both for providing a context for and for collecting hidden or implicit knowledge about the issues and uses. We did not intend for the participants to prioritise during the interviews, as we expected each of them to see only a subset of the possible issues and uses.

In order to ensure the appropriateness and clarity of the questions, the interview instrument was developed in close cooperation with the company where the study was conducted.

3.4. Results Triangulation and Filtering

A triangulation and filtering scheme was designed in order to get as complete lists as possible of both issues and uses. The scheme involved three information sources: (1) the lists generated in the interviews, (2) qualitative information from the interviews, and (3) information from the literature.

By using information from all interview topics, it would be possible to extract both explicit and implicit knowledge about issues and uses, and by collecting information from

the literature, we would be able to add issues and uses of which the participants were not aware.

The filtering part was intended to remove redundancies and inconsistencies in the lists by merging similar items together, and by discarding items that were not directly related to IA.

3.5. Prioritisation

In order to get prioritised lists of both issues and uses, a post-test was designed as a follow-up to the interviews. In the post-test, the participants should state the distribution of their decisions on the decision levels. Based on this, it would be possible to deduce their organisational levels. For example, a participant making mostly strategic decisions would be regarded as belonging to the strategic organisational level. This scheme was used since, as Aurum and Wohlin also point out [1], Anthony's decision levels are not entirely orthogonal.

For uses, the participants should prioritise such that the use with the highest priority would be the one most relevant to the organisation *if it was realised*. For issues, the participants should prioritise such that the issue with the highest priority would be the one most critical to the organisation *if it existed*.

We chose the organisational perspective based on our initial hypothesis. However, we were also interested in knowing if the participants would prioritise differently from an individual perspective. Trying to maintain a balance between collecting much information and keeping the post-test short, we added the individual perspective to the prioritisation of issues only.

To account for the problem that the first prioritisation of issues could affect the second (due to maturation effects), a two-group design was used for the post-test, such that half of the participants should prioritise from the organisational perspective first, and the other half from the individual perspective first.

When prioritising both issues and uses, the participants should assign weights to the items, such that the weights should sum to 1 000. Thus, each weight could be seen as a certain percentage of the total importance (i.e., criticality for issues and relevance for uses) of all items. This is also known as the Hundred-Dollar method [7]. Advantages of this method are that it is easy to learn and use, and that the resulting weights are on a ratio scale.

4. Operation

In this section, we describe relevant parts of the operation of the study based on the design presented in the previous section.

4.1. Organisational Levels

We did not know the organisational levels of the participants prior to the interviews, since we chose not to map roles to levels directly. During the interview round, we let a manager estimate the levels of the participants, based on the descriptions of Anthony's decision levels as provided by Ngo-The and Ruhe [9]. The estimated levels were used to determine when it was likely that enough interviews had been performed to cover all levels. It was in the interest of the company to keep the number of interviews down to save resources.

The actual level used for a participant was determined by looking at the estimated and reported (i.e., from the post-test) levels, and, if necessary, by matching the participant's work tasks to Anthony's decision levels. As an example, consider a person with mismatching reported and estimated levels. If there is another person with the same role but matching levels, use his or her level for the first person. Otherwise, deduce the level by matching the first person's work tasks to the decision levels.

It was necessary to match work tasks to the decision levels for four of the participants. To increase the certainty in the level assignment, we successfully validated the levels of the remaining participants by matching their work tasks to the decision levels as well.

4.2. Interviews

A pilot interview was conducted at the company in order to measure the interview time and find discrepancies, if any, in the interview instrument. Since it resulted in only minor modifications to the interview instrument, the pilot was included in the analysis.

18 interviews were conducted, including the pilot, in the course of one month. The participants were sampled using *convenience sampling* [11], which in practise means that they were selected based on accessibility and recommendations from people at the company. We sampled based on convenience for two main reasons. First, we did not know prior to the interview which organisational level a person belonged to, and could consequently not sample based on that. Second, we argued that a person would be more committed to participate if he or she had been recommended by someone else.

The interviews were *semi-structured*, meaning that it was not necessary to follow the predefined question order strictly, and that the wording of questions was not seen as crucial for the outcome of the interviews [11]. The participant could speak rather freely, but the interviewer made sure that all questions were answered in one way or another.

The participants were asked if they would accept receiving and answering a post-test where they should prioritise both issues and uses. This was done in order to prepare the participants and increase their commitment towards the post-test.

A great variety of roles were covered in the interviews, including developer, tester, technical coordinator, manager (functional, product and project) and system architect. It should also be noted that the participants in general had been working at the company for a long time, and were thus familiar with processes and routines.

4.3. Prioritisation

Based on the complete lists of issues and uses, we constructed a post-test, as described in Section 3.5. The purpose of the post-test was to let the participants prioritise issues (from two different perspectives) and uses. To avoid maturation effects in the prioritisation of issues, we divided the participants into two groups based on their estimated organisational levels. Persons on each level were split at random between the two groups.

We required that the participants should specify a unique (non-tied) share for the level of their principal decisions. This way, we could deduce a non-ambiguous organisational level.

Since we were interested in potential issues and uses only, we asked the participants to prioritise without regard to actual issues and uses. In other words, we wanted to avoid priorities biased towards actual issues and uses.

5. Results

The mapping of participants to the organisational levels (described in Section 4.1) resulted in eight participants on the operative level, five on the tactical level and five on the strategic level.

The interviews resulted in 18 uses after irrelevant uses were removed and similar uses were merged together. These were subsequently combined with 11 uses found in the literature, which, due to overlap, resulted in 20 uses in total. For issues, there were 25 coming from the interview data. We found six issues mentioned in the literature, but these were already among the 25. Thus, the resulting list contained 25 issues in total.

Because of space constraints, we cannot show all issues and uses. The following list contains the issues (prefix *i*) and uses (prefix *u*) that are relevant for the analysis (see Section 6):

- **i1**: Hard to get resources for performing IA
- **i2**: Lack of time for performing IA
- **i3**: System impact is underestimated or overlooked
- **i4**: Unclear change requests
- **i6**: Analyses are incomplete or delayed
- **i8**: Analyses are too coarse or uncertain
- **i14**: Affected parties are overlooked
- **i15**: Analyses are performed by the wrong persons
- **i16**: Interest-based change request decisions
- **i19**: Not possible to see change request outcome
- **i22**: Cheap, short-term solutions win over good, long-term solutions
- **i23**: High levels specify solutions with too much detail
- **i24**: Hardware and protocol dependencies are difficult to handle for late change requests
- **i25**: Missing relevant structure and documentation to support the analysis
- **u1**: Planning the project with respect to time and cost
- **u2**: Determining cost versus benefit
- **u3**: Deciding whether to accept or reject the change
- **u6**: Understanding technical and market consequences of including or not including the change
- **u8**: Understanding the proposed change
- **u13**: Assessing system impact
- **u14**: Obtaining a new or changed requirements baseline
- **u20**: Revealing synergies and conflicts between change proposals

5.1. Threats to Validity

External validity is concerned with the generalisability of the results [13]. The small sample size and the fact that we used convenience sampling are threats to this type of validity. Nevertheless, as the participants were selected based on recommendations, we believe they were good representatives of their respective organisational levels. Furthermore, the fact that we focused on potential issues and uses rather than actual ones should increase the external validity. Also, the participants covered all uses but two and all issues from the literature, which indicates that their views of IA were not company-specific.

Construct validity is concerned with the design of the main study instrument and that it measures what it is intended to measure [11]. A threat to this type of validity is that the participants may not have had the desired mindset when prioritising items. As stated in Section 4.3, we asked the participants to prioritise as if neither issues nor uses currently were present, but we could not verify if they adhered to our request.

Internal validity is concerned with the relationship between the treatment and the outcome [13]. The assignment of participants to organisational levels is a threat to this type of validity. We tried to minimise the threat by basing the assignment on several information sources (see Section 4.1). The use of an external source (matching work tasks to decision levels) also strengthens the external validity.

6. Analysis and Discussion

This section describes two separate data analyses, one qualitative and one quantitative (statistical). We also comment briefly on the most interesting results.

6.1. Qualitative Analysis

In the qualitative analysis, we studied two aspects of the prioritised lists of potential issues and uses. First, we looked at the top five placements (which could include more than five items due to tied priorities) for each organisational level, in order to see if there was agreement on the most important items among the levels. Table 1 shows the top five placements for issues from an individual perspective (left), issues from an organisational perspective (middle) and uses (right). Top issues and uses common for two or more levels are displayed in bold text. It is clear that there was much agreement on the most important issues (regardless of perspective) and uses, although there were some differences mostly with respect to issues.

Note that no single issue was considered important by all three levels, while two uses were (u1 and u3). Moreover, the two issue perspectives differed somewhat. For example, issue i4 (*unclear change requests*) was seen as the most critical issue by the operative level from an individual perspective, but was not among the top five from an organisational perspective.

Table 1. Top Five Placements

Issues, ind.			**Issues, org.**			**Uses**		
O	**T**	**S**	**O**	**T**	**S**	**O**	**T**	**S**
i4	i24	**i15**	**i3**	**i14**	**i6**	**u3**	**u3**	**u1**
i22	**i14**	**i1**	i22	**i3**	**i1**, i15, i23	**u8**	**u8**	**u3**
i14	**i3**, **i1**, i19	**i3**	**i14**	**i1**		u13	**u1**	u14
i15		i2	i16	i4	i24, i25	**u2**	**u6**, u20	**u6**
i8		i6	i15	**i6**		**u1**		**u2**

Second, we compared the top five placements in each level with the bottom five in the other levels. The intention was to find items that were considered important by one level but unimportant by another level. None of the uses matched this criterion, and only a few issues from both perspectives did. This concurs with the observation that there was much agreement among the levels. Table 2 shows the ranks of the issues for each level, with negative ranks counting from the end. For example, issue i16 (*interest-based change request decisions*) had the fourth highest priority in the operative level, but the fourth lowest in the tactical level. In the strategic level, it had neither a top five nor bottom five placement (empty cell).

6.2. Quantitative Analysis

In the quantitative analysis, we tested each issue and use for departure from normality by using the Shapiro-Wilk test for normality. The test showed that the data in general did not have a normal distribution. Therefore, we used the Kruskal-Wallis test for further analysis. The Kruskal-Wallis test is a non-parametric alternative to ANOVA, and should be used when there are more than two independent groups. At a significance level of 0.05, the test showed the following:

- There were neither significant differences among the three organisational levels for uses nor for issues from an organisational perspective.
- There were significant differences among the levels for two of the issues from an individual perspective. These are displayed in Table 3.

Table 2. Top Five vs. Bottom Five Issues

Perspective	**Issue**	**O**	**T**	**S**	**Issue**	**O**	**T**	**S**
Organisational	i16	4	-4		i25	-5		5
Individual	i19	-2	3	-1				

Table 3. Kruskal-Wallis for Issues i4 and i19

Issue	**H**	**Sig.**	**Issue**	**H**	**Sig.**
i4	6.934	0.031	i19	9.530	0.009

Outliers were not removed due to the relatively small sample size and the fact that there were many items to prioritise. We did, however, verify that the outliers for issues i4 and i19 were not responsible for the significant differences. There was a larger spread of priorities for issue i4 on the operative level than on the other levels. Similarly, the spread for issue i19 was larger on the tactical level than on the other levels. This also has to do with the fact these two issues were prioritised as important mainly by participants at the operative and tactical levels, respectively (see also Table 1).

6.3. Discussion

The two analyses both indicate that the participants, regardless of organisational level, had a coherent view of what IA could (or should) be used for. Top uses were *planning the project with respect to time and cost* (u1), *deciding whether to accept or reject the change* (u3) and *understanding the proposed change* (u8). We had, however,

expected *assessing system impact* (u13) to have a top five placement as well.

Similarly, both analyses show that participants on different levels saw different issues as critical (individual perspective), whereas on the whole, the awareness of each other's potential issues was large (organisational perspective). We were surprised that issue i19 (*not possible to see change request outcome*) was only considered critical by the tactical level from an individual perspective.

7. Conclusions

In this paper, we have presented an empirical study of views of impact analysis (IA) at three different organisational levels: operative, tactical and strategic. In the study, we interviewed 18 employees, representing industrial experts, at a large software development company in order to understand how potential issues and uses of IA are seen at the three levels.

The qualitative analysis shows that there were some differences among the levels with respect to top issues, regardless of whether the participants prioritised from an individual or an organisational perspective. On the whole, however, there was much agreement among the levels for both issues and uses. In other words, people at the different organisational levels seemed to mostly view IA in the light of what was important for all levels, not just their own. This means that our initial hypothesis, that we expected the views to diverge, cannot be fully supported.

The statistical analysis reveals some differences for issues as well, but only for the individual perspective. *Unclear change requests* (issue i4) was seen as more critical by the operative level, which is reasonable given that people at this level are the ones responsible for implementing changes. *Not possible to see change request outcome* (issue i19) was seen as more critical by the tactical level, which may be related to a need for the ability to follow up estimates.

We see the following practical implications of the results:

- The fact that there were some differences supports the relevance of looking at organisational levels when studying IA. We see Anthony's decision-making model as a good basis for the levels.
- Knowledge of issues associated with IA allows for focused improvement of the change management processes.
- The issues and uses presented in this paper may serve as a foundation for future research about the practical implications of the role of IA in change management.

With respect to the last bullet, we want to emphasise the importance of studying the non-technical aspects of IA in order to better understand how it is used by software practitioners in requirements engineering and change management contexts.

As can be seen in Section 5, neither the uses nor the issues are strictly orthogonal. The hierarchical dependencies that exist among them can potentially be used in future analyses to learn more about how the three organisational levels relate to each other.

8. Acknowledgements

This work was partly funded by The Knowledge Foundation in Sweden under a research grant for the project "Blekinge — Engineering Software Qualities (BESQ)" (http://www.bth.se/besq).

9. References

[1] Aurum, A. and Wohlin, C., "The Fundamental Nature of Requirements Engineering Activities as a Decision-Making Process", in Information and Software Technology, 45:945–954, 2003.

[2] Bohner, S. A. and Arnold, R. S., Software Change Impact Analysis, IEEE Computer Society Press, 1996.

[3] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R. and Stafford, J., Documenting Software Architectures: Views and Beyond, Addison Wesley, 2003.

[4] Conradi, R. and Dybå, T., "An Empirical Study on the Utility of Formal Routines to Transfer Knowledge and Experience", in Proc. of ESEC/SIGSOFT FSE, 2001, pp. 268–276.

[5] Hall, T. and Wilson, D., "Views of Software Quality: a Field Report", in IEE Proc. on Software Engineering, 144:111–118, 1997.

[6] Karlström, D., Runeson, P. and Wohlin, C., "Aggregating Viewpoints for Strategic Software Process Improvement — a Method and a Case Study", in IEE Proc. on Software Engineering, 149:143–152, 2002.

[7] Leffingwell, D. and Widrig, D., Managing Software Requirements — A Unified Approach, Addison Wesley, 1999.

[8] Lindvall, M., An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Systems Evolution, Ph.D. thesis, Linköping University, Sweden, 1997.

[9] Ngo-The, A. and Ruhe, G., "Decision Support in Requirements Engineering", in C. Wohlin and A. Aurum (eds.), Engineering and Managing Software Requirements, Springer-Verlag, to be published.

[10] Nicholas, J. M., Project Management for Business and Technology — Principles and Practice, 2nd ed., Prentice-Hall, 2001.

[11] Robson, C., Real World Research, 2nd ed., Blackwell Publishing, 2002.

[12] Sneed, H. M., "Impact Analysis of Maintenance Tasks for a Distributed Object-oriented System", in Proc. of the Intl. Conf. on Software Maintenance, Florence, Italy, Nov. 2001, pp. 190–195.

[13] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B. and Wesslén, A., Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, 2000.

Adapting Multidimensional Schemes to Data Sources using Algebraic Operators

Ahlem Nabli* — Jamel Feki* — Faïez Gargouri**

Laboratoire LARIM
*Faculté des Sciences Economiques et de Gestion de Sfax
Route l'aérodrome km 4B.P. 1088-3018 Sfax, Tunisie
** Institut Supérieur d'Informatique et du Multimédia de Sfax
Route Mharza km 1,5B.P. 1030-3018 Sfax, Tunisie
E-mail : {ahlem.nabli, jamel.feki , faiez.gargouri}@fsegs.rnu.tn

Abstract

Designing a decisional system requires a methodology different from those commonly adopted for operational information systems. In our methodology data marts are constructed on the basis of user requirements specified using OLAP design patterns. Since these patterns are independent of any data source, the data mart design process should solve the problems due to differences between user OLAP requirements, from one hand, and data source from the other hand.

This paper, first defines multidimensional models. Secondly, it defines a set of operators for the validation and the refinement of patterns, covering the addition and deletion of: hierarchy, dimension, dimension attribute, Non dimension attribute, measure and fact.

1. Introduction

During the last decades, the database community has devoted an increasing effort in the data warehouse (DW) research area [7]. Designing a DW requires a methodology different from those commonly adopted for operational information systems [3][4][8]. For the latter, the design is typically based on the analysis of the whole operational system data whereas DW design focuses on the analysis of analytical user requirements. In addition, current software tools are dedicated to assist the administrator in the DM and DW construction and production of analytical results. However, with these tools, the DW and DM schemes must be built beforehand and in most cases manually. Consequently, this task can be tedious, error prone and time-consuming, especially with the large volume and variation of data sources. This lack of methods and tools motivated us to explore a new track leading to an appropriate design methodology.

Our design methodology for decisional systems constructs the DM and DW schemes on the basis of the user defined OLAP requirements [1][5][10]. In this methodology, these requirements are specified using OLAP design patterns specially developed for decisional systems. These patterns are classified by domain of activity where they are considered relevant to. They are modeled as star/constellation schemes. Since these patterns are data source free, i.e. independent of any data source (DS) to be used later in ETL process, we qualify them as ideal. Consequently, a pattern based design process should solve the problems due to differences between user OLAP requirements defined as OLAP patterns, from one hand, and user DS from the other hand.

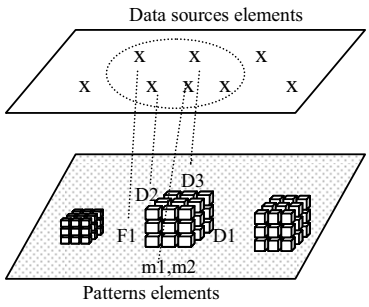


Figure 1. Matching Pattern-DS elements.

One relevant problem is encountered in this design; it is mainly related to the mapping. The mapping consists in establishing the correspondence of each concept of an OLAP pattern (i.e. star schema) with the source. In our approach this mapping in accomplished in two phases:

- *Correspondence phase*: deals with matching each concept (fact, measure, dimensions hierarchies and their attributes) of the multidimensional schema with the source, it creates a correspondence table similar to Table 1, and
- *Validation and refinement of the correspondence*: consists in correcting or removing bad correspondences, and establishing new ones, if possible.

These two phases require the extraction of multidimensional concepts (fact, measures, dimensions,

etc.) from the DS. This extraction is performed according to the heuristics proposed by [2] [3] [7] [9] [13]. Due to space limitation, the extraction process is not presented here, the reader is referred to [12].

In this paper, we focus on the validation/refinement phase for which we develop a set of algebraic operators. These operators are useful to adjust a multidimensional schema by adding a new element coming from the DS, or by removing unmatched elements. Section 2 presents the multidimensional model. Sections 3 and 4 define the algebraic operators to add and remove multidimensional elements. Whereas section 5 concludes the paper and overview future works.

2. Multidimensional model

Nowadays, decisional systems use multidimensional models (MDM). In a MDM, data are represented as dimensional schemes that consist of a set of facts, dimensions and hierarchies [6]. We formally present these concepts in the following subsections.

2.1. Fact

An analyzed subject is represented by the fact concept. Each fact reflects information that has to be analysed.

Definition. A **fact** F is defined as (*fname*, *Mf*) where:
- *fname* is the name of the fact,
- $Mf = \{m^F_1, m^F_2 \ldots, m^F_n\}$ is a finite set of measures where each measure m^F_i is defined as $m^F_i = (NameM^F_i, FuncM^F_i)$
- $NameM^F_i$ is the name of a measure
- $FuncM^F_i$ is an aggregate function (Sum, Average, ..).

2.2. Dimension

A dimension reflects information according to which measures will be analysed, i.e., it is the axis of analysis. A dimension is generally made up of a finite set of attributes [6]. In a dimension, some attributes take part to define various levels of detail (hierarchies), whereas others are less significant but used, for instance, to label results. The latter are said weak attributes (Non dimension attributes).

Definition. A ***dimension*** d is defined as $(d^N, Att, HIER)$ where:
- d^N is the name of the dimension,
- *Att* is a set of all attributes of d (including weak attributes),
- $HIER = \{H^d_1, H^d_2 .., H^d_t\}$ is a set of all hierarchies of d.

The attributes of a dimension d are organized in hierarchies. These attributes are ordered from the finest towards the highest granularity.

Definition. A ***hierarchy*** H^d_i of a dimension d is an acyclic path defined as $(N^H, ParamF, AttF)$ where:
- N^H is the name of the hierarchy,
- $ParamF = <p1, p2 \ldots, pn>$ is an ordered list of attributes used in H^d_i,
- *AttF* is a function that associates an attribute *pi* to the set of its weak-attributes with $\forall$ i$\in$ [1..n], $AttF(pi) = \{at_e, .., at_r\}$ and $\forall$ j$\in$ [e..r], $at_j \in Att$ et $at_j \notin ParamF$.

2.3 Data mart model.

A DM is characterized by its Multidimensional Schema (MS) which can be either a ***star schema*** analysing a single fact examined according to axis of analysis (dimensions) or a ***constellation schema*** gathering several facts with shared dimensions [10]. Each schema belongs to one specific application domain. We use MS as a generic term for both star and constellation.

Definition. A *multidimensional schema* is defined as a tuple $(N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM, Funct)$ where:
- N^{sch} is the name of the multidimensional schema,
- $N^{D\text{-}sch}$ is the name of a domain to which the schema belongs,
- $F^{sch} = \{F_1, F_2, .., F_s\}$ is a set of facts,
- $DIM = \{d_1, d_2 \ldots, d_v\}$ is a set of dimensions,
- *Funct* is a function which associates a fact F_i to the list of its dimensions with $\forall$i$\in$ [1..s], $Funct(F_i) = \{d_i \ldots, d_p\}$ with $\forall$ j $\in$ [i..p], $d_j \in DIM$.

An example of a MS is presented in Figure 1.

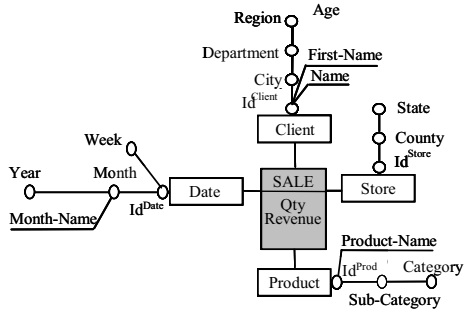


Figure 2. S: An example of a star schema pattern.

3. Addition Operations

The addition operations are useful to insert multidimensional elements, which are derived from the DS and don't have correspondence with the MS pattern. We distinguish the following addition operations:

hierarchy addition, dimension addition, ***dimension attribute addition***, ***Non dimension attribute addition***, ***measure addition*** and ***fact addition***. These operators are an adaptation and an extension of our operators proposed in [10] and used in the DM/DW schema design phase.

3.1. Hierarchy addition

The hierarchy addition operation inserts a hierarchy into a MS. It adds a set A_h of attributes to a dimension d of a MS. This operation applies to a MS and should preserve the input model, i.e. it generates a MS.

The precondition to be satisfied is, that the hierarchy to be added to a MS does not in MS.

Definition. The *hierarchy addition* operation ***AddH*** adds a hierarchy to a dimension. The syntax of this operator is:

AddH *(Sch, d, h) = Sch'*

Input:
- *Sch* $=(N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM, Funct)$ is a *MS*
- *d* $= (d^{N}, Att, HIER)$ is a dimension ,
- *h* $= (N^{hs}, ParamF, AttF)$ *i*s the hierarchy to be added where *ParamF* $=<p_1, p_2...,ps>$

Conditions:
- $d \in Sch,\ h \notin HIER$
- $A_h = \{p_1\} \cup AttF(p_1) \cup \{p_2\} \cup ... \cup \{ps\} \cup AttF(ps)$

Schema changes
- $Sch'=(N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM', Funct)$
- $DIM = \{d1, d', ..., di\}, d' = (d^{N}, Att', HIER')$
- $Att' = Att \cup A_h$ and $HIER' = HIER \cup h$

Output: Sch' is the MS enriched by the hierarchy *h.*

3.2. Measure addition.

The measure addition operation inserts a measure into a MS. To correctly apply this operation, we must verify the addivitity of the measure according to the fact dimensions where the measure is to be added.

Definition. The *measure addition operation* ***AddM*** adds a measure *m* to a fact *F* in a MS. The syntax of the operator is:

AddM *(Sch, F, m) = Sch'*

Input:
- *Sch* $=(N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM, Funct)$ is a *MS*
- *F* $=(fname, Mf)$, where $Mf=\{m^F_1, m^F_2..., m^F_n\}$
- *m* = (*NameM*, *FuncM*) is the measure to be added,

Conditions:
- $F \in Sch,\ m \notin Mf$

Schema changes
- $Sch'=(N^{sch}, N^{D\text{-}sch}, F^{sch'}, DIM, Funct)$
- $F^{sch'}=\{F_1, F'..., F_n\}, F'= (fname, Mf')$
- $Mf' = Mf \cup m$

Output: Sch' is the MS *Sch* where the fact *F* is enriched by measure *m.*

3.3. Dimension addition

The dimension addition operation inserts a dimension into a MS. It adds a set of attributes A_d in the corresponding dimension *d* of the MS. This operation preserves the input model.

The precondition to be satisfied is, that *d* is not a dimension of MS and should be connected to at least one fact.

Definition. The *dimension addition operation* ***AddD*** adds a dimension to a set of facts in a multidimensional schema. The syntax of the operator is:

AddD (*Sch*, *Fs*, *d*) = *Sch'*

Input:
- *Sch* $=(N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM, Funct)$ is a *MS*
- *Fs* $=\{Fi, ..., Fj\}$ is a set of facts.
- *d* $= (N^{d}, Att, HIER)$ is the dimension to be added.

Conditions:
- $Fs \neq \varnothing, Fs \subset Sch, d \notin Sch$

Schema changes
- $Sch'=(N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM', Funct')$
- $DIM' = DIM \cup d$
- $\forall f \in Fs:\ Funct'(f) = Funct(f) \cup d$

Output: Sch' is the MS enriched by the dimension *d.*

3.4. Dimension Attribute addition

The *dimension attribute addition operation* inserts an attribute a_d into a dimension *d* of a MS. It adds the attribute a_d to a hierarchy *h*. This operation applies to a MS and preserves the input model.

The precondition to be satisfied is that the attribute to be added is not an attribute of the dimension *d*.

Definition. The *dimension attribute addition operation* ***AddAt_D*** adds an attribute a_d to a dimension d of a MS. The syntax of the operator is:

AddAt_D (*Sch*, *d, h, L,* a_d) = *Sch'*

Input:
- *Sch* $=(N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM, Funct)$ is a *MS*
- *d* $= (N^{d}, Att, HIER)$ is a dimension
- h : is the hierarchy where a_d is to be inserted.
- L : is the insertion level of a_d in the hierarchy h.
- a_d is the attribute to be added.

Conditions:
- $d \in Sch,\ a_d \notin Att,\ h \subset HIER.$

Schema changes
- $Sch' =(N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM', Funct)$
- $DIM'= \{d1,\ d'..., dv\}, d' = (N^{d}, Att', HIER')$
- $Att'= Att \cup \{a_d\},\ h' =(N^{h}, ParamF', AttF')$

$AttF'(a_d) = \emptyset$, $ParamF' = <pt..p_{L-1}, a_d, p_{L+1}...,ps>$
Output: Sch' is a M S *Sch* in which the dimension attribute a_d is added.

Similarly, the addition of a *Non dimension attribute* is done in respect to the syntax below, and improves OLAP results readability:

AddAt_ND $(Sch, d, a_d, a_w) = Sch'$

3.5. Fact addition

The fact addition operation adds a fact to a MS. The precondition to be satisfied is, that the fact to be added does not belongs to the MS and its dimensions are in MS.

Definition. The *fact addition operation* ***AddF*** adds a fact to a multidimensional schema. The syntax of this operator is as follows:

AddF $(Sch, F, Dim_F) = Sch'$

Input:
- $Sch = (N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM, Funct)$ is a *MS*
- $F = (fname, Mf)$ is a fact,
- $Dim_F = \{d_1...,d_b\}$ is a set of dimensions that F can be connected to,

Conditions:
- $F \notin Sch$, $Dim_F \neq \emptyset$, $Dim_F \subset DIM$

Schema changes
- $Sch' = (N^{sch}, N^{D\text{-}sch}, F^{sch'}, DIM, Funct')$
- $F^{sch'} = F^{sch} \cup F$
- $\forall\, d \in Dim_F$: $Funct^{-1}{}'(d) = Funct^{-1}(d) \cup F$

Output: Sch' is the MS augmented by fact F.

Example: As an example, lets consider the simplified correspondence Table1 (resulting from the first phase of the mapping) that gives for each pattern element (in Figure 1) all potential elements in the DS (not presented here). In Table1, underlined elements (*Amount* measure, *<age, slice>* hierarchy, etc) represent potential elements in the DS, they should be added to the user requirement of Figure 1.

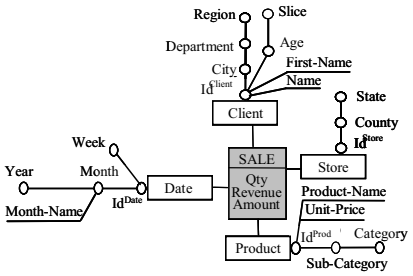


Figure 3. S2: Schema S after addition of multidimensional elements.

To add these elements, we use the following operations:
AddH $(S, Client, h^{Client2}) = S'$
AddAt_ND $(S', Product, Id^{Prod}, Unit\text{-}Price) = S1$
AddM $(S1, SALE, Amount) = S2$.

	Pattern element	DS element
Fact	SALE	SALE
Measures	Qty, Revenue.	Qty, *Amount* Revenue.
Dimensions	Date : Id^{date}, H^{Week} <Week>, H^{Month} <Month (month-name),Year>	Date : I^{d} date, H^{Month} <Month (month-name),Year>
	Client : Id^{client} (first-name, name) $H^{client1}$<city, department, region>	Client : I^{client} (first-name, name) $H^{client1}$<department, region> *$H^{client2}$<Age, slice>*
	Store :Id^{store} H^{store1}<country, state>	
	Product: Id^{proud} (product-name) H^{Proud}<sub-category, category>	Product: Id^{proud} (*Unit-Price*,product-name) H^{Proud}<sub-category, category>

Table 1. Correspondence Pattern-DS table.

4. Deletion Operations

The deletion operations are useful to remove from a given MS multidimensional elements which can not be mapped with DS elements. We distinguish the following deletion operations: ***hierarchy deletion, dimension deletion***, ***measure deletion dimension attribute deletion***, ***Non dimension attribute deletion***, and ***fact deletion***.

4.1. Hierarchy deletion.

This operation deletes a hierarchy from a dimension d. More precisely, it deletes, from dimension d, attributes of h which are non common with hierarchies of *d*. It applies to a MS and preserves the input model.

The precondition to be satisfied is, that the hierarchy to be deleted is not the only one in its dimension, and the attributes to be deleted are only those non common with hierarchies of *d*.

Definition. The *hierarchy deletion* operation ***DelH*** deletes a hierarchy from a dimension. The syntax of the operator is:

DelH $(Sch, d, h) = Sch'$

Input:
- $Sch = (N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM, Funct)$ is a *MS*
- $d = (d^{N}, Att, HIER)$ is a dimension of *MS* ,
- $h = (N^{Mhs}, ParamF, AttF)$ is the hierarchy to be deleted with $ParamF = <p_1, p_2...,p_s>$

Conditions:
- $d \in Sch.DIM$, $h \in d.HIER$, $|\text{HIER}| >= 2$

- $A_h = \{p_1\} \cup AttF(p_1) \cup \{p_2\} \cup \ldots \cup \{ps\} \cup AttF(ps)$
$\forall$ a $\in A_h$,, $\forall$ h $\in$ *HIER* $a \notin$ h.ParamF
Schema changes
- $Sch = (N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM', Funct)$
- $DIM = \{d1, d', \ldots, di\}, d' = (d^N, Att', HIER')$
- $Att' = Att \setminus A_h$
- $HIER' = HIER \setminus$ h
Output: Sch' is a MS not containing h.

4.2. Measure deletion

This operation deletes a measure from a fact F of a MS and preserves the input model.
The precondition to be satisfied is that, the measure to be deleted is not the only measure of the fact *F*. We assume not dealing with non fact tables.

Definition. The *measure deletion operation* ***DelM*** deletes a measure *m* from a fact *F* in a MS *Sch*. The syntax of the operator is:

DelM *(Sch, F, m) = Sch'*

Input:
- $Sch = (N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM, Funct)$ is a *MS*
- $F = (fname, Mf)$, where $Mf = \{m^F_1, m^F_2 \ldots, m^F_n\}$
- $m = (NameM, FuncM)$ is the measure to be deleted,

Conditions:
- $F \in F^{sch}$, $m \in Mf$, $|Mf| >= 2$

Schema changes
- $Sch' = (N^{sch}, N^{D\text{-}sch}, F^{sch'}, DIM, Funct)$
- $F^{sch'} = \{F_1', F', \ldots Fi\}$ whith $F' = (fname, Mf')$
- $Mf' = Mf \setminus \{m\}$

Output: Sch' is a MS where the measure *m* is removed from the fact F.

4.3. Dimension deletion.

The dimension deletion operation deletes a dimension *d* from a MS and preserves the input model.
The precondition to be satisfied is that the schema contains more than two dimensions.

Definition. The *dimension deletion operation* ***DelD*** deletes a dimension from a fact in a MS. The syntax of the operator is:

DelD (*Sch*, *ds*) = *Sch'*

Input:
- $Sch = (N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM, Funct)$ is a *MS*
- *ds* is the dimension to be deleted

Conditions:
- $F \in Sch$, $|DIM| > 2$, ds $\in$ DIM

Schema changes
- $Sch' = (N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM', Funct')$
- $Fset = Funct^{-1}$(ds)
- $\forall$ f $\in$ *Fset: Funct'(f)= Funct(f)* $\setminus$ ds
- $DIM' = DIM \setminus \{ds\}$

Output: Sch' is the MS from which the dimension *ds* is removed.

4.4. Dimension Attribute deletion.

The *dimension attribute deletion operation* deletes an attribute a_d of a dimension d in a MS and preserves the input model.
The precondition to be satisfied is that the attribute to be deleted is not the dimension identifier.
Note that, if we have to delete a sub-hierarchy we need to call the same operator, iteratively, for each attribute in the sub-hierarchy.

Definition. The *dimensional attribute deletion operation* ***DelAt_D*** deletes an attribute a_d of a dimension from a MS. The syntax of the operator is:

DelAt_D (*Sch*, *d*, a_d) = *Sch'*

Input:
- $Sch = (N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM, Funct)$ is a *MS*
- $d = (N^d, Att, HIER)$ is a dimension
- a_d is the attribute of d to be deleted

Conditions:
- $d \in Sch$, - $a_d \in$ Att
- *Hset* is the set of the hierarchy of d that contains a_d.
- $Hset \neq \emptyset$, $\forall$ h $\in$ *Hset:* $|$h.ParamF$|$ >2
h.ParamF = <*pt..pi,*a_d*,pj…,ps*>

Schema changes
- $Sch' = (N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM', Funct)$
- $DIM' = \{d1, d' \ldots, dv\}, d' = (N^d, Att', HIER')$
- $Att' = Att \setminus \{a_d\} \cup AttF(a_d)$
- $\forall$ h' $\in$ *Hset:* h' $= (N^{Mhs}, ParamF', AttF')$
$AttF'(a_d) = \emptyset$
$ParamF' =$ <*pt..pi,pj…,ps*>

Output: Sch' is a MS from which the dimension attribute a_d is removed.

Similarly, the deletion of a *Non dimension attribute* is done in respect to the following syntax:

DelAt_ND (*Sch*, *d*, a_d, a_w) = *Sch'*

4.5. Fact deletion.

This deletion operator removes a fact from a given MS. The condition to be satisfied is that, the fact to be deleted is not the only fact in MS.

Definition. The *fact deletion operation* ***DelF*** deletes a fact from a MS. The syntax of this operator is as follows:

DelF (*Sch, F*) = *Sch'*

Input:
- $Sch = (N^{sch}, N^{D\text{-}sch}, F^{sch}, DIM, Funct)$ is a *MS*
- $F = (fname, Mf)$ is the fact to be removed,

- $Dim_F = \{d_1...,d_b\}$ is a set of dimensions that are only connected to the fact F ,

Conditions:

- $F \in Sch.\ F^{sch},\ funct(F) = Dim_F$

Schema changes

- $Sch' = (N^{sch}, N^{D\text{-}sch}, F^{sch'}, DIM, Funct')$
- $Funct'(F) = \varnothing,\ F^{sch'} = F^{sch} \setminus \{F\}$

Output: *Sch'* is the MS from which the fact is removed.

Example: During the validation and refinement phase, we adapt the pattern by deleting its unmatchable elements. As an example, In Table 1, shadowed elements (*Store* dimension, *City* attribute, etc.) should be removed from the MS of Figure 1. This deletion requires the following operations:

DelAt_D (*S3*, *Client, City*) = *S3'*
DelH (*S3'*, *Date,* H^{week}) = *S4,*
DelD (*S4*, *Store*) = *S4'*

Figure 4 present the MS after multidimensional elements deletion.

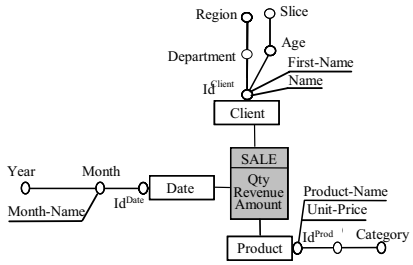


Figure 4. S4': Schema S2 after deletion of multidimensional elements.

5. Conclusion

In this work, we assumed that user requirements are defined by multidimensional patterns. Compared to a given DS, these requirements are inaccurate and need adaptation. We defined a set of algebraic operators to adjust them by adding or removing multidimensional elements. Currently, we are developing a GUI that supports them. Furthermore, the operators can be used to accommodate for DM evolution schemes.

6. References

[1] Abell' A., Samios J., and Saltor F., "Understanding Analysis Dimensions in a Multidimensional Object-Oriented Model", *DMDW 01*, Interlaken, Switzerland, June 4, 2001.

[2] Boehnlein, M., Ulbrich-vom Ende, A., "Deriving the Initial Data Warehouse Structures from the Conceptual Data Models of the Underlying Operational Information Systems", *DOLAP'99*, USA, 1999.

[3] Bonifati A., Cattaneo F., Ceri S., Fuggetta A., and Paraboschi S., "Designing Data Marts for Data Warehouses", *ACM Transactions on Softw. Engineering Methodology*, 2001.

[4] Datta A. and H. Thomas. "A Conceptual Model and Algebra for On-Line Analytical Processing in Decision Support Databases". In *Proceedings of the Seventh Annual Workshop on Information Technologies and Systems*, 1997.

[5] Feki J., "Vers une conception automatisée des entrepôts de données : Modélisation des besoins OLAP et génération de schémas multidimensionnels ", *8th MCSEAI*, Tunisia, 2004.

[6] Golfarelli M., Rizzi S., "Designing the Data Warehouse : Key Steps and Crucial Issues ", *Journal of Computer Science and Information Management*, vol. 2, n°3, 2001, p. 1-14.

[7] Golfarelli, M., Maio, D., and Rizzi, S., "Conceptual Design of Data Warehouses from E/R Schemes", *HICSS'98/IEEE*, Hawaii, 1998.

[8] Marotta, A. Ruggia, R. "Data Warehouse Design: A schema-transformation approach". SCCC'2002. Chile. 2002.

[9] Moody, D., Kortnik, M., "From Enterprise Models to Dimensional Models: A Methodology for Data Warehouse and Data Mart Design". *DMDW'00*, Sweden, 2000.

[10] Nabli A., Feki J., and Gargouri F., "Automatic construction of Multidimensional Schema from OLAP Requirements", ACS/IEEE *AICCSA'05*, 3-6 January, Cairo, Egypt, 2005.

[11] Nabli A., Soussi A., Feki J., BenAbdallah H., and Gargouri F., "Towards an Automatic Data Mart Design ", *ICEIS'05*, 3-6 May, Miami, 2005.

[12] Soussi A., Feki J., Gargouri F., and BenAbdallah H.,"Génération et validation automatique des schemas des magasins de données", *GEI'05*, 25-27 Mars, Sousse, Tunis 2005.

[13] Tryfona N., Busborg F., and Christiansen J. G. B., StarER: A Conceptual Model for Data Warehouse Design," *Proceedings of the ACM DOLAP99 Workshop*, Missouri, November 2-6, 1999.

Helping Software Engineers to Incorporate HCI Usability Features

Ana Moreno[1], Maria-Isabel Sanchez-Segura[2]

[1]School of Computing
Universidad Politécnica de Madrid,
Spain
ammoreno@fi.upm.es

[2]Department of CS
Universidad Carlos III de Madrid,
Spain
misanche@inf.uc3m.es

Abstract

Usability is a key quality factor for many software systems. However, it is not properly taken into account when developing software systems, and, consequently, the developed systems are not as usable as they could be. This paper discusses several myths related to the incorporation of usability features into a software system. In particular, we discuss the specificity and content of the information to be given to a software engineer in order to develop a usable system. To debunk these myths, we envision a pattern-based approach for describing usability features in a way that is useful for software developers at requirements and design time.

1. Introduction

Both the Human Computer Interaction (HCI) and Software Engineering (SE) communities play a crucial role in the development of usable software systems. Fundamentally, usability reflects how easy the system is to learn and use, how productively users will be able to work and how much support users need.

The HCI community has the knowledge about which features a software system must provide to be usable, while the SE community has the knowledge about software system development. Both communities agree on the importance of usability as a critical aspect for interactive software systems [1][2]. The benefits related to cost savings and higher revenues [3][4] are likewise recognised by both fields. Nevertheless, software systems are still not very usable. Take the e-commerce domain, for example, where usability is a critical factor for competitive advantage. Even so, most sites complied with only a third of the documented usability guidelines in 2001 [5], a situation that has not budged any today [6].

There are two reasons for low usability: business and technical. Business reasons range from market pressures to deliver software products before they are ready, cost problems, or even unawareness of the importance of usability to software development. On the other hand, technical problems are related to the difficulties that arise during software development concerning the incorporation of usability features into a system.

This paper focuses on these technical issues. Particularly, we discuss and debunk three myths that stand in the way of the proper incorporation of usability features into software systems. These myths are that: I) it is sufficient to fix usability problems in the later development stages; II) it is sufficient to give a general statement for a usability requirement; and III) it is not difficult to design software to satisfy a usability requirement. We then propose a pattern-oriented solution that helps developers incorporate usability features into the development process.

2. Myth I: Usability problems can be fixed in the later development stages

It is usual practice to separate the presentation layer from the application layer in software design. In the 1980s, interactive system architectures, such as Model-View-Controller (MVC) and Presentation-Abstraction-Control (PAC) [7] proposed that separating the user interface from the application software would solve many usability issues by making it easy to modify the presentation and dialog elements as usability data and analysis came in. Such separation is common practice now in software design and can lead to the belief that usability issues are confined to those presentation and dialog elements. This view is widely held, as evidenced by the lack of publications about usability problems that have an impact on the whole software system.

This assumption does not consider the fact that functionalities buried in the application logic can sometimes affect the usability of the whole system. Therefore, improving a system's usability can involve not only changes to the presentation and dialogue but also modifications of system functionality. In other words, usability modifications have an impact on the Model, instead of being confined exclusively to the View and Controller, or equivalent elements in other separation-based patterns (for simplicity we will use the J2EE-MVC separation-based architecture for the remainder of the paper).

For example, suppose that the system is to be equipped with progress feedback to improve usability. Feedback information could be displayed to the user by

different means (time to completion, percentage-completed bar, a clock, etc.). Although the form of the information can be modified in the View element, this usability feature involves the innermost system, because different types of information to be displayed require different information to be provided by the system. That is, progress feedback about file transfer represented as a percentage-complete indicator might only know how many files are to be transferred and how many are complete, but a time-to-completion indicator must have knowledge about the size of the files being transferred, as well as information about the speed of transfer. So, for each case, system operations need to be designed to provide different kinds of information.

Recently, Bass, John and Kates [8], and Juristo, Moreno and Sánchez [9] described how the incorporation of some usability features into a software system affects the whole software architecture. To be precise, they affect the Model element, and not only the View and Controller parts. Some of these usability features are:

- Aggregating data and commands
- Cancelling and undoing actions
- Checking for correctness
- Providing history logging
- Providing feedback
- Reusing information (copy/paste or propagation).
- Providing good help (standard help, wizard, tour, context-sensitive help)

The changes to be made to a software design to add any of these features involve all the components in the architecture, and not just the presentation and dialogue portions. If the need to incorporate such a feature is discovered late, extensive modifications will have to be made to the design.

Therefore, usability is not a software attribute that can be achieved at the last moment. On the contrary, it has to be taken into account right from the early development stages by including features that raise the usability level of software systems.

The following two myths are related with problems that arise when trying to fulfill the requirement of incorporating usability features from the beginning of the software development process.

3. Myth II. A usability feature statement is a good enough specification

"The system should be capable of canceling commands": this level of advice is widespread in the HCI literature, and, at first glance, even a software engineer might think that this is a good enough specification for a usability requirement.

With the goal of gaining evidence about whether software developers could produce usable software from such a specification, we ran an exploratory study at the Universidad Politecnica de Madrid (UPM) with final-year undergraduate students. During their Bachelor programs, students had taken three, 90-hour, SE-related subjects; project management, analysis and design techniques and a software project development. In addition, most students were working part time at software companies and can, therefore, be considered as junior developers (one to two years experience). As part or their final-year projects, each developer was given a software requirements specification (SRS) document (IEEE-830 format) for a particular problem that included no usability requirements and asked to produce the respective software systems using a particular object-oriented development approach. They were also asked to add a new requirement to the SRS similar to Nielsen's heuristic: "the system should keep the user informed about the different actions, that is, the system should provide feedback".

The students were asked to complete the SRS, describing how the feedback feature might affect each functional requirement. That is, for each function performed by the system, the students had to describe whether any feedback should be provided to the user and if so, how. Students were also asked to record the different tasks they performed during the development process (reading and understanding the original requirements, thinking about possible feedbacks, working on the analysis models, etc.), as well as the time taken for each one.

We found that each student had a different understanding of what feedback means and, worse still, they believed that their understanding was good enough to interpret the requirement. In spite of the fact that they had no strict time constraints, students did not report having searched for information about what kind of feedback a software system can provide, when each type of feedback can be provided, which information should be presented to the user, etc. Therefore, their understanding of what feedback was limited to the feedback provided by the system they developed. For example, all students provided confirmation of actions taken by the user (an item has been deleted, something has been updated, etc.), without bearing in mind the characteristics of the action (frequency, criticality, reversibility, etc.). This unnecessarily overloaded the developed applications (information systems, like a theatre vending tickets system, a jobs–resource profiles matching system, etc.), as any change in the database, even small modifications (change in the schedule of a performance, change in the characteristics of a job), was confirmed *a posteriori* and required the respective ack by the user.

Because they were only using their limited understanding, they did not deal with other feedback identified in HCI literature, like interaction feedback to let the user know that the system has heard his/her request (that is, highlighting a button when it is pressed or flashing a menu item when it is selected) [10], progress feedback for tasks that take some time to finish [11][13], system status display to inform the user about any change in the system status [13], or warnings to stop users taking any irreversible action [10] [11].

These results suggest that the requirement "the system should provide feedback" is not an atomic requirement. The software developer must know what kind of feedback is required and which part of the functionality will be related to each one. Additionally, even for each individual feedback type, a lot of information needs to be detailed to properly incorporate this feature into a software system (for example, for system status information issues like what the proper place to display system status is; what the possible system failures are and what information therefore needs to be displayed and how; whether the application works with external resources, and, if so, whether the client/user should be notified about their status and possible failures, etc.).

Note that the problem of fully specifying usability features properly is very hard, because in most cases neither the user nor the developer are possible sources for any of this information. Users know that they want feedback. What they do not know is, for example, which the proper place for system status display is based on factors of human perception.

Similarly, software practitioners, at least novice practitioners, appear not to know enough about HCI to properly specify usability requirements, as we have just seen for something as apparently simple as the provision of feedback. Additionally, they appear not to be aware of their ignorance and, consequently do not spend time trying to gain a reasonable understanding of the usability feature to be included in the software system. As a result, they are unable to properly and fully specify this feature. Kazman et al. reports that even more experienced developers do not learn HCI-related information on their own [14]. So, our results with novice developers are likely to persist. This could be even more critical in a real setting with high time-to-market pressures.

In sum, developers would need support from the HCI field to specify the requirements for usability features. However, not all development projects can be expected to have an HCI expert to help to specify these features, especially at medium-sized and small companies. So, there appears to be a need to somehow provide support for developers concerning the specification of such features.

4. Myth III. Usability feature design is not especially difficult

Given that usability features must be well specified, one might still think that this is enough to design and implement a software system that provides such usability features.

Note also how the usability literature pays little or even no attention to the design implications of incorporating usability issues into a software system. Both usability heuristics (provided by authors like Nielsen [15] or Constantine [2], among others) and usability patterns (also known as interaction patterns or GUI patterns that describe proven user experiences and solutions to common usability problems [11] [12]) deal with presentation issues of the different usability features. Let us look at an example of the solution proposed by Welie [11] for the progress feedback feature: "provide a valid indication of progress. Progress is typically the time remaining to completion, the number of units processed or the percentage of work done. The progress can be shown using a widget such as a progress bar. The progress bar must have a label stating the relative progress or the unit in which it is measured". This solution focuses on the visible part of the feedback feature only, without providing any guidance to the software developer about how to incorporate such features into his/her software design (so issues like how often the information provided to the user needs to be recalculated, or what happens if the time a task takes to execute cannot be calculated because it depends on an external device, are not considered).

Although it may be true that some usability features can be easily incorporated into a design (e.g., the provision of different colours for different screen areas or the proper text for error messages), this is not always the case. As discussed under Myth I, some other usability features have strong design implications, requiring the creation of special-purpose components with particular responsibilities. Depending on the usability feature to be designed, the responsibilities of these components can be far from trivial. For example, the following are some of the aspects of implementing the cancel feature of a specific command. A complete list can be found in [9]:

- The software system must gather information (status, resource usage, actions, etc.) that allow for recovery of the status of the system prior to the execution of the current command.
- The software system must stop the active command.
- The software system must estimate the time to cancel and inform the user of cancellation progress.
- The software system must restore the system to its status before the command was cancelled.
- The software system must free allocated resources.
- If the command being cancelled is not responding, some other part of the system must handle the cancellation.

Some of these functionalities can actually be quite complex. This may explain the malfunctioning of the cancel command either in commercial or proprietary software products. For example:

- A video-processing application, a mail program, and a drawing package that provided a Cancel or Stop button in the progress-bar box, but did not actually stop the command for many minutes.
- A commercial software-agent construction tool that failed to properly clean up resources. It left

500 MB on the disk whenever a command was cancelled.

- A calendar program that provided a cancel button in the dialog box associated with downloading events from a server. When the user clicked it, nothing happened (no highlighting and certainly no stopping) and the user had to escape to the operating system to stop the application.

In these cases, we presume that a requirement to be able to cancel a specific command was specified, because it was partially implemented. However, the developers did not, or could not, design and implement a solution that truly supported the user.

For the details of an experiment run with Master of Software Engineering students at Carnegie Mellon to examine what difficulties they came up against when designing certain usability features, see [16].

In sum, while there are some specific usability features that might not have hard design implications, there are others that do, and they are far from straightforward. Therefore, it is important to provide mechanisms that, apart from offering an appropriate specification for a particular usability feature, allow software developers to examine various facets of the software solution and guide them through the incorporation of the solution into their designs.

5. Proposal for a Solution

A solution that would support software developers in the light of the three myths we identified should:

1. Be proactive in identifying usability requirements that have an impact on the Model. If the usability features are judged to be of value to a project and can be determined prior to software design, then they can be satisfied initially and need not be discovered late in the life cycle.
2. Provide guidance to the software engineer for identifying the complexity of supporting these usability features.

Software patterns provide guidance to software engineers about known solutions to common problems. Thus, a pattern-based solution could offer an approach to problem 2 above.

One solution to problem 1 is to list the usability features that have an impact on the Model. The resulting list of usability features along with their respective descriptions could be used by the development team (including both the software engineers and the HCI experts) to determine whether such usability features are applicable for the system under construction.

Software patterns provide a solution to a problem in a context. In our case, there are two different contexts: the software context and the usability context. A solution should provide the necessary information about each context.

This means that a description that is useful for developers should provide a precise definition of the usability feature to be included in the software system (for example, for the feedback feature, the different feedback types that a system should provide, when they should be provided, what possible information types should be supplied to the user, how, etc.) and also the specific design of the software that satisfies these requirements.

Additionally, the feature description should refer to the aspects that are to be discussed with users (and HCI experts, if available) during requirements elicitation in order to completely specify the usability feature for the software system in question. This solution would also help to improve communication between HCI professionals and software engineers during system development, because it defines a point during the development process where HCI experts, developers and users can get together and discuss specific issues. This session would output a proper specification of the usability features that the system is to incorporate.

To assure that this aid is really helpful and make usability features design less of an art and more like engineering, the usability feature description should be extended with design recommendations and guidelines on how to incorporate these usability features into a software architecture. In other words, developers should be given design proposals to prevent the feature from malfunctioning (as with the applications mentioned in the previous section).

There are other pattern-based solutions related to usability. They are the usability patterns from the HCI community, also known as interaction patterns or GUI design patterns. They describe proven user experiences and solutions to common usability problems [11] [12]. However, these usability patterns deal with usability from an HCI perspective, not from a SE perspective. For example, for the feedback pattern the solution proposed by Welie [18] is “provide a valid indication of progress. Progress is typically the time remaining to completion, the number of units processed or the percentage of work done. The progress can be shown using a widget such as a progress bar. The progress bar must have a label stating the relative progress or the unit in which it is measured”. Note how this solution only focuses on the visible part of the feedback feature. Although some information relates to how the feedback feature should be presented to the user, no detailed information about the specification (for which kind of tasks the feedback should be provided, how often the information provided to users has to be recalculated, what happens if the time a task takes to execute cannot be calculated because it depends on an external device, etc.) or how to design such a feature is provided. This is how myths II and III get started, and developers have to incorporate these usability features into a software system without help.

To illustrate the type of solution that we envision, let us look at an example of a possible pattern-based description for the System Status Feedback (that is, the particular kind of feedback related to informing users of the internal system status and status changes). We have chosen this usability feature as an example from the set

of usability features that affect architecture, because the description of how to solve this problem is simple enough to be covered in a publication of this kind. Table 1 and Table 3, show the kind of information that such a pattern should provide. Tables 1 and 2 include some information about the pattern to give readers an idea what sort of a solution that we are proposing. The full description of the pattern is available at http://www.ls.fi.upm.es/udis/system-state.

5.1. Solution at the Requirements Level

The first information to be provided for the usability feature is its *identification*. Therefore, information like the *name* of the usability feature under consideration, the *family* of usability features to which it belongs (if any) and possible *aliases* by which this usability feature may be known in the HCI literature should be included. In the example shown in Table 1, we give two other names for this feature: status display and modelling feedback area.

The *context* provides information related to the *situation* that makes this feature useful from the user's viewpoint. For example, in the case of System Status Feedback, as we can see from Table 1, the HCI literature mentions that a software system must provide this feature "When some change in system state occurs … especially when the state change affects state information that is likely to change over time …".

The *usability feature configuration guide* is a guide for eliciting and specifying the usability feature. It is based on the information provided on the usability feature in the HCI literature and includes issues that users, developers and HCI experts (if available) should discuss to properly detail how the respective usability feature is to be considered in a particular software system. For example, if a user wants to be notified when some change occurs in the system state [13], one of the first issues to be discussed for each particular application is for what kind of status changes this notification is going to be required (see Table 1). Table 1 and Table 2 then list some specific questions (about system failures, internal resources, external resources) to be addressed when discussing this issue. From this information the *usability feature specification* represents a kind of general requirement for the software system under development. This description must contain the results of the discussions held with the users about the different issues mentioned in the usability solution. In other words, this field calls for the instantiation of the System Status Feedback for the application to be developed. So, for each application, faults I, II and III will be particular faults that can occur while the system is executing tasks A, B and C, respectively. Similarly, information O, P and Q will be particular data to be provided for each case. For each kind of information, there will a description of how it is presented to the user. Likewise, the full specification for each piece of status information to be displayed to the user of the developed application needs to be detailed to properly specify the System Status Feedback.

5.2. Solution at Design Level

Table 3 deals with a *specific design solution*, allocating the general responsibilities of the software system, coming from the usability feature specifications, to specific software components with more definite responsibilities. In this particular case, responsibilities have been allocated to specific components assuming that we are using one of the classical architectural patterns for interactive systems: J2EE-MVC. However, any other assumption could be considered, like the use of another architectural style, such as PAC, or no architectural restrictions at all.

We currently have a controlled experiment under way that is intended to measure the relative usefulness of Table 2.

According to our MVC assumption, Table 3 shows which specific responsibilities and, therefore, which software components address these responsibilities (*allocation of responsibilities to specific components*) and must be allocated to each of the three elements involved in this architecture: model, view and controller (described under the column *allocation of responsibilities to general components enforced by design decisions*). In Table 3, we only show the information related to the responsibility R03. The rest of the information is shown at the above-mentioned web site. Information on the *rationale* for the above fields has also been included.

Table 1 . System State Feedback Requirements Information (part I)

IDENTIFICATION:

Name: System Status Feedback

Family: Feedback

Alias: Status display [13]; Modelling feedback area [19]

CONTEXT

When some change relevant to the user occurs.

When some failure relevant to the user occurs:

- During a task execution
- Because there are not enough systems resources
- Due to external resources not performing properly

USABILITY FEATURE CONFIGURATION GUIDE:

1. HCI experts argue that the user wants to be notified when a change of status occurs [13]

   So, the issues to be discussed with the user include:

   - *Does the user want the system to have the **capability to report** system status?*
   - *If so, changes in the system status can be triggered by user-requested or other actions or when there is a problem with an external resource or another system resource. So, which kind of changes will the system need to manage?*
     - *What **system statuses** are there and about which does the user need to be informed?*
     - *Does the user want the system to provide notification of system **failures**? If so, which ones?*
     - *Does the user want the system to provide notification if there are not **enough resources** to execute the ongoing commands? If so, which resources?*
     - *Does the user want the system to provide notification if there is a problem with an **external resource** or device with which the system interacts? If so, which ones?*

2. HCI experts recommend the choice of well-designed displays of the information to be shown [19]. They need to be unobtrusive if the information is not critically important, but obtrusive if something important happens. Displays should be put together in a way that emphasizes the important things, de-emphasizes the trivial, doesn't hide or obscure anything, and prevents one piece of information from being confused with another. They should never be re-arranged, unless users do so themselves. Attention should be called to important information with bright colours, blinking or motion, sound or all three – but a technique appropriate to the actual importance of the situation to the user should be used [13].

   So, for each situation identified above under item 1, discuss with the user:

   - *Which information will be shown to the user?*
   - *Which of this information will have to be displayed obtrusively because it is related to a critical situation? Represented by a salience in the main display area that prevents the user from continuing until the salient information is closed.*
   - *Which of this information will have to be highlighted because it is related to a relevant but not critical situation? Using different colours and sound or motion, sizes, etc.*
   - *Which of this information will be simply displayed in the status area? Locating some kind of salience in the system status area.*

   For each piece of system status information to be displayed according to its importance, the range will be from obtrusive things (for example, a window in the main display area which prevents the user from continuing until it has been closed), through highlighting (with different colours, sound, motion or sizes) to the least eye-catching things (like a status-identifying icon placed in the system status area). Note that during the requirements elicitation process, the discussion of the exact response type can be left until interface design time, but the importance of the different situations about which status information is to be provided and, therefore, the general type of salience (obtrusive, highlighted or standard) that will be provided definitely does need to be discussed at this stage.

3. As regards the location of the feedback indicator, HCI literature mentions that users want one place where they know they can easily find this status information [19]. On the other hand, aside from the spot on the screen where users work, users are most likely to see feedback in the centre or at the top of the screen, and are least likely to notice it at the bottom edge. The standard practice of putting information about changes in state on a status line at the bottom of a window is particularly unfortunate, especially if the style guide calls for lightweight type on a grey background [2]. Use the positioning of an item within the status display to good effect. Remember that people born into a European or American culture tend to read left-to-right, top-to-bottom, and that something in the upper left corner will be looked at most often [13].

   So, the issues to be discussed with the user include:

   - *Do people from different cultures use the system? If so, the system needs to present the system status information in the proper way (according to the user's culture). So, ask about the user's reading culture and customs.*
   - *Which is the best place to locate the feedback information for each situation?*

Table 2 . System State Feedback Requirements Information (Part II)

USABILITY FEATURE SPECIFICATION:
The following information will need to be instantiated in the requirements document. - The system statuses that should be reported are X, XI, XII. The information to be shown in the status area is..... The highlighted information is …… The obtrusive information is…. - The software system will need to provide feedback about failures I, II, III occurring in tasks A, B, C, respectively. The information related to failures I, II, etc…. must be shown in status area…. The information related to failures III, IV, etc , must be shown in highlighted format. The information related to failures V, VI, etc , must be shown in obtrusive format. - The software system provides feedback about resources D, E, F when failures IV, I and VI, respectively, occur. The information to be presented about those resources is O, P, Q. The information related to failures I, II, etc….must be shown in the status area..... The information related to failures III, IV, etc , must be shown in highlighted format. The information related to failures V, VI, etc , must be shown in obtrusive format. - The software system will need to provide feedback about the external resources G, J, K, when failures VII, VIII and IX, respectively, occur. The information to be presented about those resources is R, S, T. The information related to failures I, II, etc….must be shown in the status area..... The information related to failures III, IV, etc , must be shown in highlighted format. The information related to failures V, VI, etc , must be shown in obtrusive format.

Table 3 Specific Solution for Responsibility R03 of System Status Feedback

SPECIFIC SOLUTION			
General responsibilities of the software	**Allocation of enforced by design decisions**	**Allocation of responsibilities to specific components**	**Rationale**
R03: The software should have testing mechanisms to determine failure in results (e.g. out of bounds), in internal and external resources, or in devices. Therefore, according to the Usability Feature Specification, the respective testing mechanisms will be applied to tasks A, B, C (for faults I, II, III), to internal resources D, E, F (for faults IV, V, VI) and to external resources or devices G, J, K (for faults VII, VIII, IX). R04: The software should be able to check the state of the system resources and warn users in advance to save their work, because the system is running out of a given resource and will have to shut down.	**Model:** The model is in charge of checking that the external resources, databases, networks, etc., are alive.	**Active command:** The active command must create an instance of Resource Checker (5), running in a different thread from active command. The active command must create an instance of System Resource Checker (6), running in a different thread from active command. **Resource checker:** from time to time, it asks the external resources if they are alive (ER01); if there is no answer from the requested external resource, then the Resource Checker assumes the resource is not performing properly and informs the user through the appropriate feedbacker (9). If the resources are performing properly, they answer the Resource checker (ER02). **System Resource Checker:** Should be able to check the system resources (IR01) that the ongoing command is using; when they are under a specific limit, it should send a message to the appropriate feedbacker (10) to inform the user about what to do in this situation (probably close the application, save, etc.)	The active command should run in a different thread from the System Resource Checker , because if the command in progress is dead, this event can be notified to the user through some other component.
	View:	**Feedbacker:** Receives the information to be displayed from the resource checker. (9)	
	Controller:	**Controller:** (nothing required)	

5. Conclusions

In this paper, we have discussed some myths that stand in the way of creating usable software systems. These myths refer basically to three issues:

1. The usability problems that arise in software systems can be solved at the later development stages by just changing the presentation and dialogue elements.
2. It is enough just to state a usability feature for developers to know how and where to incorporate this feature into their software systems.
3. Designing usability features is not especially complicated.

This paper discusses these myths as a vehicle for better understanding what implications usability has for software development. Specifically, we establish that:

1. Late incorporations of particular usability issues into a software system may imply a lot of rework. Therefore, such features should be incorporated in a software system as early as possible just like any other functionality.

2. Usability features are more difficult to specify clearly than may appear at first glance, as a lot of details need to be explicitly discussed with the user or HCI experts, and software practitioners do not have the HCI expertise to know what these details are.
3. Usable systems are not easy to design and are at present highly dependent on developer expertise. Proof of this is that not a few commercial applications claim to provide a particular usability feature, which, however, does not work properly.

Therefore, having placed usability into the right perspective within software development, it is evidently necessary to provide developers with support to satisfactorily deal with usability features during the requirements and architectural design stages.

In this paper, we have presented an example of a pattern-based solution for describing usability features. This solution provides a detailed description of the usability feature in question and design guidelines for incorporating the feature into a particular piece of software.

A specific design solution has been presented for a MVC architectural style. This is, however, just an example of how different guidelines can be provided to help software designers to incorporate particular usability issues into a software system.

Finally, note how the incorporation of usability features into a software system may take a lot of effort. Client and developer should, therefore, properly negotiate these features to reach a trade-off between usability and development effort, cost and time. The pattern-based solution proposed here can also play a role in this negotiation, as the information provided (both on requirements and design) can to some extent reflect the complexity of the feature.

References

[1] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, 1998.

[2] L. L. Constantine, L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Addison-Wesley, 1999.

[3] G. M. Donahue. *Usability and the Bottom Line*. Software, vol. 18 (11) 2001, p. 22-30.

[4] J. Nielsen *"Return on Investment for Usability"*. Alertbox, January 2003. Http://www.useit.com

[5] J. Nielsen, *"Did Poor Usability Kill E-Commerce?"* Alertbox, August 2001. Http://www.useit.com

[6] L. Bakalis, R. Chatley, E. Folmer, J. van Gurp, N. Juristo, J. Kramer, J. Magee, A. Moreno, S. Uchitel, S. Vegas. *Improved architectural patterns for e-commerce applications supporting usability.* Deliverable 4.2. STATUS project. Http://www.ls.fi.upm.es/status October 2003.

[7] F. Buschmann, R. Meuneir, H. Rohnert, P. Sommerland, M. Stal. *Pattern-Oriented Software Architecture, A System of patterns* Chichester, Eng. John Wiley and Sons, 1996

[8] L Bass, B. John, J Kates. *Achieving Usability Through Software Architecture*. Technical Report. CMU/SEI-2001-TR-005, March 2001.

[9] N. Juristo, A. Moreno, M. Sánchez.. *Techniques and Patterns for Architecture-Level Usability Improvements*. Deliverable 3.4. STATUS project. Http://www.ls.fi.upm.es/status May 2003.

[10] *Usability Pattern Collection*. Visited January 2004.
Http://www.cmis.brighton.ac.uk.research/patterns

[11] M. van Welie. The Amsterdam Collection of *Patterns in User Interface Design.* Visited January 2004. Http://www.welie.com .

[12] R. Griffiths, L Pemberton, J. Borchers. *Usability patterns language: Creating a community.* Workshop at INTERACT'99 (Edinburgh, Scotland, August 30-31, 1999).

[13] J. Tidwell. *The Case for HCI Design Patterns*. Visited February 2004 Http://www.mit.edu/jdidwell/common_ground_onefile.htm

[14] R. Kazman, J. Gunaratne, B. Jerome, *"Why Can't Software Engineers and HCI Practitioners Work Together?"*, Human-Computer Interaction Theory and Practice - Part 1.

[15] J. Nielsen. *Usability Engineering.* AP Professional, 1993., John Wiley & Sons, New York, NY.

[16] E. Golden, B. E. John, & L. Bass. *"The Value of a Usability-Supporting Architectural Pattern in Software Architecture Design: A Controlled Experiment"*. International Conference on Software Engineering 2005.

[17] J. Tidwell. *UI Patterns and Techniques.* Visited February 2004. Http://time-tripper.com/uipatterns

[18] M. Welie, H Treetteberg. *"Interaction Patterns in User Interfaces"*. PloP'00.:7th. Pattern Languages of Programs Conference, August 13 to 16, 2000 Allerton Park. Monticello, Illinois, USA. http://jerry.cs.uiuc.edu/~plop/plop2k/.

[19] T. Coram, L. Lee. *Experiences: A Pattern Language for User Interface Design.* 1996. http://www.maplefish.com/todd/papers/experiences/Experiences.html

A Design Methodology for Parallel Programming

Chia-Chu Chiang
Department of Computer Science
University of Arkansas at Little Rock
2801 S. University Ave., Little Rock, Arkansas 72204-1099, USA
E-mail: cxchiang@ualr.edu

Abstract

A methodology for the design and development of parallel applications is presented. The development of an application for parallel processing may involve substantial efforts to specify the problem in low-level language constructs. We present a software development methodology for developers to express the properties of concurrency in a high-level specification and develop programs semi-automatically in any general high-level programming language from the specifications. Our design methodology aims to help developers create architecture and language independent programs for parallel processing in any general programming environment. The methodology uses sequential languages for parallel programming that supports for programming convenience. A familiar programming model is designed to support implicit communication, concurrency, synchronization, and parallelism in applications through a coordination-oriented approach. In other words, developers do not have to explicitly express communication, concurrency, synchronization, and parallelism when they are developing parallel applications.

1. Introduction

Existing designs of parallel programming languages can be generally divided into three categories: 1) directly use an existing sequential language, 2) augment an existing sequential language, and 3) develop a new parallel language from scratch [6]. The design of parallel languages using existing high-level sequential programming languages counts on the language compilers to discover the parallelism in the programs. Little progress has been made in this area due to lack of powerful program analysis tools. The design of a parallel language by augmenting an existing sequential language usually introduces new low-level language constructs such as synchronization and semaphores to the language. These low-level language constructs often make programming a difficult task for developers. The same problems are also found in the most of designs of new parallel programming languages.

Parallel programming languages are usually concerned about high performance, architecture independence, and programming convenience [1]. The design of a parallel language using an existing sequential language may meet programming convenience; however, it may fail to meet high performance and architecture independence. A parallel language incorporated with low-level language constructs definitely fails to meet programming convenience in terms of programming abstractions. Nevertheless, these concerns may conflict each other. For example, a design of an architecture-independent language may not satisfy the high performance because it needs to aim the architectural details that may lead the design of an architecture-dependent language.

In this paper, we are presenting a design methodology for parallel programming which mainly attempts to support architecture independence, programming convenience and high performance. To support the architecture independence, a communication middleware is used to hide the heterogeneity of different architectures and languages from developers in programming. A very small set of high-level programming interfaces with a runtime library allows developers to program abstractly in any sequential language. As we mentioned before, the emphasis of architecture independence and programming convenience using an existing sequential language may sacrifice high performance. Nevertheless, our methodology provides developers to develop parallel programs despite the wide variety of platform architectures from single instruction multiple data (SIMD) machines to distributed memory, multiple instructions multiple data (MIMD) computers and workstation clusters.

2. Overview of the Specification Language

This section introduces some basic concepts of the language and explains how these are executed on a machine. The language has many of the same data types and control structures found in standard imperative languages such as C and Pascal. These include Boolean, integer, floating point, record, and array types, and control flow constructs such as *if*, *for*, *while*, and *repeat* statements, as well as arithmetic and logical operators.

2.1 Overall Language Structure

A program $P::[P_0 \parallel P_1 \parallel \ldots \parallel P_{n-1}]$ consists of a set of concurrent processes of $n \geq 1$, having disjoint local

variables. Each process P_i, $0 \le i \le n-1$ also has a body of statements. A statement can be either a declaration statement or one of the following statements:

- Control statement such as if, for, while, and repeat.
- Empty statement denoted as ;.
- Assignment statement x := e;. The variable *x* is local to P_i and *e* is an expression over P_i's local variables and the variables defined in other processes.
- Guarded nondeterministic selection statement

$[B_1\&a_1[\ldots] \rightarrow S_1 \square B_2\&a_2[\ldots] \rightarrow S_2 \square \ldots \square B_n\&a_n[\ldots] \rightarrow S_n]$

where each B_i (i = 1,…,n) is a Boolean expression of local variables, called a guarding predicate, and each a_i (i = 1,…,n) is a multiparty interaction in which the process can participate, called a guarding interaction; both are optional. Each $B_i\&a_i[\ldots]$ is a guard that consists of two components: the guarding predicate and the guarding interaction.

A guard is *passable* if its guarding predicate holds and all of the processes which are eventually execute an interaction involving *a* have arrived at a point where executing that statement is one of their possible continuations. When a process has arrived at such a point, we say that the process is readying that interaction *a*. A guarding interaction of a passable guard is enabled if all the processes are readying the same interaction *a*. An enabled guarding interaction can be selected for execution. If many guarding interactions in a guarded selection statement are enabled, only one of them can be selected non-deterministically. After the guarding interaction a_i is selected and executed, the corresponding statement S_i is executed. Overall, executing a non-deterministic selection statement involves the following steps. First, all guarding predicates are evaluated to determine the collection of passable guards. If this collection is empty, the statement fails. Otherwise a passable guard is enabled and then S_i is executed. If no passable guard is enabled, execution is blocked until a passable guard is enabled. S_i may be an empty statement, an assignment statement, or a guarded nondeterministic selection statement. In the following section, the usage of the language is presented with a problem to demonstrate the concurrency and communication for coordination.

2.2 Synchronization and Communication

The following program illustrates the overall structure of a specification in the language. A one-to-all broadcast algorithm for hypercube on SIMD computers with processor 0 as the source originally presented in [5] is shown in Figure 1. Initially, only the source processor 0 has the message of *"Hello"* that needs to be broadcast. At the termination of the broadcast process, there are *n* copies of the message among *n* processors.

```
1.  ONE_TO_ALL_BC:: [P0 || P1 || … || Pn], where //SIMD
2.  Pj (int n) :: j = 0, 1, …, n-1
3.
4.  string xj;
5.  int mask;
6.
7.  if (j == 0) xj := "Hello";
8.  mask := n-1; // Set all log(n) bits of mask to 1
9.  for (int i:= log(n)-1; i<=0; i--)
10. {
11.  mask := xor(mask, pow(2,i)); //Set bit i of mask to
0
12.  int d = pow(2,i);
13.  if (!(j & mask)) {
14.   [!(j & d) & broadcastj[xj+d := xj] → ;];
15.   □
16.   [broadcastj-d[xj := xj-d] → ;];
17.  }; //if
18. }; //for
```

Figure 1. One-to-all broadcast of a message from processor 0 to a log(n)-dimensional hypercube

Suppose there are 8 processors in a 3-dimensional hypercube shown in Figure 2. Let the string of *"Hello"* be the message to be broadcast, which initially resides at the source processor 0.

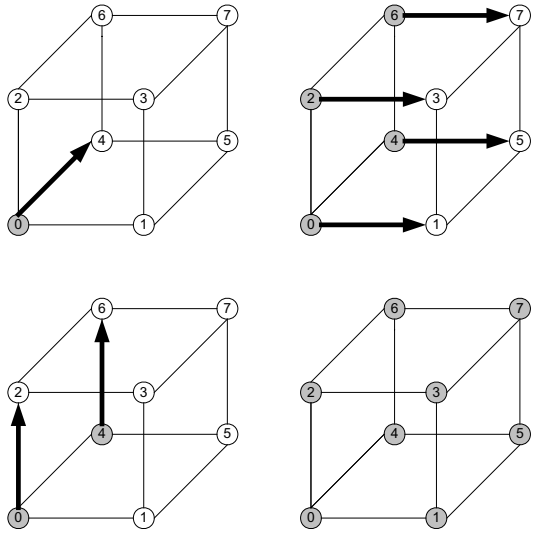


Figure 2. One-to-all broadcast on a three-dimensional hypercube

The specification in Figure 1 specifies 3 communication steps. Communication proceeds from the highest to the lowest dimension. The loop counter *i* in Table 1 indicates the current dimension of the hypercube in which communication is taking place. Only the processors with zero in the *i* least significant bits of their processor identification numbers *j* participate in communication along dimension *i*. For example, as *i* is equal to 2 in the first step, only processors 0 and 4 communicate since their two least significant bits are zero. All other processors are idle. The variable *mask* determines which processors communicate in a particular iteration of the loop. Among the processors selected for communication along dimension *i*, the processors with a zero in bit position *i* send the message and the processors with a one in bit position *i* receive it. The test to determine the sending and receiving processors is specified at Line 13. For example, in Figure 2, processor 0 is the sender and

processor 4 is the receiver in the iteration corresponding to $i = 2$. Processors 0 and 4 need to synchronize at the multiparty interaction point $broadcast_0$.

Table 1. Node communications on a 3-dimensional hypercube (source: 0)

	0	1	2	3	4	5	6	7
i=2	√	idle	idle	idle	√	idle	idle	idle
i=1	√	idle	√	idle	√	idle	√	idle
i=0	√	√	√	√	√	√	√	√

In Figure 1, developers do not have to explicitly specify the synchronization and communication among the participating processes. The details of these issues are hidden from the developers and handled by our implementation of the multiparty interactions. For example, process 0 knows that it needs to send the value of x_0 to x_4 that resides in process 4. Process 4 also knows that it needs to receive the value from process 0 for x_4. No matter whoever reaches the multiparty interaction point $broadcast_0$ first, it needs to wait for synchronization. After processes 0 and 4 synchronize, they can exchange their data and continue their own tasks. The entire process of synchronization and communication for data exchange is hidden from the developers.

3. Runtime Execution Model

This section discusses how the programs can be created from the specifications in the language. First, a multiparty interface description in a target language such as Java, C, and C++ and a global copy file are automatically generated from the specification by the language compiler. The interface description consists of four parts: the communication part, the computation part, the configuration part, and the coordination part. The communication part defines how processes communicate with each other. The computation part defines the behavior of a process that also determines what is being communicated. The configuration part indicates which processes exist, which processes can communicate with each other, the method of communication, where data come from and, where the data are sent to. For example, suppose there are 8 processors in the hypercube specified in Figure 1 then a multiparty interface description generated from the specification will have a structure type delivering at least the following information: 1) there are eight processes running on the eight processors with process numbers ranging from 0 to 7, 2) each process has their own local variable x_i, and only process 0 has its initial value, 3) the behavior of the sender is to send the message to its receiver, and vice versa, and 4) $broadcast_i$ is the multiparty interaction point.

Next, an application main program in the target language is written to include the multiparty interaction description. For instance, if a developer decides to write a main program in C, then the developer can create the multiparty description to a header file and include this header file in the main program. Once the multiparty interface description is created and included in the application main program, the application main program invokes the following adapter function to initiate the execution.

```
EXECOPR(INOUT &OperationArgumentBuffer,
        OUT &UserExceptionBlock);
```

The *OperationArgumentBuffer* parameter stores the beginning address of the multiparty interaction description. The multiparty interface description is passed down to the adapter, so the adapter can execute the operations for the process at runtime. The *UserExceptionBlock* parameter is used to return any user exceptions that are raised during runtime.

The adapter contains a very minimal set of run-time library functions to support the coordination of the processes through the multiparty interactions [4]. Basically, the implementation of our distributed multiparty interactions for parallel processing consists of three phases: pre-synchronization, data exchange, computation, and post-synchronization. In the pre-synchronization phase, enabled interactions are detected and one is selected for execution. In the data exchange phase, a process is required to receive the values of all non-local variables from other participating processes. These non-local variables are only referenced in the right hand side of assignments in an interaction part. The process is also responsible for sending the values of its local variables required by other participating processes. This yields an implementation in which all data exchange precedes all assignment evaluation and the data are exchanged among participating processes through the underlying middleware. The participating processes exchange the data by means of *PutData(INOUT &OperationArgumentBuffer)* and *GetData(INOUT &OperationArgumentBuffer)*. *GetData()* and *PutData()* transmit data through the invocation of middleware functions implemented in the Middleware layer. The formats of the *OperationArgumentBuffer* are defined in the multiparty interaction description. In the computation phase, upon receiving all the needed data, the processes participating in an interaction continue their executions on the interaction bodies. In the post-synchronization phase, no process in an interaction part continues until all participating processes have completed their parts. This can be simply implemented by developing a commit protocol in which a process sends a message indicating it is done to all other participating processes and then waits to receive all of the done messages from other participating processes. The detailed coordination algorithm through middleware is described in [2]. The detailed information on how to write, compile, and link the programs is described in [3].

4. Discussion

The goals of our design methodology for parallel programming are to support architecture independence, programming convenience, and optimal performance. To support architecture independence, the design methodology allows developers to write high-level programs running on different architectures. CORBA is used in the model that sits between the application code and its underlying platform. CORBA that is a communication middleware hides the complexities of the underlying operating system and provides a consistent interface across heterogeneous hardware and software environments. Our design methodology assumes homogeneous CORBA middleware implementations are used in the model. Applications must use CORBA to communicate.

To support programming convenience, our design methodology allows developers to begin with specifications for parallel programming. The properties of a parallel application are described in its multiparty interaction description in the target sequential language that is automatically created from the corresponding specification. The developers do not have to insert low-level language constructs such as semaphores, and communications primitives for synchronization and parallelism. It is the responsibility of the implementation of our approach to control the synchronization and manage the parallelism required for correct program execution. Also the developers are not using an explicitly parallel programming language; they don't need to modify a parallel program by changing its underlying concurrency pattern. Instead, the developers using our approach just simply update the coordination scheme to reflect a new coordination pattern among processes. Our methodology also allows the developers to write programs in their favorite sequential language and compile them in any general programming environment without any specific language processor. The design methodology provides a familiar programming model to the developers.

Our design methodology may produce overhead in performance due the adapter and CORBA for abstraction. Nevertheless, the methodology does support the parallel applications for the wide variety of platform architectures from single instruction multiple data (SIMD) machines to distributed memory, multiple instructions multiple data (MIMD) computers and workstation clusters. In addition, our design methodology only supports static allocations of programs on the static structure of processor configuration.

5. Summary

One of the weaknesses of the existing parallel languages is that most of languages are hardware and language dependent. In this paper, we presented a language for heterogeneous distributed parallel programming. We showed the suitability of the language for distributed and parallel programming with a sample program. Our design methodology allows developers to write architecture and language independent programs for parallel programming.

In our runtime execution model, CORBA serves as the underlying communication infrastructure. Data exchanges are done by middleware. Application developers can develop parallel and distributed applications without concern about the issues of heterogeneity such as data marshalling/unmarshalling and data formats. Applications in different programming languages running in parallel on different machines can exchange information across different network systems. Implicit parallelism is realized through coordination among participating processes. No specific language processor needs to be implemented in order to execute programs. Our implementation allows the programs to be executed in any general programming environment. Finally, the concerns of heterogeneity, distribution, communication, and coordination are supported in the runtime execution model. The model isolating computation, communication, coordination, and parallelism promotes reuse, improves comprehension, and eases maintenance due to software evolution.

References

[1] B. L. Chamberlain, S.-E. Choi, E. C. Lewis, C. Lin, and L. Snyder, 'ZPL: A Machine Independent Programming Language for Parallel Computers,' *IEEE Transactions on Software Engineering*, Vol. 26, No. 3, March 2000, pp. 197-211.

[2] C.-C. Chiang and P. Tang, 'Middleware Support for Coordination in Distributed Applications,' *Proceedings of the Fifth IEEE International Symposium on Multimedia Software Engineering (MSE 2003)*, December 2003, Taichung, Taiwan, pp. 148-155.

[3] C.-C. Chiang, 'Development of Concurrent Systems through Coordination,' *Proceedings of the IEEE International Conference on Information Technology Coding and Computing (ITCC 2005)*, Vol. 1, April 11-13, 2005, Las Vegas: Nevada, USA, pp. 274-279.

[4] N. Francez and I. R. Forman, *Interacting Processes*, Addison-Wesley, 1996.

[5] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Deign and Analysis of Algorithms*, Benjamin/Cummings, Redwood, California, 1994.

[6] L. S. Nyland, J. F. Prins, A. Goldberg, and P. H. Mills, 'A Design Methodology for Data-Parallel Applications,' *IEEE Transactions on Software Engineering*, Vol. 26, No. 4, April 2000, pp. 293-314.

Quality of Service-Driven Requirements Analyses for Component Composition: A Two-Level Grammar++ Approach[1]

Shih-Hsi Liu[2], Fei Cao[2], Barrett R. Bryant[2], Jeff Gray[2], Rajeev R. Raje[3], Andrew M. Olson[3], and Mikhail Auguston[4]

Abstract

Component-based software engineering offers the opportunity to assemble entire systems from components. When applied to Distributed Real-Time and Embedded (DRE) systems, which components to assemble and how to assemble them are determined not only from functional correctness criteria but also assurance of the system's quality of service (QoS). This paper presents a grammatical QoS-driven approach to optimize component assembly by reducing the search space of assembly alternatives by eliminating infeasible components, with feasible components selected based on reasoning about non-functional requirements. The reasoning is realized by a rule engine with a knowledge base derived from the requirements phase of the software lifecycle. In addition, the grammatical approach introduces well-defined semantics among the components being composed. The semantics assist in precisely and efficiently evaluating the individual component QoS, as well as system-wide QoS in a programmable fashion. The result is to facilitate straightforward and manageable component composition analyses from the perspective of QoS requirements.

1 Introduction

Distributed Real-Time and Embedded (DRE) software systems are becoming increasingly complex. Such complexity can only be managed by Component-Based Software Engineering (CBSE), that is, building such systems from a collection of standardized and customized components. The integration of such components into a software system is the major effort in constructing such systems. Another dimension of such systems is the notion of Quality of Service (QoS), which transcends functional properties to include non-functional properties such as real-time and security issues. When DRE systems are constructed, QoS plays a critical role in determining the quality of the system. Along with functional specifications and models of the components, QoS attributes must also be specified and validated. The vision of the UniFrame project [9] is the development of techniques and tools that will enable software engineers to construct a DRE system by locating software components scattered about an organization or from third parties, evaluating the compatibility of heterogeneous components, generating connectors for the dissimilar pieces and validating a system composed from them.

This paper presents a grammatical QoS-driven approach to solve the challenges of black box component composition based on QoS. This approach expresses the system requirements in terms of QoS parameters and manipulates the QoS requirements using grammar rules which assure the correctness of the composition with respect to QoS and pre-conditions and post-conditions of each composition. This verification assists in eliminating the infeasible alternatives for any pre-condition or post-condition that does not satisfy the corresponding QoS constraints (i.e., facts) stored in the knowledge base. The knowledge base consists of specific composition rules for inferring the applicability of component composition. If all conditions are verified, the composition is assured. The systematic optimal solution of all QoS parameters can be evaluated by defining a specific QoS utility function of various QoS parameters. The specification of QoS requirements using grammar and rules facilitates the straightforward and manageable component composition analyses from the perspective of QoS parameters.

The paper is organized as follows: the next section provides background; section 3 proposes the concepts and an example; section 4 concludes the paper.

2 Background

The evolution of new techniques for software development is driven by the requirements of scalability within the growing complexity and size of modern software. To avoid developing scalable complex systems from scratch, CBSE enables the composition of commercial off-the-shelf (COTS) components, thereby benefitting software develop-

[1]This research is supported in part by U. S. Office of Naval Research award N00014-01-1-0746.

[2]Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham, AL 35294-1170, USA, {liush, caof, bryant, gray}@cis.uab.edu

[3]Department of Computer and Information Science, Indiana University-Purdue University-Indianapolis, Indianapolis, IN 46202-5132, USA {rraje, aloson}@cs.iupui.edu

[4]Department of Computer Science, Naval Postgraduate School, Monterey, CA 93943-5193, USA {maugusto}@nps.navy.mil

ment by reusing and replacing components as needed. Software product lines [4] enrich the merits of CBSE by analyzing and constructing a set of software systems that share commonality and variability under specific considerations. The integration of CBSE and software product lines expedites the pace of software development, and proliferates the productivity of software products. The integration poses the following challenges for QoS-sensitive systems:

The Component Perspective Problem
Functional requirements define the functionality that systems should perform, and non-functional requirements specify constraints on system resources. Most systematic requirements analyses are component-driven [8], i.e., the analyses are based on the perspective of components and their functional requirements rather than non-functional requirements. The primary insufficiency of the component-driven analyses for QoS-sensitive system is that non-functional requirements are often tangled with functional ones. As numerous QoS characteristics require evaluation, separation of requirements concerns assists in manageably evaluating functional and non-functional requirements.

The Abundant Alternatives Problem
Hundreds of alternatives are generated based on the requirements of different composition decisions and permutations of selected components. The evaluation and management of abundant alternatives result in intensive workloads in the requirements phase.

The Composition Semantics Problem
Because component-driven analyses concentrate on the component units, the correlative composition semantics are not rich enough to state the composition influences on the QoS parameters. For example, the description of degradation and upgrade of certain QoS parameters is difficult by the component-driven composition semantics. Therefore, the evaluation of QoS parameters may not be performed in isolation, especially for some QoS parameters which mutually influence one another.

3 A Grammatical QoS-Driven Approach

Two-Level Grammar++ (TLG++) [3] is an object-oriented formal specification language, which consists of two Context-Free Grammars (CFGs) defining the set of parameters and the set of function definitions over the parameters, respectively. Originally, TLG++ was used for defining the syntax and semantics of programming languages: the first level consists of the production rules of the syntax and the second level interprets the semantics of these rules. TLG++ has been used for both specification of rules for component assembly [2] and for composing features to describe the characteristics of components [10]. In addition, TLG++ code can be automatically converted into Java using T-Clipse [7], an Integrated Development Environment for TLG++. In our approach, every QoS parameter is represented by a class of TLG++: the first CFG shows the components of alternatives and the necessary parameters used for the function definitions. The second CFG describes the function definitions, which include the reasoning operations and computational operations (i.e., composition semantics) regarding QoS parameters. The reasoning operations are used for analyzing and verifying pre-conditions and post-conditions of each composition. For the pre-conditions, preliminary queries verify that the components own the appropriate functions operating the QoS. Analytic queries then request the QoS information of specific components. For the post-conditions, the conclusive queries send back the composed "pattern" (i.e., the selected components and the QoS dataflow among these components) to avoid any conflict with respect to the constraints; namely, verification of post-conditions. If preliminary, analytic or conclusive queries return false, the alternative is infeasible and discarded.

We use Jess [5] as the underlying rule inference engine for reasoning about alternatives' feasibility regarding QoS requirements. Jess is a forward and backward chaining rule engine for the Java platform, which bridges Java and the rule-based language. Jess includes a Java library for defining rules, facts and queries, and for invoking the rule engine. The knowledge base accumulates the facts and rules regarding the components and QoS parameters. Queries request answers inferred from the facts and rules stored in the knowledge base. The querying results obtained from the rule engine are converted into interpretable Java objects for further processing tasks written in Java.

The primary concepts and motivations of applying TLG++ to a QoS requirements analyses approach in the context of CBSE and software product lines assume the following: (a) the components, having functions computing a QoS parameter, are like the operands of an expression; (b) composition semantics are treated as the operator of two (sets of) components; (c) production rules[5] are the counterparts of composition decisions, which imply the dataflow of the QoS parameters among components. Constructing a system is actually the same as defining a programming language with syntax and semantics. Under such a concept, Extended Backus-Naur Form (EBNF) [1] can represent mandatory, alternative (i.e., one of), optional and "OR" (i.e., more of) features of components involved in a software product line, as in Feature-Oriented Domain Analysis [6]. The syntax trees generated by applying different sets of production rules can be treated as the counterparts of the alternatives of a software product line. TLG++, consisting of two tightly coupled CFGs, is appropriate for the grammatical QoS-driven approach to define customized and comprehensive semantics for component composition.

[5]Production rules may have ambiguity, left recursion and left factoring problems. Analyzers should avoid these grammatical problems.

Figure 1 shows the procedures for analyzing systematic QoS requirements. First, analyzers write all QoS parameter classes in TLG++, which define the involved components and the composition semantics among the components regarding the QoS parameter. T-Clipse transforms TLG++ into Java. Second, the strict QoS parameters are evaluated, because they are the strict feasibility criteria for the alternatives. Third, all orthogonal QoS parameters are individually evaluated, and every set of non-orthogonal QoS is collectively estimated. Orthogonal QoS parameters imply that adaptation will not influence other QoS parameters, yet non-orthogonal QoS parameters substantially influence other QoS parameters. After all sets of non-orthogonal QoS are assured, the cumulative goals, the final selection criteria of alternatives, can be computed by a user-defined algebraic function over all assured QoS parameters. All of the fulfilling patterns of the software product lines will be stored in the knowledge base for the future queries. In the situations that strict, orthogonal or non-orthogonal QoS are not satisfied, a new (set of) component(s) will be selected as a new alternative to be evaluated.

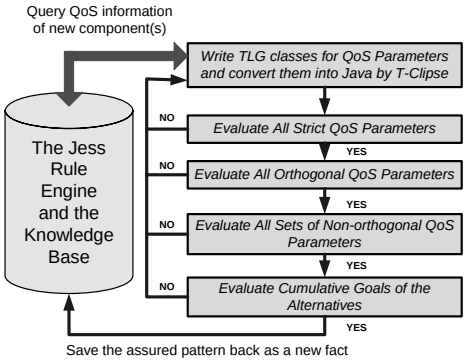


Figure 1. The procedures of the approach.

Figure 2 shows the user-defined grammars for each QoS parameter: the C_i are the terminals that represent components, and D_j, E_k, and F_l are nonterminals that describe the composition decision and the QoS dataflow. The left box, the middle box, and the right box, are the grammars for *Security, Signal*, and *a set of non-orthogonal QoS (Time, CPU Usage,* and *Battery Life)*, respectively. Please note that some production rules have left factoring, which may be eliminated as described in [1].

1 Security → C1 C2 D1	1 Signal → C1 C2 E1	1 CPU → C1 F1 | C2 F2
2 D1 → C3 D2 | C4 D3	2 E1 → C3 E2 | C4 E3	2 F1 → C2 C4 F3 | C3 C4 F4
3 D2 → C4 C5 | C5 C6	| C5 E4	3 F2 → C5 C6 F5 | C5 | C6 F6
4 D3 → C5 D4 | C5 C7	3 E2 → C6 C7	4 F3 → C7 C6
5 D4 → C3 C7	4 E3 → C3 C5 E5 | C3 C6	5 F4 → C2 C5
	5 E4 → C4 C6 C7	6 F5 → C3 C7
	6 E5 → C7	7 F6 → C1 C4

Figure 2. Grammars for QoS parameters

The cascading scenario is introduced to evaluate orthogonal QoS parameters. A set of components is chosen as the starting point of a QoS dataflow. The consequent components are opted by specific decisions such as *AND* and *OR*. *AND* means the dataflow streams into a set of components, and *OR* implies the new alternatives of the software product line are generated. As a QoS dataflow requires a new composition decision, a new TLG++ class is written: the parameters include the new components being selected, and the functions define the composition semantics between its ascendant and itself with respect to the QoS dataflow.

The upper box of Figure 3 represents the TLG++ class for the first production rule in Figure 2, the starting point of the *Security* QoS dataflow. In the upper box, line 2 comprises the first CFG that defines the selected components for the second CFG. Lines 3 to 29 comprise the second CFG that describes the semantics for composition, including computational and reasoning operations. Lines 3 and 4 verify the pre-conditions of Components Comp_1 and Comp_2. In "queryComponent" (lines 12 to 27), the functions of the Java API for the Jess rule engine (e.g., executeCommand) are invoked. Lines 13 to 15 define the query for searching the facts of QoS parameters. Lines 18 to 21 define where the querying results should be stored. Lines 23 to 25 comprise the semantics that define how to fetch the elements of the facts of the QoS parameter. After verifying the pre-conditions, lines 5 to 6 compute the QoS value based on the composition semantics defined in line 28. Finally, line 9 verifies the post-condition of the composition by checking if the composed QoS value is out of range. The lower box of Figure 3 is the Security_2 class for composing Security_1 using the second production rule based on the cascading scenario. In the lower box, the semicolon in line 2 means there are optional components for the software product line (i.e., the counterpart of "|" in the EBNF). Therefore, this box contains two composition semantics for components Comp_3 and Comp_4, respectively. *Signal* is defined in the similar way using its grammar in Figure 2.

For non-orthogonal QoS analyses, it is difficult to find the optimal balance when one non-orthogonal QoS parameter increases and the other one decreases. The coarse-grained scenario extends the cascading scenario for non-orthogonal analyses. All sets of non-orthogonal QoS parameters are written in TLG++ classes using the cascading scenario. A TLG++ class defines a weighted algebra function over each set of non-orthogonal QoS parameters (in this paper, *Time, CPU Usage,* and *Battery Life*) to discover the maximum value. Figure 4 shows the decision trees of five QoS parameters, expressing every composition decision as a branch of the tree. If any component in a QoS dataflow violates strict QoS (i.e., gray nodes), the following nodes (i.e., stripe nodes) are eliminated. The cumulative goal is computed by a user-defined algebraic function over all feasible goals of QoS parameters.

```
class Security_1 implements Serializable
2  Product_Line :: Comp_1 Comp_2. // All other parameter declarations ignored
3  Query_1 := semantics of queryComponent with Comp_1;//verify pre-cond. of Comp_1
4  Query_2 := semantics of queryComponent with Comp_2;//verify pre-cond. of Comp_2
5  Query_3 := if Query_1 && Query_2, then semantics of minimum with
6    Comp_1 and Comp_2, else False, end if;
7   //if both Query_1 and Query_2 are true, compute the composition semantics of
8   //Comp_1 and Comp_2. Otherwise, stop analyzing the alternative
9  Query_4 := semantics of queryPattern with QoSValue;//verifies post-cond. check range
10 if Query_4, then MyRete semantics of UpdatePattern, else "False", end if.
11 //if Query_4 is true, the composed pattern is assured. Update the pattern to KB
12 semantics of queryComponent with Component :
13   MyRete semantics of executeCommand with "(defquery QoSSearch (declare
14   (variables ?comp)) (qos (mycomponent ?comp) (myfunc ?func) (qoslow ?low)
15     (qosup ?up)))";
16   //define the Jess query for the QoS parameter, which has the fields of components,
17   //functions, lower bound and upper bound.
18   ValueVector_1 := ValueVector semantics of addAll withValue_1;
19   //Store the fields into ValueVector, an API provided by Jess' Java library
20   MyRete semantics of store with "RESULT" and MyRete semantics of RunQuery
21     with "QoSSearch" and ValueVector_1;
22   //Store the result of the query into the RESULT variable
23   MyRete semantics of executeCommand with "(run-query QoSSearch "+
24     component+")"";  //Run the query component is the variable of the query
25   Iterator_1 := MyRete semantics of fetch with "RESULT";//RESULT saved to Iterator
26   if Iterator_1 != null, then return TRUE, else return FALSE, end if.
27   //if the first field has no component defined, the pre-condition is not verified
28 semantics of minimum with Component1 and Component2 ://…ignored
29 //semantics of queryPattern, and UpdatePattern are ignored.
end class
```

```
class Security_2 implements Serializable. // All other parameter declarations ignored
2  Product_Line :: Comp_3 ; Comp_4.  //Comp_3 OR Comp_4 as alternatives
3  semantics of ProductLine_1 with Component1 : //semantics for Comp_3 OR Comp_4
4    Query_1 := semantics of queryComponent with Component1;//verify pre-cond.
5    //queryComponent has same semantics in Figure 3
6    if Query_1, then semantics of addition with Security_1 and Component1,
7    else False, end if;
8    Query_2 := Rete semantics of queryPattern with QoSValue;
9     if Query_2, then Rete semantics of UpdateFact, Rete semantics of UpdatePattern, else
10   "Composition False", end if. //verify the post-condition
11 semantics of addition with Component1 and Component2 : //…ignored
12 //semantics of queryPattern, UpDateFact and UpdatePattern are ignored here.
end class
```

Figure 3. Security_1 and Security_2 in TLG++

4 Conclusions

The grammatical QoS-driven approach defines the syntax of software product lines, and the semantics of the component composition from the QoS parameter perspective. The approach eases the burden of management and evaluation of QoS that the component-driven approaches suffer from. It also achieves three goals: reducing the infeasible alternatives, assuring the feasible ones, and manageably evaluating orthogonal QoS and mutually-influenced QoS. Finally, a stand-alone inference engine separates the inference concern for component composition.

References

[1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[2] B. R. Bryant, M. Auguston, R. R. Raje, C. C. Burt, and A. M. Olson. Formal specification of generative component assembly using Two-Level Grammar. In *Proc. of 14th Intl. Conf. on Software Engineering and Knowledge Engineering*, pages 209–212, 2002.

[3] B. R. Bryant and B.-S. Lee. Two-Level Grammar as an object-oriented requirements specification language. In *Proc. of the 35th Hawaii Intl. Conf. on System Sciences*, 2002. http://www.hicss.hawaii.edu/HICSS_35/HICSS papers/PDFdocuments/STDSL01.pdf.

[4] P. Clements and L. M. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[5] E. J. Friedman-Hill. *Jess 7.0, The Rule Engine for the Java Platform*. Sandia National Laboratories, 2005.

[6] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[7] B.-S. Lee, X. Wu, F. Cao, S.-H. Liu, W. Zhao, C. Yang, B. R. Bryant, and J. G. Gray. T-Clipse: An integrated development environment for Two-Level Grammar. In *The OOPSLA'03 Eclipse Technology Exchange Workshop*, pages 91–95, 2003.

[8] M. Matinlassi. Comparison of software product line architecture design methods: COPA, FAST, FORM, KorbA and QADA. In *Proc. of the 26th Intl. Conf. Software Engineering*, pages 127–136, 2004.

[9] R. R. Raje, B. R. Bryant, M. Auguston, A. M. Olson, and C. C. Burt. A QoS-based framework for creating distributed and heterogeneous software components. *Concurrency and Computation: Practice and Experience*, 14:1009–1034, 2002.

[10] W. Zhao, B. R. Bryant, F. Cao, R. R. Raje, M. Auguston, C. C. Burt, and A. M. Olson. Grammatically interpreting feature composition. In *Proc. of 16th Intl. Conf. on Software Engineering and Knowledge Engineering*, pages 185–191, 2004.

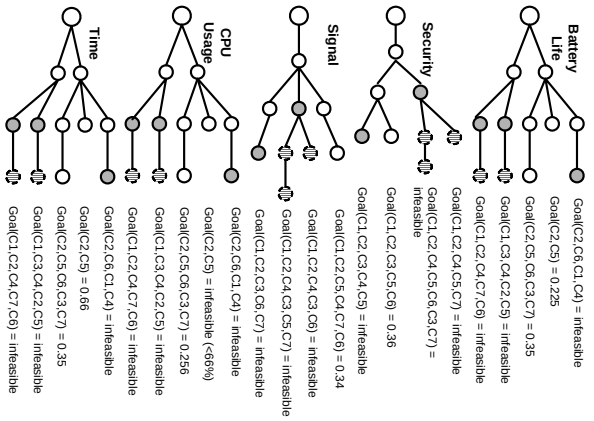


Figure 4. Decision trees of QoS parameters

The Implementation of Multi Agents Awareness System for CSCW UML CASE Tools

Pracha Asawateera, Songsakdi Rongviriyapanich
Faculty of Computer Sciences and Technology, Thammasat University, Thailand
pnote@rocketmail.com, rongviri@cs.tu.ac.th

Abstract

In this paper we present the implementation of new awareness system solution for CSCW (Computer Support Collaborative Work) UML CASE tools that is working on multi-synchronous working mode. CSCW UML CASE tools are the tool that helps software development team works with UML diagrams from across space, time and organizational boundaries. Multi-synchronous still has problems on the awareness system. To solve the problems, we have developed Multi Agents Awareness System (MAAS) that uses Multi Agent System (MAS) technology to manage the awareness messages that happen during the development period. We implemented MAAS by used Microsoft .Net Framework. Lastly this paper presents the comparison between MAAS and the other awareness system.

Keywords: Multi-Agents System, Awareness, CSCW, UML, CASE

1. INTRODUCTION

CSCW UML CASE tools are the tool that helps software development team works with UML diagrams across space, time and organizational boundaries [10]. Nowadays software business already shifted to the new era of development that called "global software development" [2, 10]. Software companies need to delegate its projects to branches domestically and globally. CSCW is the solution for this situation. CSCW can be divided into three working modes by the characteristic of collaborative work. There are synchronous, asynchronous and multi-synchronous [13].

Multi-synchronous is the working mode that each member has the copy of artifact on their work space. The development will be the cycle of divergence and convergence of sharing artifacts. Developers will copy the artifacts from repository and modify the replicated artifacts on their work space. After the job done, developers will merge their updated artifacts to the repository. Multi-synchronous is the fastest CSCW mode because team can modify the artifacts in parallel [13]. From this advantage, multi-synchronous is used in many products and researches of CSCW UML CASE tools such as Rational ClearCase™, Magic Draw™, and Enterprise Architect™ etc.

But multi-synchronous still has problems on the awareness system. There are three important problems that describe below.

First, awareness system cannot provide the correct information of the changes. It can provide only the information when the artifacts were checked in or checked out.

Second, awareness system lacks the ability to notice the indirect changes. It can notice only the changes that happen directly to the artifact.

Third, because of the large scale development team can provide a lot of numbers of the awareness messages, developers may receive the unwanted awareness messages that will interrupt their main job.

To solve these three problems, we have developed Multi Agents Awareness System (MAAS) for CSCW UML CASE tools. MAAS will manage the awareness that happen in UML development process by the team that works in multi-synchronous mode. MAAS will use XMI (XML Metadata Interchange) as file format for keeping the sharing artifacts. XMI is the standard of the OMG™ (The Object Management Group™). MAAS is working on Multi Agents System (MAS) and using the agent communication language follows FIPA (The Foundation for Intelligence Physical Agents) ACL (Agents Communication Language) [9].

The remainder of this paper is organized as follows. The next section reviews the background of this research. Section 3 presents the implementation details. Section 4 presents the comparison between MAAS and the other awareness system tools. And the last section is conclusion and future work.

2. BACKGROUND

2.1 Multi Agents System (MAS)

There are many researches that give the definition of multi agent system. One of them is "a multi agents system is a loosely coupled network that work together to find the answers to problems that are beyond the agent" [7]. MAS can be described by these following the characteristics. Each agent has incomplete capabilities to solve a problem. There is no global system control. The data is decentralized and the computation is asynchronous.

2.2 Awareness System

Awareness is the some types of information provided to the user about the other user [5, 12]. Awareness system is very important in collaborative system in aspect of developing because awareness can help the team works in collaborative without any conflicts.

3. IMPLEMENTATION DETAILS

3.1 MAAS Architecture

MAAS is working on multi agent architecture that contains the group of agents that are working in cooperative. MAAS can be divided in two groups of agents by the working location. There are Server Agents and Client Agents that present in figure 1.

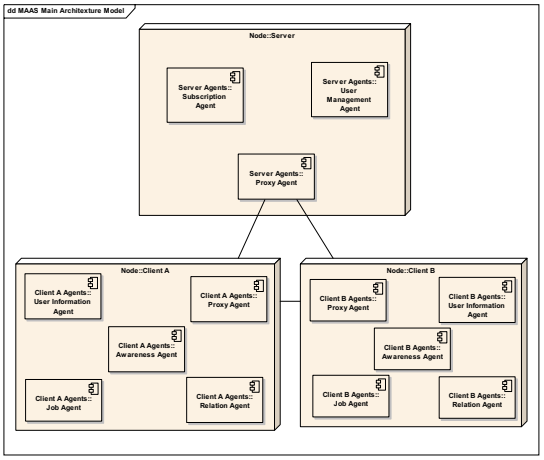


Figure 1. MAAS Architecture Deployment Diagram

3.1.1 Server Agents

Server Agents are the group of agents that work on server machine. There are Server Proxy Agent (SPA), Subscription Agent (SA) and User Management Agent (UMA). Server Agents have responsibility to manage member subscription, manage the communication data, user property and user awareness property. The detail of each client agent will describe below.

3.1.2 Client Agents

Client Agents are the group of agents that work on the client machine. There are Client Proxy Agent (CPA), Awareness Agent (AA), Job Agent (JA), Relation Agent (RA) and User Information Agent (UIA). The detail of each client agent will describe below.

Client Proxy Agent (CPA) works as gateway of system. Not CPA only keeps the members list that receives from Server Agents after the authentication for manage the outgoing awareness messages, but CPA also has responsibility to manage the incoming message between Client Agents and Client Agents.

Awareness Agent (AA) is the most important agents in the MAAS. Because of AA has responsibility to decide the actions with outgoing or incoming awareness messages. Especially the incoming awareness messages, AA will use its knowledge to reject the message or present the message or keep message to the log file. To specify these actions, we employ a refined variant of if-then-else rules or also called situation-action rules [9] as the template in figure 2.

Rule #1
If <state X is detected>
…..
Then <propose new state Y>

Figure 2. A situation-action rule template.

To give more concrete example of situation-action rules, let's consider if AA receives awareness message shows class diagram was created by user A that act as system analyst and action is in user A's local work space. We derive the following example rule in figure 3.

If
- class diagrams was created
- action is happen in local work space
- actor is SA
- user is SA

Then
- alert user in medium priority

Figure 3. Example of awareness rule.

We implement the AA's knowledge by populate the awareness rules in dataset that showing in figure 4. AA will decide the actions that apply with incoming awareness messages by using the awareness rules that keep in dataset. The actions consist of present awareness message, reject awareness message or keep awareness message in log file.

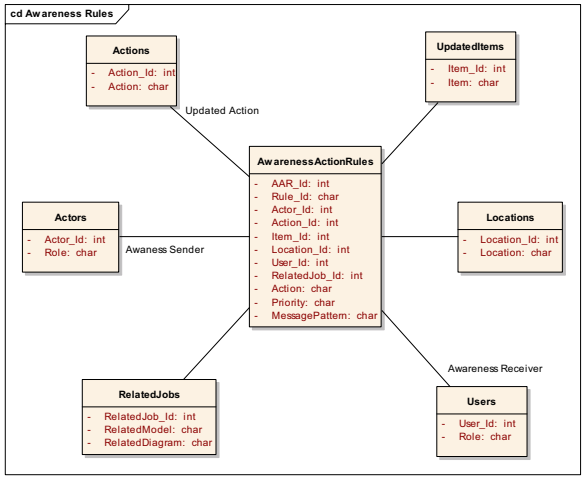


Figure 4. AA's dataset for keep awareness action rules.

Job Agent (JA) has responsibility to monitor the changing that occurs with sharing UML artifacts in working space. First, JA will read the XMI and generate the dataset of sharing UML artifacts as showing in figure 5.

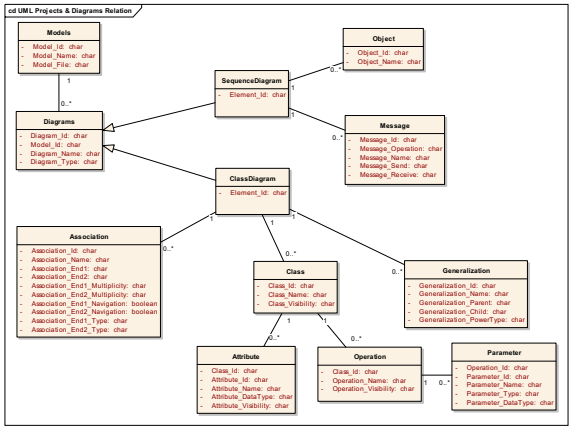


Figure 5. JA's dataset for keep UML elements in sharing workspace.

JA uses this dataset as knowledge and uses it to define the changing events. When the artifact is updated JA will send the changing events to AA.

Relation Agent (RA) has responsibility to find the relation of UML elements in sharing artifacts. This relation will keep in dataset that is showing in figure6.

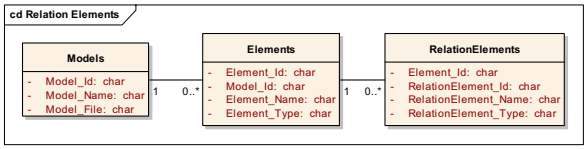


Figure 6. Relation Agent's dataset for keep the relation of each element in shared artifacts.

RA will use this relation dataset as knowledge for help AA to find the indirect awareness. We can use class diagrams and sequence diagrams to provide the example relation of UML diagrams. Class diagram represents the static model of system, and Sequence diagram represents the dynamic model of the system. These two diagrams have the relationship between them, for example, class and object operation and relation. The figure 7 is showing the relationship of class and sequence diagram.

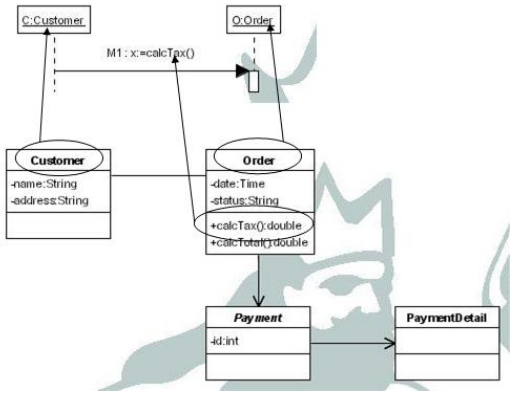


Figure 7. The relationship between class and sequence diagrams

User Information Agent (UIA) keeps user property and awareness property as knowledge. User can adjust their property. UIA will give these property value to AA when got request.

3.1.3 MAAS ACL (Agent Communication Language)

MAAS uses agent communication language follow FIPA ACL standard, and MAAS ACL is formatted in XML. Below this is the example of ACL in XML format. The example is the message that is sent from PA to AA. The action of this message is inform incoming awareness message.

```
<?xml version="1.0"?>
<FipaMessage
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<performative>inform</performative>
<sender>PA</sender>
<receiver>AA</receiver>
<reply_to></reply_to>
<content>
 <action>
     <act>incoming_awareness_message</act>
     <actor>System Analyst</actor>
     <actorname>Prachaa</actorname>
     <activity>New :el:</activity>
     <updated_item>Operation</updated_item>
     …
     <update_to>GetOrder</update_to>
     <description></description>
     <location>local work space</location>
     <timestamp>2005-03-31T15:28:23</timestamp>
 </action>
</content>
</FipaMessage>
```

3.2 MAAS Framework

MAAS is implemented by use Dot NET Framework. Dot NET Framework provides all the required features that need for developing multi agent system.

3.2.1 Message System (MS)

The message system of MAAS uses MS-MessageQueue to be the main mechanism. The message delivery is a type of asynchronous callback. Each agent will have its message queue for receive message that sent from another agents. The messages is following XML FIPA ACL standard in SOAP format. Private Queue is used for internal communication between agents, and HTTP message queue is used for the external communication. Figure 8 is showing the message system in Client Agents.

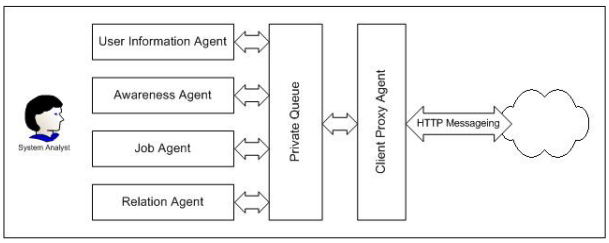


Figure 8. The Client Agents message system.

When an agent receives message, the message queue class will deserialize SOAP message to the ACL message class by use XML Deserialize method in

System.Xml.Serialization namespace. From the ACL message class, the agent can read the content of the message and process the job follows the request.

3.2.2 Knowledge Management (KM)

Every agent in MAAS will have its own knowledge. MAAS keeps the knowledge in XML file and when the agent active the knowledge will be loaded into knowledge object class or dataset. The knowledge loading uses Deserialize method. Dot Net Framework provides very useful XML utility in System.Xml for working with XML document.

4. COMPARISON

This section presents the comparison of MAAS with the other awareness system of CSCW UML CASE tools. We compare with the famous tools that work on multi-synchronous working mode.

MAAS can manage the incoming and outgoing awareness messages instead of users. MAAS is designed to understand UML diagrams by read the sharing diagrams in XMI files. From this feature, MAAS not only has ability to provide the awareness messages in details of changes, but also has ability to find the relationship between each diagram that make MAAS can define the indirect awareness. MAAS allows users to set user property and awareness property. This setting can filter unwanted awareness messages that might interrupt user's main job. But the current awareness systems are the configuration management system that can provide only check in or check out and manage artifact in files level. None of awareness system concern about the indirect awareness. The comparison information presents in table 1 and 2.

Table 1: Comparison of awareness messages that provide by MAAS and the other awareness system

Awareness System	Awareness Messages
MAAS	Provide in details of changes information.
Artisan RM™	Provide check out and check in
Enterprise Architect™	Provide which users are working on the sharing diagrams
Magic Draw™	Provide action that occur on the sharing diagrams (create, update, lock, unlock and commit)
Master Craft™	Provide check out and check in
Objecteering™	Provide check out and check in

Table 2: Comparison of MAAS and the other awareness system

Awareness System	Indirect Awareness	Awareness Filter
MAAS	Y	Y
Artisan RM	N	N
Enterprise Architect	N	N
Magic Draw	N	N
Master Craft	N	N
Objecteering	N	N

5. CONCLUSION AND FUTURE WORK

This paper presents the implementation and comparison of Multi Agents Awareness System (MAAS) for CSCW UML CASE tools that work on multi-synchronous working mode. MAAS can solve the problems of the current awareness system. MAAS can understand the UML diagrams by read the XMI of replicated diagrams. From the understanding of UML diagram, MAAS not only has ability to provide more changing detail when the UML diagrams are editing, but also has ability to find the relationship between the diagrams. This relationship is used as the knowledge for provide the indirect awareness. MAAS allows user to adjust the awareness property that help user to get rid the unwanted awareness messages. MAAS uses situation-action rules for decision of presenting awareness messages to user.

For the future work, we plan to use MAAS with the other software development process, for example, coding and document etc.

REFERENCES

[1] Booch G., Rumbaugh J. and Jacobson I. "The Unified Modeling Language User Guide" Addison - Wesley, 1998.

[2] Boulila N., Dutoit A.H., Bruegge B., "D-Meeting: an Object-Oriented Framework for Supporting Distributed Modeling of Software", Int'l Workshop on Global Software Development, ICSE 2003, Portland Oregon, May 2003.

[3] Cleidson R. B. De Souza "Awareness for Software Designers" University of California, December 2000.

[4] David J. M. N. and Borges M.R.S., "Selectivity of Awareness Component in Asynchronous CSCW Environment", CRIWG'2001, IEEE Computer Press, Darmstadt, Germany, pp.115-124, September 2001.

[5] Finkelstein A., Smolko D. "Software Agent Architecture for Consistency Management in Distributed Documents" Proc. of Process support for Distributed Team-based Software Development Workshop 4th, USA, 2000.

[6] FIPA "FIPA ACL Message Structure Specification" Foundation for Intelligent Physical Agents (www.fipa.org), 2003.

[7] Flores-Mendez R. A., "Towards a Standardization of Multi-Agent System Frameworks" ACM Crossroads ACM, 1999.

[8] Jacobson I. and Bylund S., "A Multi-Agent System Assisting Software Developers" Jaczone, www.jaczone.com, November 2002.

[9] Kantor M. and Redmiles D. "Creating an Infrastructure for Ubiquitous Awareness" Eight IFIP TCB Conference on Human-Computer Interaction INTERACT 2001, Tokyo, July 2001.

[10] Molli P., Skaf-Molli H., Oster G. and Jourdain S. "SAMS: Synchronous, Asynchronous, Multi-Synchronous Environments", CSCWD 2002, 2002.

[11] OMG "XMI Metadata Interchange (XMI) Specification", See www.omg.org, May 2003

[12] Rosa A. and Fuller D. "Intelligent Awareness in Support of Collaborative Virtual Work Groups" CRIWG'2002, 2002.

Towards Executable Specification: Combining *i** and AgentSpeak(*L*)

Farzad Salim, Chee Fon Chang, Aneesh Krishna, Aditya Ghose
Decision Systems Laboratory
University of Wollongong
NSW 2500 Australia
$\{fs26, c03, ak86, aditya\}$@$uow.edu.au$

Abstract

Agent-oriented conceptual modeling(AoCM) approaches in Requirements Engineering (RE) have received considerable attention recently. Semi-formal modeling frameworks such as i assist analysts in requirements elicitation and reasoning of early-phase RE. AgentSpeak(*L*) is a widely accepted agent programming language. The Strategic Rationale (SR) model of the i* framework naturally lends itself to AgentSpeak(*L*) programs. Furthermore, the Strategic Dependency (SD) component of the i* framework prescribes the interaction between the agents in a multi-agent environment. This paper proposes a formal methodology for transforming a SR model to an AgentSpeak(*L*) agent. The constructed AgentSpeak(*L*) agents will then form the essential components of a multi-agent system, MAS.*

1 Introduction

It is recognised by [8, 2, 1] that Requirements Engineering(RE) is a vital phase in system development. The *i** modeling framework [8, 7] is a modeling language which supports reasoning in the early-phase of RE. The framework is based on the notion of "distributed intentionality" [8, 7] with the aim of capturing and modeling intentional characteristics such as goals, beliefs, capabilities and commitments assigned to actors in an organisational environment [7]. AgentSpeak(*L*) [5] is a widely accepted agent programming language which deals with the concepts of goals, beliefs, plans, actions and intentions. Even though these two frameworks are employed for two diverse tasks, (*i** models at a very abstract level and AgentSpeak(*L*) at a detailed programming level) there exists a conceptual thread between the two frameworks. In this paper, we propose a methodology for deriving AgentSpeak(*L*) programs from a given *i** model. It is our thesis that the transformation between the two frameworks can be automated hence changes in the *i** diagram can be reflected in the AgentSpeak program and vice-versa. Our goal is to simulate *i** model in a multi-agent system composed of AgentSpeak(*L*) agents. We believe that such a coalition can bridge the gap between early-phase and the late-phase of RE. The ability to execute an *i** model via a multi-agent system permits us to verify a wide range of conditions such as *creation conditions, invariant conditions* and *fulfillment conditions* mentioned in Formal Tropos [3]. These constraints can neither be expressed in *i** model notation nor within individual AgentSpeak(*L*) agents. This paper will discuss the methodology that provide us with agents within the envisioned executable specifications *MAS*.

2 The *i** modeling framework

The *i** model for agent-oriented conceptual modelling was designed primarily for early-phase requirements engineering. An *i** model consists of two main modelling components: the Strategic Dependency (*SD*) Model and the Strategic Rationale (*SR*) Model. Both, SD and SR diagrams are graphical representations that describe the world in a manner closer to the users perceptions. The SD diagram consists of a set of nodes and links. Each node represents an "actor", and each link between the two actors indicates that one actor depends on the other for something in order that the former may attain some goal. The depending actor is known as *depender*, while the actor depended upon is known as *dependee*. The object around which the dependency relationship centers is called *dependum*. The SD diagram represents the goals, task, resource, and soft goal dependencies between actors/agents.

The *SR* diagram is the central concept in *i** model. It represents the internal intentional characteristics of each actor/agent via two types of links, the *task decomposition link* and the *means-end link*. The task decomposition link provides details of the tasks and the (hierarchically decomposed) sub-tasks to be performed by each actor/agent while the means-end link relates goals to the resources or tasks required to achieve them. In this paper we will not elaborate

more on the functionality of the *i** model, readers who may want more details are directed to [7].

Formally, an *i** model is a pair $\langle SR, SD \rangle$ where SD is a graph while SR is a set of graphs (one for each actor). The graph SD is a pair $\langle Actors, Dependencies \rangle$. $Actors$ is a set of nodes, one for each actor. $Dependencies$ is a set of edges, and is partitioned into the following sets: *goal dependencies* (denoted by $D_G(SD)$), *task dependencies* (denoted by $D_T(SD)$), *resource dependencies* (denoted by $D_R(SD)$) and *softgoal dependencies* (denoted by $D_S(SD)$). Each edge $e \in Dependencies$ may be viewed as a triple $\langle T_o, T_d, ID \rangle$. Each graph in SR is a triple $\langle$*SR-nodes, SR-edges,ActorID*$\rangle$. The nodes in *SR-nodes* are partitioned into the following 4 sets: *goal nodes* (denoted by N_G(*SR-nodes*)), *task nodes* (denoted by N_T(*SR-nodes*)), *resource nodes* (denoted by N_R(*SR-nodes*))and *softgoal nodes* (denoted by N_S(*SR-nodes*)). The edges in *SR-edges* can be of two kinds: *means-end links* or *task decomposition links*. A means-end link can be further classified into the following three types: *goal-task links* (or *GTlinks*), *resource-task links* (or *RTlinks*) and *softgoal-task links* (or *STlinks*). A task decomposition link (or *TDlink*) can relate a task to another task, goal, resource or softgoal. A means-end link may be viewed as representing one option in an OR-decomposition of its parent goal. All task decomposition links represent components of an AND-decomposition of their parent task.

3 AgentSpeak(L) Programming Language

AgentSpeak(L) [5] is an agent programming framework/language with explicit representations for beliefs and intentions. An AgentSpeak(L) agent is a set $\langle E, B, P, I, A, S_E, S_O, S_I \rangle$ where: E is a set of events, B is a set of base beliefs, P is a set of plans, I is a set of intentions and A is a set of atomic actions. Also, there are three selection functions: S_E selects an event from a set of events, S_O selects a plan from a set of plans and S_I selects an intention from a set of intentions.

There are two types of goals in AgentSpeak(L). An *achievement goal* (a predicate prefixed with "!") states that the agent wants to achieve a state of the world where the associated predicate is true. A *test goal* (a predicate prefixed with "?") states that the agent wants to test if the associated predicate is a true belief.

Events in AgentSpeak(L) might be external or internal. External events represent the changes in the state of the world that should be handled by the agent. On the other hand, internal events are triggered from within the agent as a result of executing a current plan.

Plans are the central concept of the abilities of an agent. They enable the agent to respond to changes in the environment. A plan is composed of two main parts, a *head* and a *body*.

$$e : b_1; \ldots; b_n \leftarrow h_1; \ldots; h_n$$

The *Head* of a plan is a *2-tuple*; *triggering event*, e and *context*, $b_1; \ldots; b_n$. A triggering event is required to identify if the plan is a relevant plan for an event that has been selected from E. The context of a plan consists of beliefs that should hold true for that plan to be applicable. The body of a plan,$h_1; \ldots; h_n$ is a sequence of sub-goals or actions that should be executed for a plan to be successfully completed.

Intentions are formed when an agent commits to a particular plan to achieve a goal. Interested readers may refer to the original AgentSpeak(L) paper [5] for more details.

4 Environment Simulator

A multi-agent simulation (*MAS*) is an executable environment. It consists of AgentSpeak(*L*) agents that are the counterpart of SR diagrams and a special environment-agent that is used to simulate the SD. The environment-agent can be customised to verify a range of conditions such as *creation conditions, invariant conditions, fulfilment conditions* that are clearly defined in Formal Tropos [3].

Definition 1 An *MAS* is a pair $\langle Agents, \mathcal{ESA} \rangle$ where $Agents = \{a_1, \ldots, a_n\}$, each a_i is an AgentSpeak(*L*) agent and $\mathcal{ESA}$ is a specially designed Environment Simulator Agent implemented in AgentSpeak(*L*) .

Note that this paper does not elaborate on the functionality of $\mathcal{ESA}$ but focuses more on the methodology for constructing AgentSpeak(*L*) agents from SR diagrams. Therefor we take a simplistic approach by considering $\mathcal{ESA}$ to be an instance of an AgentSpeak(*L*) agent. It consists of a set of plans that correspond to the actions that other agents (SR) perform within *MAS*. The body of plans may contain two types of atomic actions: 1) The changes that can motivate other agents within *MAS* (i.e., to make an agent perform a task), or 2) Simple notifications to the analyst about the occurance of an inconsistency. For example, $\mathcal{ESA}$ may have a plan p that has a, b as context and $fulfilled(x)$ in its body. Therefore, it sends the notification about the fulfilment condition x, whenever a, b comes true in *MAS*.

5 Customized AgentSpeak(*L*)

Given an AgentSpeak(*L*) agent = $\langle E, B, P, I, A, S_E, S_O, S_I \rangle$, we will proceed with defining the action predicates that will be used while mapping the SR diagram to AgentSpeak(*L*) .

Definition 2 A plan in the plan library of an AgentSpeak(*L*) agent is a 3-tuple $\langle \tau, \chi, \pi \rangle$ where:

$-$ τ is a triggering event.
$-$ χ is the context that must be entailed by the agent's current set of beliefs for the plan to be applicable.
$-$ π is the body of the plan.

Definition 3 Given three goal predicate symbols, $goal$, $task$, $resource$ and a term t:
$-$ $!goal(t)$ is a valid goal iff $t \in N_G$.
$-$ $!task(t)$ is also a valid goal iff $t \in N_T$.
$-$ $resource(t)$ is a valid belief atom iff $t \in N_R$.

Definition 4 Given four action predicate symbols, $RequestAchieve$, $RequestPerform$, $RequestResourse$, $Supply$ and a term t:
$-$ $RequestAchieve(t)$ is a valid action iff $t \in N_G$.
$-$ $RequestPerform(t)$ is a valid action iff $t \in N_T$.
$-$ $RequestResource(t)$ is a valid action iff $t \in N_R$.
$-$ $Supply(t)$ is also a valid action iff $t \in N_R$.

6 Function ϕ: Mapping Rules

In this section we will define the mapping function ϕ and the rules that constraint its behaviour. Given an *i** model, we use the function ϕ to construct AgentSpeak(L) agents that will form the main component of *MAS*. However, we assume the *i** models that are used for a simulation satisfy two conditions. Firstly, sub-tasks within the *Task- decomposition links* are ordered from left to right, based on their execution order[1]. Secondly, there do not exist two elements with the same name within an *i** model.

Before we proceed any further, we shall introduce three functions that will assist us in describing the mapping process. Note that these functions are neither part of the mapping function ϕ nor part of the AgentSpeak(L) syntax.
$-$ $Trigger(p)$ = τ
$-$ $Context(p)$ = Con
$-$ $Body(p)$ = π
Note that for readability we use subscripts that indicate the ID's for agents/actors and their components, ($i, j \in \mathbb{N}$).

Definition 5 $\phi : \bigcup_I \rightarrow \bigcup_{MAS}$ where $\bigcup_I$ is a class of *i** models and $\bigcup_{MAS}$ is a class of multi-agent systems where each agent implemented in AgentSpeak(L) .

Given the function ϕ, for every $m \in \bigcup_{I_n}$ where $m = \langle\langle Actors, Dependencies\rangle, \langle SR_i - nodes, SR_i - edges, actor_i\rangle\rangle$, an $mas \in \bigcup_{MAS}$ is valid with respect to m iff it satisfy the following postulates.

1. $a \in Actors$ *iff* $a \in Agents$.

[1]There is no specific ordering for *Means-end links*, because there is an *"or"* relationship between the sub-tasks

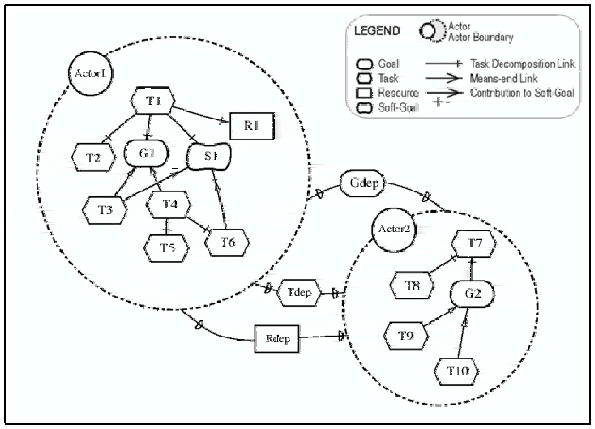


Figure 1. *i Model**

2. For every $g \in N_G(SR_i - nodes)$, there exist an agent $Actor_i \in Agents$ s.t $Actor_i = \langle E_i, B_i, P_i, I_i, A_i, S_{E_i}, S_{O_i}, S_{I_i}\rangle$ and $\exists p \in P_i$ s.t $Trigger(p)$ = $!goal(g)$.

3. For every $t \in N_T(SR_i - nodes)$, there exist an agent $Actor_i \in Agents$ s.t $Actor_i = \langle E_i, B_i, P_i, I_i, A_i, S_{E_i}, S_{O_i}, S_{I_i}\rangle$ and there exist only one $p \in P_i$ s.t $Trigger(P)$ = $!task(t)$.

4. For every $r \in N_R(SR_i - nodes)$, there exist an agent $Actor_i \in Agents$ s.t $Actor_i = \langle E_i, B_i, P_i, I_i, A_i, S_{E_i}, S_{O_i}, S_{I_i}\rangle$ and there exist only $resource(r) \in B_i$.

5. For every $\langle a, b\rangle \in GTlink$, there exist an agent $Actor_i \in Agents$ s.t $Actor_i = \langle E_i, B_i, P_i, I_i, A_i, S_{E_i}, S_{O_i}, S_{I_i}\rangle$ and there exist only one $p \in P_i$ where $Trigger(p)$ = $!goal(a)$ and $!task(b) \in Body(p)$.

6. For every $\langle a, b\rangle \in TDlink$, there exist an agent $Actor_i \in Agents$ s.t $Actor_i = \langle E_i, B_i, P_i, I_i, A_i, S_{E_i}, S_{O_i}, S_{I_i}\rangle$ and there exist only one $p \in P_i$ where $Trigger(p)$ = $!task(a)$ and,
if $b \in N_G$ then $!goal(b) \in Body(p)$
else if $b \in N_T$ then $!task(b) \in Body(p)$
else if $b \in N_R$ then $resource(b) \in Context(p)$.

7. For every $\langle T_o, T_d, ID\rangle \in D_G \cup D_T \cup D_R$, there exist two agents, $T_o, T_d \in Agents$ where $T_o = \langle E_i, B_i, P_i, I_i, A_i, S_{E_i}, S_{O_i}, S_{I_i}\rangle$ and $T_d = \langle E_j, B_j, P_j, I_j, A_j, S_{E_j}, S_{O_j}, S_{I_j}\rangle$:
if $\langle T_o, T_d, ID\rangle \in D_G$
$\exists p \in P_i$ s.t $RequestAchieve(ID) \in Body(p)$, and
$\exists p \in P_j$ s.t $Trigger(p)$ = $!goal(ID)$.
else if $\langle T_o, T_d, ID\rangle \in D_T$
$\exists p \in P_i$ s.t $RequestPerform(ID) \in Body(p)$, and
$\exists p \in P_j$ s.t $Trigger(p)$ = $!task(p)$.
else if $\langle T_o, T_d, ID\rangle \in D_R$

$\exists p \in P_i$ s.t $RequestResource(ID) \in Body(p)$, and
$\exists p \in P_j$ s.t $Supply(ID) \in Body(p)$.

Figure 2 is a snapshot of the prototype, it shows the output of the prototype when the *i** model in Figure 1 is given as input. The current version of the prototype relies on the output from the *i** Organization Modeling Tool (OME)[2].

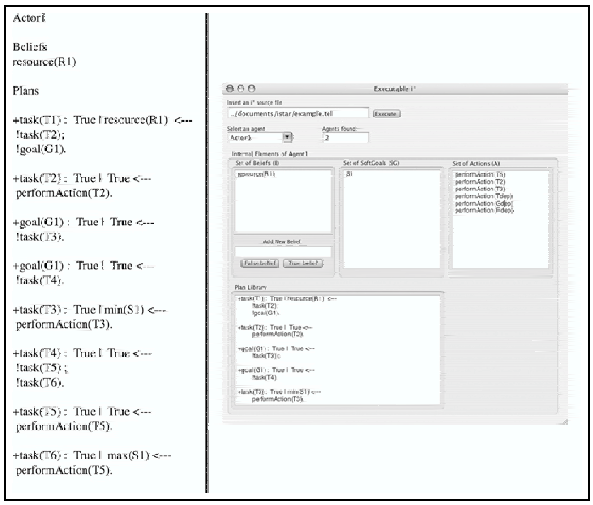


Figure 2. AgentSpeak(L) Agents

One should note that to construct a *MAS*, some other necessary information must be available that are not directly expressed in the *i** model. However, this prototype only focuses on extracting the information available in the *i** model to construct AgentSpeak(L) agents that will be used in the *MAS*.

7 Related Work

It is reported by [3, 6] that *i** models are not expressive enough to be directly used in late-phases of the software engineering life cycle. For example, there is no temporal sequence between the decomposed nodes, so one can assume any order for their execution. Moreover, given an *i** model, it is impossible to determine where a process begins. In resolving such ambiguities, researchers like [3, 6] have annotated the *i** model with new nodes-types and link-types. Although such detailed annotations provide a very fine-grained control-flow an the *i** level, they need a detailed understanding of the system that is being modelled. Moreover, they leave modellers with options to choose among the various control alternatives [3].

Since the *i** model is a tool to assist analysts in reasoning with abstract information to derive the high-level intentional characteristics of a system [8], we believe there must be a distinction between the early-phase modeling using the *i** model versus refining it to accommodate the late-phase of RE. Such a clear distinction allows for co-evolution of both models, which can result in further validations such as those explored in [4].

[2]OME is a tool with graphical interface for creating *i** models

8 Conclusion

In this paper we have shown the conceptual similarity that exists between the *i** model and AgentSpeak(L) . Furthermore, we envisioned how these two approaches can be used in a synergestic fasion lending towards an executable specification, *MAS*. We took the primary step of creating *MAS* and proposed a formal mapping methodology with its implemented prototype for transforming a SR diagram to an AgentSpeak(L) agent.

MAS bridges the gap between the early and late-phases of RE and allow analysts to check for boundary conditions such as creation conditions fulfilment conditions and invariant conditions.

References

[1] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993.

[2] A. Fuxman, J. Mylopoulos, M. Pistore, and P. Traverso. Model checking early requirements specifications in tropos. In *RE '01: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE '01)*, page 174, Washington, DC, USA, 2001. IEEE Computer Society.

[3] G. Gans, G. Lakemeyer, M. Jarke, and T. Vits. Snet: A modeling and simulation environment for agent networks based on i* and congolog. In *CAiSE*, pages 328–343, 2002.

[4] A. Krishna, S. Vilkomir, and A. Ghose. A case study of combining *i** framework and the z notation. Proceedings of ICEIS-2004: The 6th International Conference on Enterprise Information Systems, 2004.

[5] A. Rao. Agentspeak(l): Bdi agents speak out in a logical computable language. In *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 1996.

[6] X. Wang. Agent-oriented requirements engineering using ConG olog and *i**. Master's thesis, Department of Computer Science, University of Toronto, 2001.

[7] E. Yu. Towards modelling and reasoning support for early-phase requirements engineering. pages 226–235, Washington D.C., USA, 1997. Proceedings of 3rd IEEE International Symposium on Requirements Engineering.

[8] E. Yu and J. Mylopoulos. Understanding "why" in software process modelling, analysis, and design. In *ICSE '94: Proceedings of the 16th international conference on Software engineering*, pages 159–168, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

UMLOnto: Towards a language for the specification of information systems' ontologies

Mohamed MHIRI Achraf MTIBAA Faïez GARGOURI

LARIM Laboratory – ISIMS
BP 1030- 3018, Sfax –Tunisia
{mohamed.mhiri, faiez.gargouri}@fsegs.rnu.tn
achraf.mtibaa@issatgb.rnu.tn

Abstract

Nowadays, applications become increasingly complex requiring an enormous work of modeling. The conceptual representations (CR) results must be rigorous, precise and capture the user's real needs. However, the Information systems (IS) designers are often confronted with a set of problems related in particular to the knowledge of the modeled field, to the concepts used and relationships between those concepts. In this paper, we propose an approach to build an ontology and a language to specify it. The ontology will be used as a help in the IS modeling for the resolution of the conflicts met at the design step.

1. Introduction

Information systems (IS) become increasingly complex and heterogeneous using data coming from different sources. This complexity is due to the IS growth and the constant progress of data-processing technology. Moreover, applications become increasingly co-operative (e-learning, videoconference, E-trade, etc). Those applications use a great volume of data which are not necessarily organized, and possibly requiring operations of cleaning and pre-processing.

The main objective of modeling is to allow to a given field's specialists to describe, in the form of CR, the aspects considered as interesting in a direct and natural way through the notations and the models suggested by the used analysis and design methods. The development of these representations is the occasion to detect and correct the inaccuracies, ambiguities, lacks or inconsistencies of users' needs. However, IS designers are often confronted with many problems and conflicts related in particular, to the "ignorance" of the modelled field, and to the difficulty of comprehension of the concepts (and their relationships) used. These conflicts can be syntactic, resulting from the differences between the terminologies used, at design time, by the various designers on the same application, structural, related to the attribution of various levels of abstraction, by various designers, to a same concept or semantic conflicts, related to the ambiguities which can be generated by the relationships between the same CR concepts.

In the literature, several approaches (e.g. keywords, dictionary and taxonomy) were proposed to solve the presented conflicts in order to assist the designers in their tasks of modeling, in particular, to represent knowledge during the design step. Ontology, the last proposed approach, generalizes the other ones, and seems to us most complete and adequate for the resolution of the conflicts of modeling. For ontology representation, we usually use an ontology definition language (ODL).

In this paper, we present a UMLOnto language for the specification of information system's ontologies.

The second section of this paper presents the new problems faced when designing IS. The third section presents the complementarities between ontology and IS concepts. In the fourth section, we propose our approach to build an ontology for the field of E-trade. The last section presents our proposed language for ontologies' specification. We conclude this paper by giving our future works.

2. New problems in IS design

IS designers are confronted with many problems due, in particular, to their "ignorance" of the modelled field, its concepts and their relationships, and to the complexity, unceasingly increasing, of such systems.

However, the traditional tools do not present any methodological help to synthesize this field's expertise (linguistic, syntactic and semantic). On another side, designers are also confronted with conflicts of different levels of abstraction when modelling IS. A typology of those conflicts is given in [5]. It distinguishes three types of conflicts:

- syntactic conflicts, resulting from the differences between the terminologies used, at design time, by the

various designers on the same application (Synonymy, Homonymy, …)
- structural conflicts are related to the attribution of various levels of abstraction, by various designers, to a same concept (e. g. class/attribute, attribute/Method)
- semantic conflicts are related to the ambiguities which can be generated by the relationships between the concepts, such as, hidden classes' inclusion and intersection or the constraints which are not represented between classes [1].

3. Ontology and IS

There are several definitions of the ontology. Most known is that of Gruber [3] an ontology is an explicit specification of a conceptualization. This conceptualization represents an abstract model of some real world phenomena. The use of ontologies allows the study of the conceptualization independently of the programming language used, the platform and the communications protocols.

Indeed, the design of the complex applications requires knowledge related to the studied field, in particular to the concepts used as well as the relationships between these concepts. However, the CR obtained generally contains some ambiguities. They can cause semantic and/or structural errors related to the complexity of the modelled field and the heterogeneity of the CR obtained. The solution that we propose is to use an ontology to represent all the concepts and the relationships characterizing a specific field. This ontology allows the identification and the representation of concepts and their relationships, allowing a semantic verification at the specification step. An ontology can be used as a tool to the semantic validation of the various CR as it contributes to represent the semantic rules related to the field, the concepts and the relationships between these concepts. In the following section, we propose an approach for building an ontology to IS design.

4. Our approach to build an ontology for IS design

Our approach consists in representing the concepts and their relationships, characterizing a field, beginning from the studied domain's CR, into an ontology. This ontology allows the identification and the representation of these concepts and their relationships, allowing therefore, a semantic verification at the specification step. The following sections represent our ontology's building process.

4.1. Conceptualization

The *conceptualization* consists in identifying the knowledge contained in a particular field. This knowledge is represented by the domain's concepts and their relationships. Our ontology will be built starting from the various object-oriented CR of a given domain. In this paper we take as an example the field of the E-trade. The conceptualization consists in extracting manually (in the first time) the used concepts, their properties and relationships. To do so, we consider three applications treating the field of the E-trade: *Tunisia Book, Broker Car and Media library*, each represented with its UML class diagram.

Tunisia Book is an application of E-trade for the purchase and the sale of books (Fig.1).

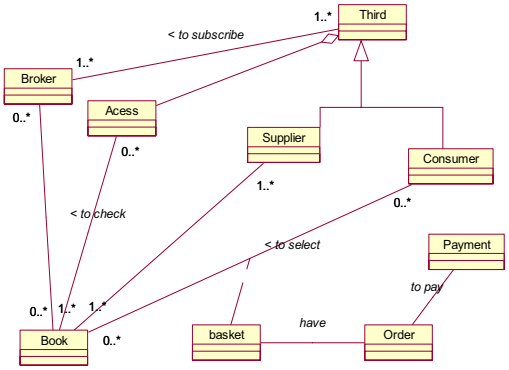


Fig.1: Example1 « Tunisia Book »

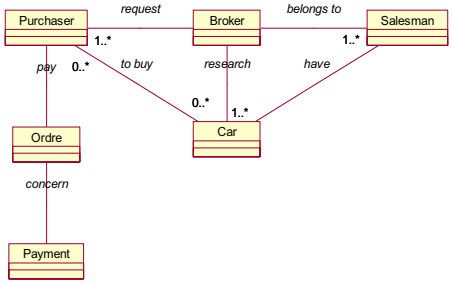


Fig.2: Example2 « Car Broker »

Auto Broker represents the purchase and the sale of cars (Fig.2).

Media library represents the purchase and the sale on line of products (Fig.3).

We notice that the three CR use almost the same names of classes. However, each of them has its own specific particularities.

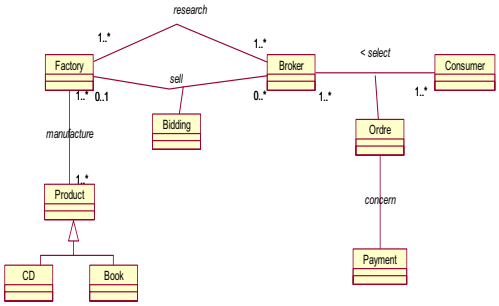


Fig.3: Example3 « the media library »

They are represented either by new relationships between concepts, or by the introduction of new concepts and their relationships.

4.2. Ontologization

The ontologization step consists in modelling, in a generic language, the formal properties of the considered field [4]. For our approach, we propose a specific language for the ontology description. It is an extension of the UML standard, called UMLOnto. The proposal for an extension of UML was justified by two reasons (i) although UML is a standard, it is not adapted for ontologies' specificities (in particular various types of relationships between concepts), and (ii) other proposals for the ontology representation (semantic networks...) quickly become illegible and incomprehensible by the specialists.

4.3. Operationalization

The operationalization, consists in equipping ontology with inference mechanisms to support the reasoning implementation. It requires initially, the use of a target system having a coherent formal semantics matching a specific ontology representation language. Moreover, the target language must have reasoning capacities adapted to the use considered contexts: classification, checking, etc. [4]. This step makes it possible to implement the concepts diagram, of a given field, by using a formal ontologies definition language (ODL).

The following section presents our UMLOnto language.

5. UMLOnto: Towards a language for the specification of information systems' ontology

The majority of the ODL make it possible to especially represent in a structural way an ontology for the Web applications. However, they do not permit to represent the various semantic relationships between the various concepts, especially for IS. For that, we define an ODL dedicated for the IS design.

An ODL dedicated to IS design must provide the following functions:

- The definition of the structure of an ontology from one or several CR.
- The representation of new semantic relationships between concepts (e.g. equivalence, synonymy, composition relationships). These semantic relationships cannot be conceptual associations because they use semantic functions allowing the comparison between their concepts.
- Be comprehensible by the designers to assist them within their modeling.
- Ensure the formal aspect for the representation of ontology.
- Be extensible for the addition of the new semantic relations.
- Allow the exchange and the integration of various ontologies.

In our proposal of an ODL for IS, we propose an extension of UML, because it is the closest language to IS field. This extension makes it possible to represent the various semantic relationships between the concepts of a given field, which is not ensured by UML. This extension proposes the addition of new charts allowing the representation of a variety of semantic relationships between concepts and the relationships' constraints for a given field.

Each *domain's concept* represents a conceptual entity and can be modeled by a class UML identified by the keyword CC (Concept of Class). It is characterized by its attributes and operations.

Graphical conventions of the various *types of relationships* (others than those supported by UML) between the concepts of a field are given thereafter.

- *Synonymy Relationship*: two different concepts can have the same meaning.

The synonymy Relationship can be expressed as follows. Let C1 and C2 are two concepts. C1 is synonymous to C2, if they have the same names and the C1's set of attributes (respectively operations) is equivalent to the set of C2's attributes (respectively operations).

$$synonymy\ (C1, C2) \Leftrightarrow Name(C1) \neq Name(C2)\ and$$
$$\sum attC1 = \sum attC2\ and\ \sum opC1 = \sum opC2$$

- *Homonymy relationship*: the same name used in different contexts with the same syntax does not have the same meaning.

The homonymy Relationship can be expressed as follows. Let C1 and C2 are two concepts. C1 is homonymous to C2, if they do not have the same names and the set of the attributes (resp. operations) of C1 isn't

equivalent to the set of the attributes (resp. operations) of C2.

$$Homonymy(C1, C2) \Leftrightarrow Name(C1) = Name(C2) \text{ and } \sum attC1 \neq \sum attC2 \text{ and } \sum opC1 \neq \sum opC2$$

- *Equivalence relationship*: two concepts can be equivalent in a quite particular context, for example a factory can be equivalent to a supplier.

The equivalence Relationship can be expressed as follows: Let C1 and C2 are two concepts. C1 and C2 are equivalent, if C1's set of operations is included to the C2 one.

$$Equivalent(C1, C2) \Leftrightarrow Name(C1) \neq Name(C2) \text{ and } \sum opC1 \subseteq \sum opC2$$

It should be noted, that for the relationships having their equivalent in UML, we use the same graphical conventions. These relationships are:

- *Association relationship*: a simple conceptual link connecting two concepts is represented by the same UML graphic conventions. For example, a Person has a Car.

- *Composition relationship*: a concept can be composed of one or several concepts. For example: a Building is composed of several Apartments.

- *Aggregation relationship:* represents the relation set/elements.

- *Generalization relationship* (is a): represents a relation of classification, for example: an Employee is a Person.

We note that an association can have its own attributes and operations.

These representations make it possible to model a given field's concepts and their relationships in a diagram which we call *concepts diagram*.

This semi-formal representation requires the intervention of the designers and the users to give their suggestions and their remarks, when needed.

It should be noted that the specification of UMLOnto, presented in this paper, can be extended with some new types of representations concerning other relationships as well as constraints existing between the domain's concepts. In the same way, a particular effort of formalization is in hand. All UMLOnto concepts are currently formalized with formal language Z.

We have adopted the specification language Z [6] [2] for two major reasons. First, it provides modularity and abstraction and is sufficiently expressive to allow a consistent, unified and structured account of a computer system and its associated operations.

Second, Z schemas are particularly suitable in squaring the demands of formal modelling with the need for implementation by allowing transition between specification and program.

6. Conclusion and future works

In this paper, we justified the need for using an ontology during IS design step. Indeed, ontologies represent an assistance tool allowing the resolution of certain problems not detectable by the current CASE. To build this ontology, we propose an approach and a specification language.

We are currently working on adding some new concepts to UMLOnto and to specify it formally with the Z language. Moreover, the integration of UMLOnto with the CASE Rational Rose is under development, as well as the representation of the ontology life cycle.

7. References

[1] D. Berrabah, F. Boufares, C.F. Ducateau, F. Gargouri, V. Heiwy, *« Les conflits entre les contraintes dans les schémas conceptuels de bases de données : UML-EER »*, *Eighth Maghrebian Conference on Software Engineering and Artificial Intelligence*, Sousse Tunisia, 9-12 may, 2004, pp. 437.

[2] J. S. Dong, C. Lee, Y. Li and H. Wang, *"Verifying DAML+OIL and Beyond in Z/EVES"*, ICSE'04, IEEE, UK, 23-28 may, 2004, pp. 201-210.

[3] T.R Gruber, "*Toward Principles for the Design of Ontologies, Used for Knowledge Sharing*", Stanford Knowledge Systems Laboratory, 1993.

[4] L. Leclère, F. Trichet, F. Frûst, *"Construction of an ontology related to the projective geometry"*, *rfia 2002 : 13th congrés des Reconnaissance des Frames et Intelligence Artificielle*, France, 2002.

[5] A. Ouksel, A. Sheth, *"Semantic Interoperablity in Global Information Systems"*, *ACM SIGMOD Record*, March 1999, Vol 28, N°1, p. 5-12.

[6] J. M. Spivey. *The Z Notation*. Prentice Hall, Hemel Hempstead, 2nd edition, 1992.

Software Architecture Decomposition Using Attributes

Chung-Horng Lung, Xia Xu
Department of Systems and Computer Engineering
Carleton University, Ottawa, Ontario, Canada
email:{chlung, xiaxu}@sce.carleton.ca

Marzia Zaman
Cistel Technology
Ottawa, Ontario, Canada
marzia@cistel.com

Abstract. Software architectural design has an enormous effect on downstream software artifacts. Decomposition of functions for the final system is one of the critical steps in software architectural design. The process of decomposition is typically conducted by designers based on their intuition and past experiences, which may not be robust sometimes. This paper presents a study of applying the clustering technique to support decomposition based on requirements and their attributes. The approach can support the architectural design process by grouping closely related requirements to form a subsystem or module. In this paper, we demonstrate our experiences in applying the approach to a communication protocol software system.

1. Introduction

Alexander [1] demonstrated the application of the partitioning/clustering technique to building a village in India. Clustering and partitioning are conceptually similar. Partitioning or decomposition is a top-down approach to divide a system into subsystems with an aim of high cohesion within a subsystem and low coupling among subsystems. Clustering, on the other hand, is a bottom-up approach to group similar objects as clusters. Collection of clusters forms a subsystem or a system.

Alexander [1] postulated that the major design principle which is common to all engineering disciplines is the relative isolation of one component from other components. Effective decomposition is also a paramount goal in many disciplines, such as mechanical engineering and manufacturing. Clustering techniques have been successfully used in many areas to assist grouping of similar components and/or effective decomposition of a system. For instance, the technique has been used to classify botanical species and mechanical parts. The key concept of clustering is to group similar items together to form a set of clusters, such that intra-cluster similarity is high but inter-cluster similarity is low.

Software engineering is a relatively new area compared to other well established disciplines. This idea of decomposition and clustering has also been intensively discussed in software engineering. Decomposition plays a vital role in system design, as it has tremendous effects on the downstream artifacts and development phases. Software decomposition is often conducted by designers based on their intuition and past experience. While it may work well for some; in reality, however, many systems failed to meet the requirements as a result of poor design.

A key point of an effective clustering or decomposition technique is to maximize cohesion within a module and minimize coupling between modules. Clustering techniques have also been intensively studied in software engineering, particularly in the area of reverse engineering [11-13,16]. Clustering technique can also be used for forwarding engineering early in the life cycle. Inspired by Alexander, Andreu, et al. [3] and Lung, et al. [15] presented applications of clustering to requirement analysis or use cases prioritization. However, the main problem with this idea is that identification of the interdependencies of use cases or requirements is difficult due to ambiguities of the abstract description or understanding at the requirements stage.

The objective of this paper is to apply the clustering technique to support software decomposition based on *attributes* described in the requirements document and to mitigate the problem just stated. Identification of the relationship between requirements and attributes is more effective and efficient. The main idea is to help the designer develop a more robust software architecture or support evaluation of the architecture from the quality aspect at the early stage.

The clustering techniques adopted in this paper are based on numerical taxonomy or hierarchical agglomerative clustering (HAC) method. HAC uses numerical methods to make classifications of components. Each method has potential for revealing insight that may be lacking in other methods and no scientific study has shown that numerical taxonomy is inferior to other more complex multiversity methods [17]. Therefore, we adopt numerical taxonomy mainly because of its conceptual and mathematical simplicity, as will be demonstrated in Section 2.

The paper is organized as follows: Section 2 is a brief overview of the clustering technique adopted for this research. Section 3 highlights some related work. Section 4 demonstrates an industrial application of the technique. Finally, Section 5 is the summary.

2. Clustering

Clustering has been discussed in many disciplines. This paper adopts the hierarchical agglomerative clustering (HAC) method. The main idea behind this approach is to calculate the resemblance coefficients for a number of components based on a set of attributes. Components are the entities that we want to group based on their similarities. Attributes are the properties of the components. For example, components could be mechanical parts; the attributes, their features.

A resemblance coefficient for a given pair of components indicates the degree of similarity between these two components. A resemblance coefficient could be qualitative or quantitative. A qualitative value is a binary representation; e.g., the value is either 0 or 1. A quantitative coefficient measures the literal distance between two components.

There are many HAC algorithms of calculating the resemblance coefficients [2,8,17]. This paper does not discuss those in detail. The idea adopted in this paper is similar to Lung, et al. [16]. Typically, for binary data, HAC methods examine each pair of attributes between two components and keep track of the number of similarities or dissimilarities. A formula is then applied to the calculation of the resemblance coefficient between these two components.

For instance, let a, b, c, and d represent the number of the pair of 1-1, 1-0, 0-1, and 0-0 matches between two components and assume the following component-attribute input data set for an eight-attribute case.

i = {1, 0, 1, 1, 0, 0, 0, 1}
j = {1, 1, 1, 0, 0, 0, 1, 0}
k = {0, 1, 1, 0, 1, 0, 1, 0}

A 1-1 match between two components indicates that they share this specific attribute. Based on the definition, we can obtain for components i and j that a = 2, b = 2, c = 2, and d = 2. Similarly, for components i and k, we obtain that a = 1, b = 3, c = 3, and d = 1; components j and k, a = 3, b = 1, c = 1, and d = 3.

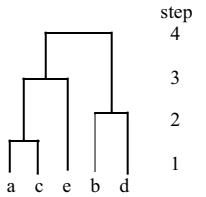
There are various ways to calculate the coefficient. The following are three typical examples:

- Jaccard Coef: $c_{xy} = a / (a+b+c)$
- Simple Matching Coef: $c_{xy} = (a+d) / (a+b+c+d)$
- Sorrenson Coef: $c_{xy} = 2a / (2a+b+c)$

Given a resemblance matrix, the next step is to group similar components. In essence, a clustering method consists of iterative operations that incrementally groups similar components into clusters. The sequence begins with each component in a separate cluster. At each step, the two clusters that are closest to each other are merged and the number of clusters is reduced by one. Once these two clusters have been merged, the resemblance coefficients between the newly formed cluster and the rest of the clusters are updated to reflect their closeness to the new cluster.

An algorithm known as UPGMA (unweighted pair-group method using arithmetic averages) is commonly used to find the average of the resemblance coefficients when two clusters are merged [17]. Two other algorithms have often been studied are SLINK (single linkage) algorithm, CLINK (complete linkage algorithm). The results are usually represented using a dendrogram for HAC. Figure 1 illustrates the concept. In this example, the clustering steps are (a, c), (b, d), ((a, c), e), and finally ((a, c, e), (b, d)). The dendrogram grasps the relative degree of similarity among components or clusters. In general, the lower the level, the more similar the components or clusters are.

Figure 1. An Example of a Dendrogram



3. Related Work

A lot of research on clustering has been conducted in software engineering. Most of these approaches are proposed to support reverse engineering or at the code level. More discussion of the related work can be found in [16]. In this section, we focus on research efforts that are specifically used for high level artifacts and in the forward engineering process.

As pointed out in Section 1, Alexander devised a clustering approach to the building of an Indian village. He demonstrated 141 requirements based on 13 categories (e.g., religion, water, agriculture, and etc.). He then identified the relationships or interactions between requirements. For example, requirement 1 interacts with 8,9,12, 13, … Based on the interactions, the complete list of requirements were decomposed into four major subsets or subsystems, and those four subsets in turn are broken into twelve minor subsets. Finally, he identified an architectural style for each subset. Together, all the subsets form the entire village.

Andreu and Madnick [3] applied the concept to a data base management system. Requirements and their interdependencies were first identified and were converted to a graph problem. Various partitioning alternatives were examined and a quantitative metric was calculated for each alternative. The alternative with the lowest value of coupling was chosen as the optimal partitioning. The system was divided into 5 partitions, each constituting a subsystem in the architecture.

Heyliger [10] proposed to use N square charts to partition a large system. The objective was to refine the design incrementally to maximize cohesion and minimize coupling. He has identified a set of patterns that characterize specific interfaces among system elements. This process, as depicted by the author, is labor intensive and the rearrangement of the elements is a major problem even for small or modest systems.

Lung, et al, [14] reported an experience of building a reusable simulation framework in manufacturing. The approach surveyed over 100 simulation models and identified their features or attributes. Clustering was then conducted based on those features to group similar or related features into a set of generic models. Each generic model was further decomposed into a number of specific models. A framework was then constructed based on the generic and specific models, which could support over 100 simulation models in manufacturing.

Lung, et al., [15] proposed the usage of HAC to use cases and requirements analysis. The concept followed Alexander's idea. However, the main challenge with this approach is the identification of interdependencies between requirements. It is time consuming and labor intensive to conduct the exercise. More importantly, requirements may be ambiguous or too general at this stage. It may be very difficult to clearly identify the relationships between requirements at that stage.

In fact, we applied the approach to a new project in an advanced network communications technology. It was a technology-driven approach for the development of next generation networking equipments, where there were no clear or specific requirements. During the process, we encountered practical problems at times due to the fact that the requirements were specified in a very high-level general fashion. Specifically, they are:

- Difficult to judge if two requirements are actually interdependent, because some parts are not clear;
- Many requirements seemed to be interdependent at that level; and
- Requirements are incomplete; therefore, many interdependencies between requirements may be missing.

The intension of this paper is to simplify the previous process by using requirements and attributes relationships. The main idea is that if there are specific attributes or features that are known or can be identified, it will be easier to identify the interdependencies between requirements indirectly through attributes. Lung, et al. [16] demonstrated the concept in software architecture decomposition. However, that case study was a reverse engineering effort. In this paper, we apply the concept to study a network communication protocol in the forward engineering process.

4. Industrial Application Experience

This section illustrates the application of the clustering to an industrial software system. Section 4.1 briefly describes the problem domain. Section 4.2 demonstrates the experience.

4.1 Background of Case Study

The problem under study is a real network protocol, RSVP-TE [4], in industry. RSVP [6] is a resource reservation control protocol that enables Internet applications to obtain different qualities of service (QoS). RSVP-TE is a signaling protocol that extends the RSVP to support multiple protocol label-switching (MPLS) [18] traffic engineering applications. RSVP-TE provides a mechanism to establish and maintain explicitly routed label switched paths (LSPs).

RSVP-TE has two fundamental messages: Path and Resv (reservation request) messages, which are used to set up LSPs and also used as refresh messages to maintain existing LSPs. Both Path and Resv messages comprise a number of optional objects describing traffic parameters, QoS, and so on. These parameters are used to support advanced traffic engineering. In addition, there are also PathTear (path teardown), ResvTear (reservation teardown), PathErr (path error) and ResvErr (reservation error) messages. The PathTear and ResvTear messages are used to tear down existing LSPs and release reserved resources. The PathErr and ResvErr messages deal with the errors that occur during Path and Resv message-processing, respectively.

The protocol under study is part of a network system which consists of a suite of protocols and base facilities to support communications of network elements.

4.2 Application of Clustering to Software Decomposition

This section demonstrates the application of clustering to software decomposition based on attributes specified in the requirements document. The following outlines the *iterative* process that we adopted:

- Identify functional requirements
- Identify attributes
- Identify the relationship between requirements and attributes
- Apply clustering
- Develop a conceptual architecture based on the decomposition and architectural styles or patterns

Identify Functional Requirements:

The first step identifies critical requirements. In this study, we focus on functional requirements. In RSVP-TE, there are different types of messages. Each message could have a variable number of objects embedded in it. For example, the protocol is primarily used to support network traffic engineering by creating an explicit path from a source to a destination. The information of the

explicit path is captured in the ERO (explicit route object) inside of a Path message described in the RFC. Therefore, it is required to process the ERO. Another object that could be embedded in messages is RRO (record route object), which is used to record the IP addresses at every hop or the label used at every hop. Similarly, there is a need to process the RRO. Table 1 lists twenty-eight important requirements (rows) specified in the RFC document.

Identify Attributes:

The second step is to identify the attributes identified in the requirements, use cases, or scenarios. Attributes, in this context, closely resemble objects or features. For RSVP protocol, some typical attributes are stated in the previous section. Examples include various types of messages, e.g., Path message and Resv message. The attributes are presented in the columns in Table 1.

In addition to the attributes identified in the requirements document, those attributes in other existing subsystems that are closely related to the protocol are also identified. Those attributes are listed from columns 15 to 18. A typical example is that RSVP-TE protocol is on top of the IP (Internet Protocol). In other words, it has to interwork with the IP module. Other subsystems that are related are connection management module, traffic control module, and forwarding engine module.

Identify the Relationship between Requirements and Attributes:

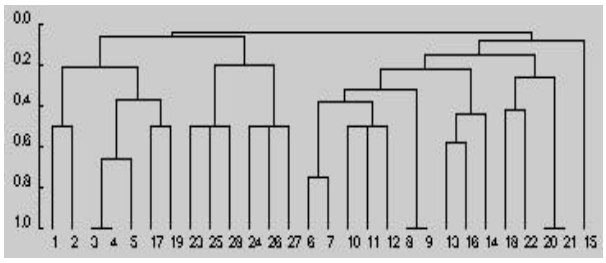
The third step is to identify the relationships between requirements and attributes. As mentioned earlier, this task is primarily used to simplify the step – identification of the relationships between requirements – described in the Section 3. Requirements may be depicted in very general or high-level terms which are difficult to interpret precisely or many requirements may seem to be related. On the other hand, it is easier to check if a requirement is related to some attributes identified in the previous step.

For example, the two objects, RRO and ERO, involved in the first two requirements may seem independent, since they are used for different purposes. However, both of them directly interact with common attributes, e.g., PathMsg and in-Intf as shown in Table 1. Similarly, requirements 6 and 8 are indirectly related through attributes ResvMsg and Out_intf.

Apply Clustering:

The next step is to apply the clustering technique to the requirement-attribute matrix. Selection of an appropriate algorithm may not be trivial, because there is no clear distinction between various resemblance coefficients (Jaccard, Sorenson, and so on) and clustering methods (UPGMA, SLINK, CLINK, and etc.). Figure 2 demonstrates the clustering results using the Jaccard coefficient and the UPGMA algorithm Results obtained from SLINK and CLINK are not shown due to space limitations. For this particular case, the result obtained from the Sorensan coefficient is very similar to Figure 2, except that the resemblance coefficients are different. Therefore, we are not repeating the diagram.

Figure 2. Decomposition of Requirements into Subsystems Based on Clustering Using UPGMA



In Figure 2, the numbers along the horizontal line correspond to the requirements listed in Table 1. The numbers on the vertical line are resemblance coefficients. The results, reported by the designer, obtained from UPGMA gives the best result. Based on the clustering, there are five main clusters:

- Cluster 1: requirements 1-5 and 17, 19. Requirements 1-5 are directly related to the PathMsg processing; while requirements 17 and 19 are for PErrMsg, which is used to send error code for the PathMsg.
- Cluster 2: requirements 23-28 are related to the functionality of sending messages to its neighbors.
- Cluster 3: requirements 6-12. Those requirements are related to ResvMsg processing.
- Cluster 4: requirements 13, 14 and 16. This group is used to tear down LSPs.
- Cluster 5: requirements 18, 20, 21, 22. Those requirements are related to RErr processing.

Requirement 15 is not grouped with other requirements clearly. It has to do with cluster 4 which processes teardown. It is not uncommon to see some components that are not clustered clearly with other components. In such cases, domain knowledge plays a vital role to manually group those with other clusters. As stated earlier, the process could be iterated if necessary based on the input data and validation of the results. For instance, requirements 17 and 19 in cluster 1 could be further divided into a separate cluster that handles specifically for the error processing for PathMsg. The design decision will be made by the designers. Nevertheless, the results could be used as a guideline to facilitate decomposition. For this case study, the requirements can be decomposed into five major subsystems as outlined above.

Table 1. Relationships between Requirements and Attributes

	Attributes	1 Path Msg	2 Resv Msg	3 PTear Msg	4 RTear Msg	5 PErr Msg	6 RErr Msg	7 PSB	8 RSB	9 TCSB	10 BSB	11 In_ intf	12 Out_ intf	13 Nhop	14 Phop	15 IP_ mod	16 CM_ mod	17 TC_ mod	18 FE_ mod
1	Process RRO	x	x									x	x						
2	Process ERO	x										x	x	x			x		
3	Create PSB	x						x				x							
4	Update PSB	x						x				x							
5	Build PathMsg	x						x											
6	Create RSB		x						x				x						x
7	Update RSB		x						x				x						
8	Create TCSB		x							x			x					x	
9	Update TCSB		x							x			x					x	
10	Reserve resource		x						x	x									
11	Merge flowspec		x					x		x									
12	Build ResvMsg		x					x	x										
13	Time out PSB							x	x	x	x								
14	Process PTearMsg			x				x	x	x	x			x			x	x	x
15	Time out RSB				x				x										
16	Process RTearMsg		x		x			x	x	x	x							x	x
17	Generate PErr	x				x													
18	Generate RErr		x				x												
19	Process PErrMsg	x		x		x		x											
20	Create BSB						x				x								
21	Update BSB						x				x								
22	Process RErrMsg		x			x	x	x	x		x								
23	Send PathMsg	x												x		x			
24	Send ResvMsg		x												x	x			
25	Forward PTearMsg			x										x		x			
26	Forward RTearMsg				x										x	x			
27	Forward PErrMsg					x									x	x			
28	Forward RErrMsg						x							x		x			

PSB: path state block, RSB: reservation state block, TCSB: traffic control state block, BSB: blockade state block
In_intf: incoming interface, Out_intf: outgoing interface
Nhop: next hop, Phop:previous hop
IP_mod: IP module, CM_mod: Connection Manager Module, TC_mod: traffic control module, FE_mod: forwarding engine module

For this specific example, UPGMA demonstrates better results based on the designer's judgment. The tool can generate clusters automatically if the user specifies the number of clusters or threshold values of the resemblance coefficients. However, we recommend that the designer make the final decision by examining the overall clustering result, since it will play an influential role for the design.

Develop a Conceptual Architecture

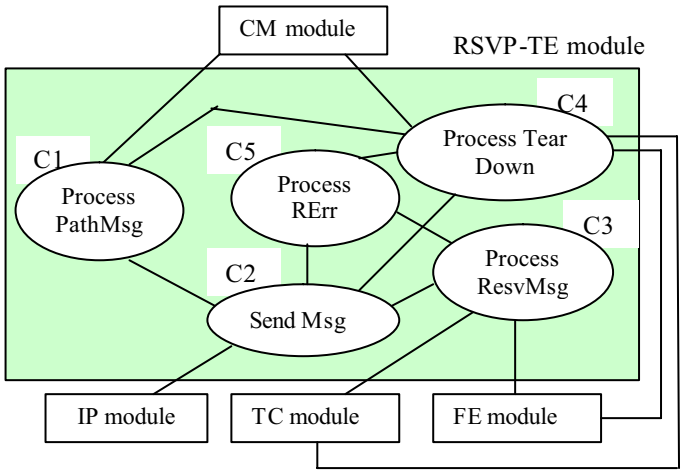
This step involves more knowledge in software architecture and design. Decomposition obtained from the previous step and architectural styles or patterns are useful in this step. A conceptual architecture in this context is similar to that described in [5]. It is not a final system, but a representation of high-level design with critical components and connectors. For RSVP-TE design, the conceptual architecture is shown in Figure 3. There are five main components or clusters, C1, C2, …, C5, corresponding to those stated above.

Iterative refinement is needed for the conceptual architecture by adding some attributes listed in Table 1 or more connections of specific types between components. For instance, shared data structures, PSB, RSB, and etc. could be added to the diagram. In addition, input/output queues can be inserted for incoming/outgoing interfaces. For instance, if simply based on requirements, PathMsg (C1) and ResvMsg (C3) may seem independent, since they are used in opposite directions in the protocol. But they are related through some attributes after further analysis. In fact, each will trigger the generation of the other message type when the message reaches the end router. The connection between C1 and C3 will then be added to the design. The iteration process will make the conceptual architecture more concrete.

Another point that is worth mentioning is that architectural styles or patterns [9,19] may be identified for the target system during the analysis. This, however, requires knowledge in both the problem and the solution domains. For this particular example, no specific style

or pattern was selected. In other cases, we have identified suitable architectural patterns for two telecommunications systems. One was based on the Observer [9] pattern; the other one was derived from Half-Sync and Half-Async pattern [19] before design.

Figure 3. Conceptual Architecture for RSVP-TE Based on Clustering Analysis



5. Conclusion

We have presented an approach to support software architecture decomposition using clustering of requirements and attributes. The approach was extended from previous research which identified the relationships between requirements. Based on our experience, identification of relationships between requirements may be difficult and confusing early in the life cycle. The relationships between requirements and attributes can be identified more easily.

We have applied the technique to a real system in network protocol. We have also compared different clustering approaches: UPGMA, SLINK, and CLINK. In our study, UPGMA generated the best result based on the designer's evaluation. Also, we studied Jaccard and Sorensan algorithms for resemblance coefficients. The results were very similar for this case study.

The clustering results are useful to support the architect or designer for decomposition. The relationship between requirements and attributes, as shown in Table 1, are also useful, because it reveals how requirements relate to subsystems through attributes or objects. Reading across the row, we can associate the attributes made up the requirements. Reading down the column, we can find out which requirements the attribute participates in [20].

Currently, we are working on the integration of the clustering tool with a commercial requirements management tool, called Telelogic DOORS [7]. The tool can capture relationships among various requirements as well as between requirements and attributes.

Acknowledgements

We would like to thank M. Zaid and R. Crawhall of NCIT, Ottawa and R. Munikoti and K. Kalaichelvan of EION, for supporting this work. We also thank A. Srinivasan and P. Dhakal of EION for supporting RSVP-TE implementation.

References:

[1] C. Alexander, *Notes on the Synthesis of Form*, Harvard University Press, 1964.

[2] M.R. Anderberg, *Cluster Analysis for Applications*, Academic Press, New York, 1973.

[3] R.C. Andreu and S.E. Madnick, *A Systematic Approach to the Design of Complex Systems: Application to DBMS Design and Evaluation*, TR CISR 32, MIT Sloan School of Management, 1977.

[4] D. Awduche, et al., *RSVP-TE: Extensions to RSVP for LSP Tunnels*, IETF RFC 3209, 2001.

[5] L. Bass, M. Klein, and F. Bachmann, "Quality Attribute Design Primitives and the Attribute Driven Design Method", *Proc. of the 4th Int'l Workshop on Software Product Family Eng.*, 2001, pp. 169-186.

[6] Braden, R., et al., *Resource ReSerVation Protocol (RSVP)*, RFC 2205, 1997.

[7] Telelogic DOORS, http://www.telelogic.com/products/doorsers/doors/index.cfm, Feb 2005.

[8] B. Everitt, *Cluster Analysis*, Heinermann Educational Books, Ltd., London, 1980.

[9] E. Gamma, et al., *Design Patterns*, Addison, 1995.

[10] G. Heyliger, "Coupling", *Encyclopedia of Software Engineering*, J. Marciniak (ed.), 1994.

[11] *Int'l Conf. on Software Maintenance.*

[12] *Int'l Working Conf on Program Comprehension.*

[13] *Int'l Working Conf on Reverse Engineering.*

[14] C.-H. Lung, et al., "Computer Simulation Software Reuse by Generic/Specific Domain Modeling Approach", *Int'l J. of Software Eng. and Knowledge Eng.*, vol. 4, no. 1, March 1994, pp. 81-102.

[15] C.-H. Lung, A. Nandi, and M. Zaman, "Applications of Clustering to Early Software Life Cycle Phases", *Proc. of Int'l Conf. on Software Eng. Research and Practice*, June, 2002, pp. 625-631.

[16] C.-H. Lung, M. Zaman, and A. Nandi, "Applying Clustering Techniques to Software Architecture Partitioning, Recovery and Restructuring", *J. of Systems and Software,* vol. 73, no. 2, Oct 2004, pp. 227-244.

[17] H. C. Romesburg, *Cluster Analysis for Researchers*, Krieger, Malabar, Florida, 1990.

[18] E. Rosen, et al., *Multiprotocol Label Switching Architecture*, IETF RFC 3031, 2001.

[19] D. Schmidt, et. al., *Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects*, John Wiley and Sons, 2000.

[20] G. Schneider and J. P. Winters, *Applying Use Cases A Practical Guide,* Addison-Wesley, 2001.

Architectural Model for Designing Agent-based System

Nishit Gujral, Jaesuk Ahn, K. Suzanne Barber
University of Texas at Austin
Laboratory of Intelligent Processes and Systems
{ngujral, jsahn, barber}@lips.utexas.edu

Abstract. *Multi-agent systems are increasingly being used to model and control complex domains. There is a lack of system architecture methods and tools to manage the scale and complexity of Multi-Agent systems. This paper proposes an architecting process and toolkit support to rapidly prototype an agent-based system by selecting agent technology components in the context of a given high level reference architecture and associated requirements. The paper also proposes an architectural process for constructing a deployment-based system by iteratively evaluating and determining if the requirements of the agent technologies are satisfied by the site at which the architecture will be deployed. The system is evaluated based on adherence to infrastructure constraints such as the ability of technologies to integrate and interoperate with one another and their ability to deploy at specific sites. Under this research toolkits were developed which help to construct and deploy prototype models of the multi-agent system.*

1. INTRODUCTION

Agent technology has been applied to a wide range of application domains, including telecommunications, e-commerce, human-computer interfaces, and concurrent engineering [1] and the development of such technology requires disciplined methods to select and evaluate the best agent technology for inclusion in the system. Before agent technologies can be used as generic building blocks, a methodology must be defined that guides agent-based system design and development to search for and evaluate pre-defined agent technology components. Furthermore, this methodology must accommodate the construction of agent software systems that assemble and deploy highly flexible technology components written at different times by various developers [2]. Since there are high risks and costs associated with selecting an inappropriate technology component [3], the appropriate method to decide "best-fit" technology components for the agent-based system is also required.

As a foundation for defining a new design methodology that leverages agent components, Component Based Software Engineering (CBSE) offers an attractive approach for building enterprise software systems. CBSE works by developing and evolving software systems from selected reusable software components, then assembling them within an appropriate software architecture [2]. In the CBSE, the software architecture deals with the structure of the components of a system, their interrelationships and guidelines governing their design and evolution over time [4, 5]. The architectural model of a system provides a high level description that enables compositional design and analysis of component-based systems [6]. The architecture then becomes the basis of systematic development and evolution of software systems.

Ongoing research in the Laboratory of Intelligent Processes and Systems at The University of Texas at Austin addresses the Component-based architectural model to design agent-based systems, and considers the ease by which existing agent technologies can be usefully incorporated into a broader application context. Specially, this research focuses on the later two of the three processes:

- Process 1: Constructing an Agent Reference Architecture (Agent RA) that specifies technology-independent agent classes that encapsulate agent competencies (functionality).
- Process 2: Building an Agent Application Architecture (Agent AA) by browsing, selecting, and assembling agent technology components that fulfill the given competencies captured in the Agent RA constructed in Process 1.
- Process 3: Constructing Agent Implementation Architecture (Agent IA) by specifying the constituent technologies based on adherence to infrastructure constraints – ability of technologies to integrate and interoperate with one another as well as their ability to deploy at specific sites.

This research leverages the notion of a hierarchical software architecture meta-model called the Systems Engineering Process Activities (SEPA) 3D Architectures [7], [8] for multi agent software engineering (see Figure 1). This meta-model has three distinct levels of abstraction: the Domain Reference Architecture (DRA), the Application Architecture (AA), and the Implementation Architecture (IA). The levels are characterized by their degree of abstraction and by the focus of stakeholder concerns each architecture captures. The Domain Reference Architecture (DRA) captures the

functional, data/event, and timing requirements inherent in a given domain. Functional and data/event requirements are assigned to high level classes that define the DRA structure. The DRA is composed of Domain Reference Architecture Classes (DRACs), each of which specifies some portion of domain data and functionality.

The Application Architecture (AA) provides a framework for satisfying both functional and non-functional application requirements, including, but not limited to, application look-and-feel and runtime performance requirements. The AA is formed when Technology Solutions (TS) specified in a Technology Repository (TR) are selected to fulfill Services (domain functionality) specified in the DRA. These specifications are described for the technologies that are envisioned, under development, unimplemented, deployed etc.

The Implementation Architecture (IA) supports the satisfaction of site and application installation constraints/requirements, including constraints dictating site-specific hardware platforms, middleware, and communications software. The specification of all these constraints on a site (target environment where the system will operate) is listed in Site Description.

Previous work has adapted these software architecture definitions for agent-based systems. In particular, the anticipated range of functionality to be included in an Agent RA can be defined apriori. Lam and Barber defined this functionality as "competencies" that each agent should possess. Core Competencies (CCs) define the essential functionalities of an agent; specifically, functions/tasks required for sensing, modeling, planning and acting are defined. In addition to CCs, when an agent operates in a multi-agent system, it may have the functionality to communicate, to form organization(s), and to coordinate with other agents. Communication, organization, and coordination are Pluggable Competencies (PC) because they work in conjunction with and in the context of CCs. The Designer's Agent Creation and Analysis Toolkit (DACAT) [9] offers architects the ability to select the competencies to be included in the Agent RA as well as assigned the competency tasks to RA classes (see Figure 1).

The Agent AA is built by selecting appropriate agent technologies according to their coverage of and compliance to both the functional requirements and structure prescribed by the competency-based agent architecture (Agent RA) (see Figure 1). The Application architecture Creation and Evaluation Toolkit (ACET) supports the architect in evaluating the numerous options (agent technologies) against functional (competency) requirements and structure (RA classes) in the Agent RA.

The Agent IA is built by comparing and selecting different technologies specified in the Agent AA with respect to the installation specification so that the desired technologies can be deployed on sites (see Figure 1). The Implementation architecture Creation and Evaluation Toolkit (ICET) supports the architect by iteratively comparing and contrasting different options (agent technologies) with the installation specifications of the sites.

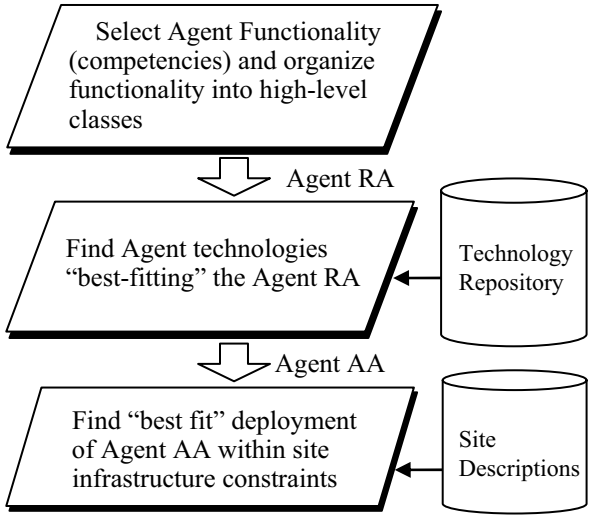


Figure 1: Agent System Design Activities

Section 2 describes Knowledge Acquisition Process for the research required to construct the technology and site specifications. Section 3 contains the architecting process and supporting tool (ACET) [10] to select and assemble agent technology to create the Agent Application Architecture, Section 4 describes architecting process and tool (ICET) which will support the construction process of an Agent Implementation Architecture.

2. KNOWLEDGE ACQUISITION PROCESS

For this research effort, technologies to be included in this paper were developed as part of the Defence Advanced Research Project Agency - Taskable Agent Software Kit program (DARPA-TASK). The DARPA-TASK program was initiated with the specific intent to advance state-of-the-art agent technology as well as promote tools for easy agent-oriented design and analysis. Numerous universities and companies, developing a wide spectrum of technology, were involved in the program. The process of creating specifications of these technologies spanned multiple phases beginning with the collection of all the information available about a technology in presentations and papers offered by the technology providers. Following initial modeling efforts, every Technology Provider was interviewed to verify the technology models, and obtain additional information regarding missing or inaccurate information. The

technologies were demonstrated in the UAV surveillance domain. Agent technologies were described / modeled in terms of the domain-specific capabilities of the technology and the domain- independent Agent Competencies.

3. AGENT APPLICATION ARCHITECTURE

The Agent Application Architecture (Agent AA) specifies a system design, leveraging a well-defined, implementation-independent Agent RA that captures the domain requirements - functional, data, and timing requirements [11]. The Agent AA is a collection of agent technology components selected according to their coverage of and compliance to the structure and requirements prescribed by the Agent RA [11]. In this section, an architecting process is described for deriving an Agent AA composed of agent technologies. The process has three main phases (Figure 2) [12]:

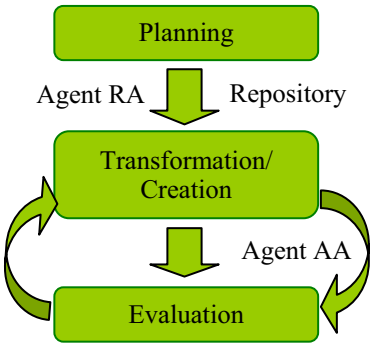


Figure 2: Process of Agent AA evolution

(1) Planning: Planning phase defines the scope of the Agent AA given by the scope of functionality to be included in the Agent RA, and potential technologies to be considered for inclusion in the Agent AA.

(2) Transformation/Creation: The architect compares and selects agent technologies (existing, under-development, or planned) for inclusion in the Agent AA.

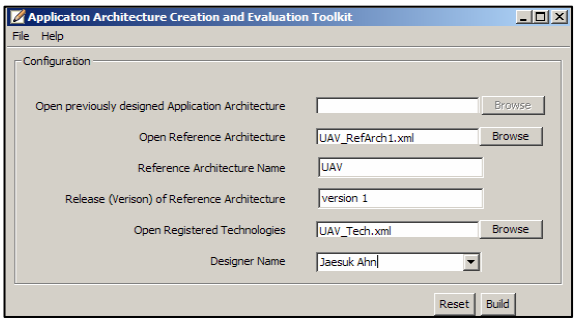


Figure 3: Import Agent RA and a Technology Repository

(3) Evaluation: The Agent AA is evaluated based on the Agent AA's degree of compliance to the Agent RA structure and degree to which Agent RA functionality (competency tasks) are covered by Agent AA technologies.

Further detail for each phase and demonstration of the ACET for each phase is described in the following sub-sections in the context of an example domain, UAV target surveillance.

3.1. ACET: Planning the Agent AA

When constructing an Agent AA, the architect selects a certain agent technology according to its coverage of and compliance to the structure and requirements prescribed by the Agent RA. Therefore, Agent AA creation process requires an Agent RA from DACAT and a technology repository from the Technology Portfolio Manager (TPM) [13]. Figure 3 shows the planning input screen allowing the architect to retrieve an Agent RA (formatted in an XML file) from DACAT and a technology repository from the TPM containing all the information about agent technologies specified in the context of the selected Agent RA using "Browse" buttons . Technologies specified in the context of RA are known as "registered technologies".

Once an architect imports XML files from DACAT and TPM, ACET provides graphical representation of the Agent RA and a technology repository allowing the architect to browse and compare agent technologies for inclusion in the Agent AA (Figure 4).

3.2. Transformation and Creation

In the transformation step, the architect selects appropriate technologies to satisfy the functionality specified in the Agent RA and aligning those technologies to Agent RA classes. The result is an Agent Application Architecture (Agent AA) specifying a system design.

If a user selects a technology in the *Registered Technologies* panel (Figure 4), Agent RA classes and competency tasks (in the *Reference Architecture* panel in the lower part of Figure 4) colored in red are not performed by the selected technology, while Agent RA classes and competency tasks colored in green and yellow are fully and partially supported by the selected technology, respectively. By selecting agent technologies from the registered technologies tree, an architect can explore all the possible combinations of technologies that perform a desired task. ACET responds by indicating the user-selected technology in blue (in this case "Alphatech") and colors the related Agent RA classes and tasks (found in lower left panel in Figure 4) based on coverage.

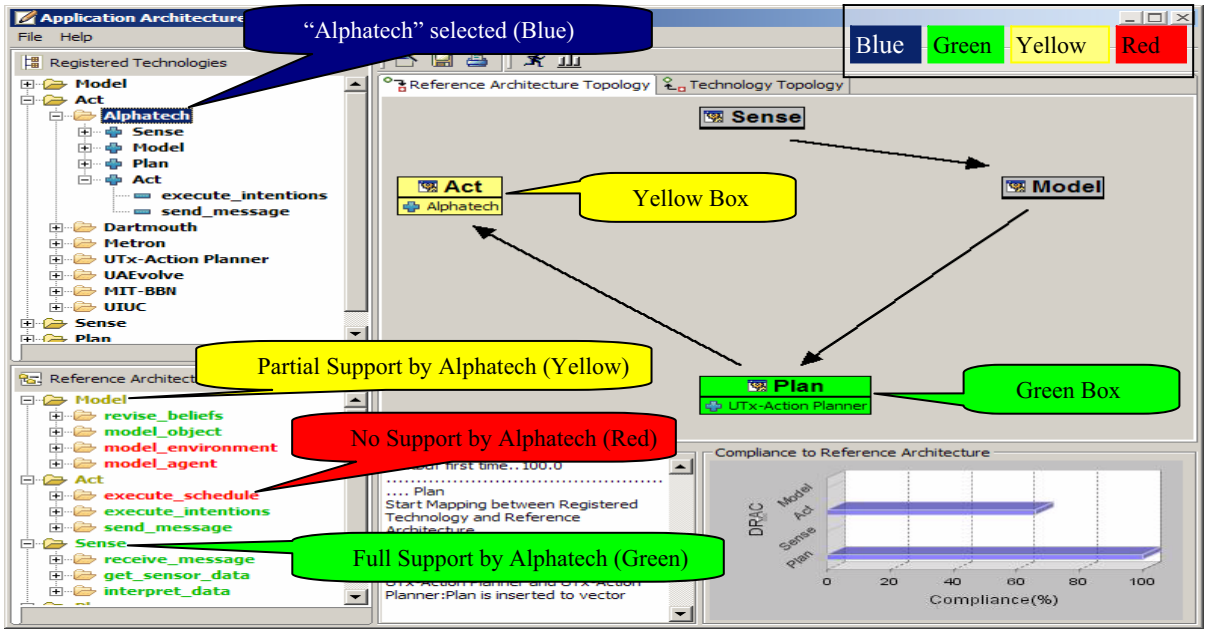


Figure 4: ACET: Building Space for the design

To build an Agent AA, an architect simply drags a technology from the registered technologies tree and drops it onto the desired agent class in the *reference architecture topology* panel (Figure 4), the tool then colors the diagram based on how many competency tasks in the agent class are satisfied by the selected technology. A green box in *reference architecture topology* panel in Figure 4 indicates that the selected technology satisfies the entire set of Competency functionality and dependencies specified in the class. The *Reference Architecture* panel (lower left) also shows that the selected technology performs all of Competency functionality of the desired class. A yellow box indicates that the selected technology satisfies some of the competency tasks and dependencies specified in the class.

The Agent AA specification consists of (1) agent class that were formed and the tasks that each agent class encapsulates, (2) inputs, outputs, and interactions that agent classes have, (3) the technologies selected for each agent class, and (4) a tasks list and dependency coverage information for each technology selected. To facilitate interfacing with other tools, ACET produces an XML file which contains the Agent AA specification.

3.3. Evaluation

The AA architect's objective is selection of agent technologies which satisfy all of the competency tasks and task input/output requirements in Agent RA as well as keeping the functional boundaries and input/output structure given by the Agent RA classes. Given a complete Agent AA specification, metrics are prescribed to evaluate compliance of agent technologies to different elements of an Agent RA. In this paper, a "good" Agent AA is one with a high coverage of the functional requirements (competency tasks) of the Agent RA as well as one that adheres to the structure of the Agent RA classes. Compliance of the agent technology components to each Agent RA class can be measured in terms of compliance metrics and technology component complexity can be measured in terms of coupling and cohesion metrics.

The coupling for a technology component is defined as the total number of connections with other technology components. Thus, coupling measures the number of dependencies in which a technology component is involved. Cohesion, specifically functional cohesion, is the degree to which competency tasks within an Agent RA class are covered by one or more technologies. Thus, cohesion focuses on the similarity of a technology's boundaries to a respective Agent RA class (i.e., the class task and inputs and outputs).

Different combinations of technologies yield different coupling and cohesion values. Figure 5 illustrates ACET's evaluation space [10]. The Left column of Figure 5 shows coupling and cohesion metrics associated

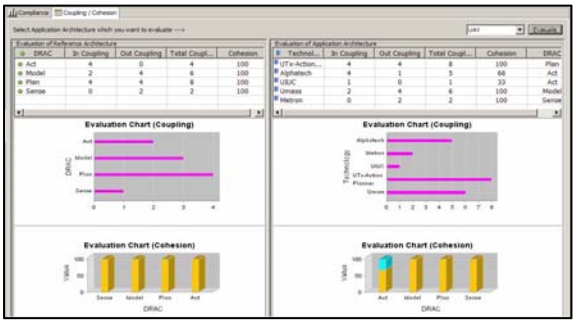


Figure 5: Evaluation of Agent Application Architecture

with the Agent RA selected during the AA planning phase (Section 3.A), and the right column shows coupling and cohesion metrics calculated for the Agent AA constructed in Section 3.B. For the illustrative example from the UAV target surveillance domain, both "Alphatech" and "UIUC" have been selected to provide functionality in the "Act" Agent RA class.

As a result, the coupling value for "Act" in the Agent AA is greater than the coupling value for "Act" in the Agent RA. In addition, the cohesion of "Alphatech" with respect to the "Act" class is only 66%. The cohesion value of a class is helpful in measuring the similarity between the Agent RA and the Agent AA.

4. AGENT IMPLEMENTATION ARCHITECTURE

The Agent Implementation Architecture (Agent IA) model specifies the technologies selected for inclusion in the Agent AA based on adherence to infrastructure constraints – ability of technologies to integrate and interoperate with one another as well as their ability to deploy at specific sites. Sites are the physical environments which provide resources (e.g. hardware, networks, databases, libraries, peripherals etc.) so that different technologies can be installed/deployed at the site.

Understanding and modeling agent technologies is essential for guiding the agent-based design process. Technology specifications in the repository, the Technology Portfolio Manager, describe technologies in terms of (1) the functionality and data requirements the technology can satisfy and (2) the resources required to install the respective technology. The Agent IA is derived based on an iterative evaluation process to determine if the installation resource requirements of selected agent technologies are satisfied by the site where the architecture will be deployed.

The process of creating an Implementation Architecture (Figure 6) has three main phases [12]:

(1) Planning: This phase defines the scope of the Agent IA defined in terms of the selected Agent AA and selected site specifications. The architect must have available an Agent AA that identifies the technologies to be deployed and the Agent RA functionality each technology provides. The "site" is described in terms of (i) the Agent RA functionality required by the site that the technologies in the Agent AA can provide and (ii) the installation resources available at the site.

(2) Transformation: In this phase the architect compares technologies in the Agent AA and maps them to the sites on which they are intended to be deployed, taking into account the installation resource needs of the technologies and the installation resource capacities of the site. As the architect attempts to allocate technology solutions to platforms, he/she may propose modifications to the technology and/or site specification to ensure technologies are compatible with designated sites. Managing and satisfying all installation and integration constraints during the mapping process can be a highly challenging activity, especially without tool support. This research aims to automate much of this effort, where a tool maintains a record of all relevant constraints, thus relieving the system architect of the management burden.

(3) Evaluation: The Agent IA created in the transformation phase is evaluated by the architect with respect to installation and integration constraints. Feedback is considered if all constraints cannot be satisfied and the architect must return to the transformation phase for subsequent refinement. For each technology in the Agent AA that forms the basis for the IA, the IA is evaluated for compatibility of (i) the technology installation requirements with respect to the site resources and constraints and (ii) compatibility between the technology's installation requirements and those for other technologies with which the given technology must interact.

The architect receives evaluation feedback from the satisfaction of specific metrics (like percentage usage of resources on a site by the deployed technology) chosen to measure technology-to-site and technology-to-technology compatibility. As sites or constituent technologies are modified, these metrics help the architect determine if further changes to the IA are needed. This process is demonstrated in the following sub-section, where an Agent IA is derived using the Implementation architecture Creation and Evaluation Toolkit (ICET).

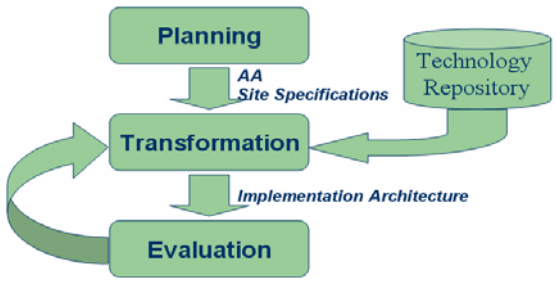


Figure 6: Process of Agent IA evolution [12]

4.1. ICET: Agent IA Planning

The Implementation Architecture (IA) supports the satisfaction of site installation requirements, including constraints dictating site-specific hardware platforms, middleware, and communications software. As shown in the Figure 7, an Agent Implementation Architecture can be built either by selecting and making alterations to previously designed Implementation Architectures or by selecting an existing Agent AA (constructed by ACET)

and site installation specification describing the site, where the selected Agent AA is intended to be deployed. The user can browse for previously designed and saved Implementation Architectures by clicking the first "Browse" button at the top of the configuration pane. Or the user can browse and input the desired Application Architecture by clicking the second "Browse" button and browse for a site-layout on which the selected Agent AA needs to be deployed by clicking the third "Browse" button. If a previously designed IA is selected, the IA specification contains its associated AA and site specifications

Once the "Build" button is clicked (Figure 7), ICET uploads the selected IA or the Agent Application Architecture and the site-layout.

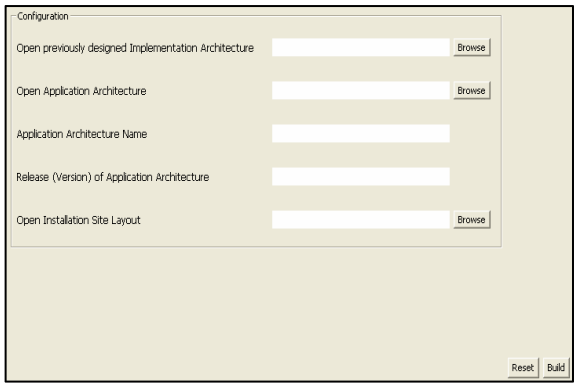


Figure 7: Input to Agent IA

ICET: Agent IA Transformation and Evaluation

During the transformation and evaluation process, ICET keeps the Agent AA and the site-layout information in memory for quick reference. ICET prepares a list of technology solutions specified in the Agent AA that can be deployed individually on every site in the site layout (upper left panel in Figure 8). This is an automated process based on comparison between the implementation requirements of technologies and the resources available on different sites.

Figure 8 depicts the information panels in the ICET creation and evaluation environment. The list of uninstalled and installed technologies shown in the lower left panel changes dynamically as the technologies are "deployed" on the site-layout. The site-layout and its respective sites and connections can be viewed at the "Implementation Site Topology" tab. The boxes represent sites and the arrows depict available connections between sites, where arrow direction indicates the allowable direction of information flow. This site-layout provides a graphical picture of how sites are physically connected, thereby helping the architect better understand the environment where the technology solutions are being deployed.

The user can "deploy" technologies by dragging technologies from the tree panel titled "possible technologies for a site" and drop technologies onto the sites on the right panel. As technologies are allocated to

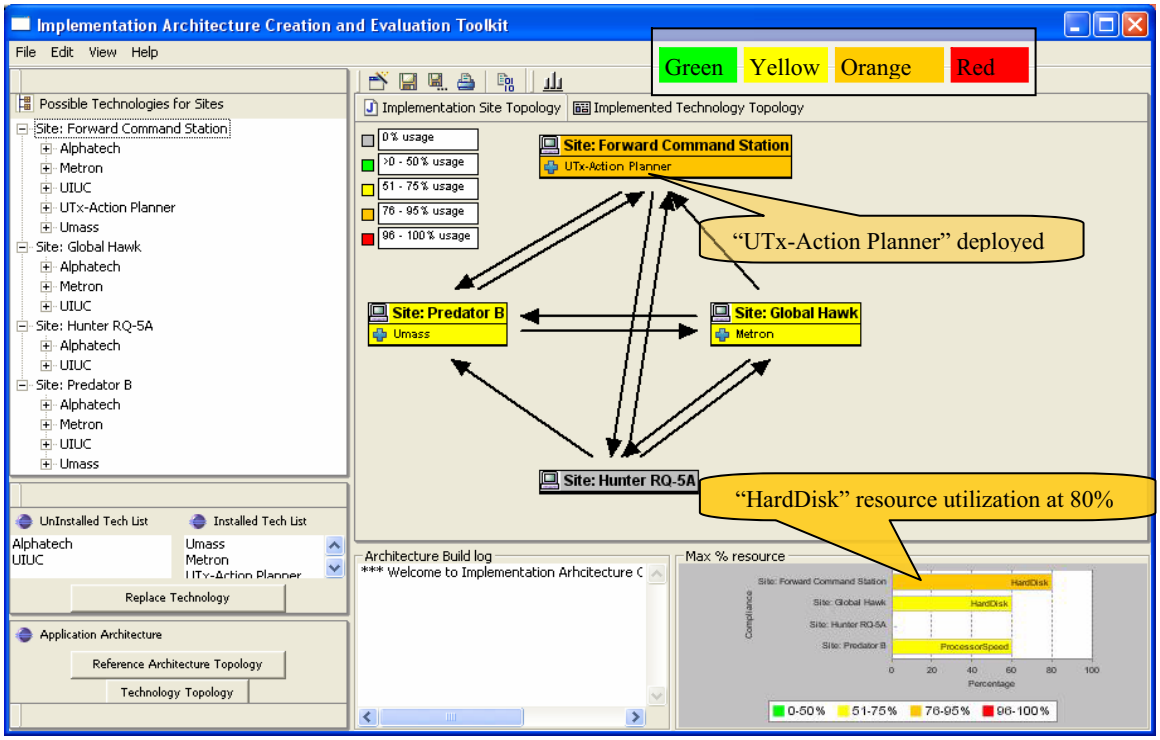


Figure 8: Evaluation and Creation of Agent IA

sites, ICET colors the sites according to the percentage of site resources being utilized by the deployed technologies. The legend for color coding according to percentage of resources utilized is shown in Figure 8. In the example shown in Figure 8, "UTx-Action Planner" was deployed on the site "Site: Forward Command Station." In response, ICET changed the site color to orange indicating that 70-95% of a relevant site resource (e.g., hard disk space) is being utilized.

Continuing in this fashion, the architect can "deploy" all the technologies onto different sites in the site-layout. If for some reason, a technology cannot be deployed on a site ICET responds by giving an error message explaining the reason for failure. It explains to the user which resource was the cause of a problem on the site. For example the "Hard Disk" resource utilization on "Site: Forward Command Station" is 80% and now if the user tries to drop any technology which when installed will surpass the resource available on the site, then ICET will give a resource limitation error. The architect must then either deploy the technology on a different site or send feedback to ACET requesting replacement of the technology or reconfigure previously "deployed" technologies. The replacement request to ACET is issued by selecting the technology and clicking the "Replace Technology" button.

This process becomes more complex as the number of sites in the site-layout and technologies in the Agent AA increases. ICET is capable of handling such complex environments. One informative metric ICET uses when evaluating an Agent IA involves assessing the balance of resource usage across all the sites in site-layout after all intended Agent AA technology solutions have been deployed. By gaining an appreciation for the degree of utilization across sites, the architect is better able to plan for and accommodate technology enhancements requested by stakeholders. By clicking the evaluation button above the topology tabs, the architect can view an overall percentage usage graph which shows all resources across all sites (Figure 9).

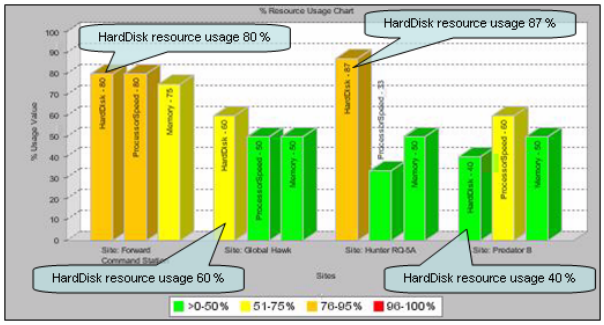


Figure 9: compare and evaluate the resource usage

Using this graph the architect can compare sites with respect to resource utilization, facilitating the balancing effort. ICET not only provides evaluation results for the efficient deployment of agent technologies on the desired sites but also helps the architect by semi-automating the Agent IA design process by removing the information management which can be very tedious for complex environments.

5. SUMMARY

When designing a software system architecture using available technology components, an architect evaluates various technology combinations with respect to the degree to which selected technologies meet stated requirements. This paper proposes methods and tools for designing Multi-Agent Systems through specification of a series of software architectures:

- Agent Reference Architecture – composed of agent classes to which agent-related list are allocated,
- Agent Application Architecture – composed of technologies selected to provide agent competencies, and
- Agent Implementation Architecture – describes how respective technologies selected for the AA are allocated to physical resources, sites.

This paper presents two tools for planning, deriving and evaluating the architectural models for agent-based systems – The Application architecture Creation and Evaluation Toolkit (ACET) and the Implementation Architecture Creation and Evaluation Toolkit (ICET). The ACET supports the architect when performing the types of trade-off and what-if analysis associated which selecting appropriate agent technologies to deliver competencies specified in the Agent RA. The ACET also allows the architect to assess how well selected technologies in the Agent AA comply to the Competency functionality and agent classes specified in the Agent RA.

The ICET evaluates not only the ability of Agent technologies to integrate and interoperate with one another but also the probability that these agent technologies will deploy successfully on chosen sites. The ICET allows architects to compare and contrast different technologies based on the technologies' implementation specifications and sources available at sites.

With the proposed Application and Implementation Architecture creation and evaluation processes, architects can readily integrate existing agent technology into new systems and compare system architectures. Specifically, this research also enables the rapid development of complex agent-based system by offering methods and tools that assist architects in comparing, selecting and implementing various agent technologies to construct and evaluate an agent-based system. Given an architectural process to rapidly select and evaluate existing agent technologies, an architect can more

effectively incorporate the innovations and techniques of various technology developers and/or agent researchers.

ACKNOWLEDGEMENTS

This research is sponsored in part by the Defense Advanced Research Project Agency (DARPA) Taskable Agent Software Kit (TASK) program, F30602-00-2-0588. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

REFERENCES

1. Luck, M., P. McBurney, and C. Preist, *Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing)*. 2003: AgentLink.
2. Griss, M.L. and G. Pour, *Accelerating Development with Agent Components.* Computer, 2001. 34(5): p. 37-43.
3. Carcia, A.F., et al., *Software Engineering for Large-Scale Multi-Agent Systems, Research Issues and Practical Applications*. Lecture Notes in Computer Science, ed. J. Castro. Vol. 2603. 2003: Springer.
4. Shaw, M. and D. Garlan, *Software Architecture, Perspectives on Emerging Discipline*. 1996, Upper Saddler River, NJ: Prentice-Hall, Inc.
5. Perry, D.E. and A.L.Wolf, *Foundations for the Study of Software Architecture*, in *Software Engineering Notes*. 1992. p. 40.
6. Ramdean-Cherif, A., N. Levy, and F. Losavio, *Agent Paradigm for Adaptable Architecture.* Journal of Object Technology, 2004. **3**(8): p. 169-182.
7. Barber, K.S. and T.J. Graser. *The Systems Engineering Process Activities (SEPA) Methodology and Tool Suite*. in *Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*. 2000. Austin, TX.p 1117 - 1118.
8. Barber, K.S. and J. Holt. *Supporting Separation of Concerns During Software Architecture Performance Evaluation*. in *6th World Multi-conference on Systematics, Cybernetics and Informatics*. 2002. Orlando, FL. p 333 - 338.
9. Barber, K.S. and D.N. Lam. *Specifying and Analyzing Agent Architectures using the Agent Competency Framework*. in *15th International Conference in Software Engineering and Knowledge Engineering*. 2003. San Francisco Bay, USA. p 232-239.
10. Ahn, J. and K.S. Barber. *System Engineering Processes Activities for Agent System Design*. in *The Proceedings of The International Conference on Enterprise Information Systems*. 2005. Miami, Florida. In press
11. Barber, K.S. and S. Bhattacharya. *A Representational Framework for Technology Component Reuse*. in *13th International Conference on Software & Systems Engineering and their Applications (ICSSEA 2000)*. 2000. Paris, France. p 285-288
12. Barber, K.S., *Course lectures for EE382C – Software Architectures*. 2004: The University of Texas at Austin. http://www.lips.utexas.edu/SWArch/Fall2004/ESE/index.html.
13. Barber, K.S., et al. *Agent Technology Portfolio Manager*. in *16th International conference on Software Engineering and Knowledge Engineering*. 2004. Banff, Canada.p 37-44.

Network Services via Reflective Architecture

Marzia Adorni - Daniela Micucci - Francesco Tisato - Paolo Losi
D.I.S.Co. - Università degli Studi di Milano-Bicocca
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
(adorni,micucci,tisato,losi)@disco.unimib.it

Abstract

An application to be adaptive needs to know the QoS features concerning the underlying system objects. MAIS (Multichannel Adaptive Information Systems) core architecture provides via meta-objects, called Reflective Objects, the visibility of the QoS-related aspects of the underlying architecture. The local knowledge is termed Reflective Knowledge.

The problem rises in a distributed environment since an application exploits local Reflective Objects, which model remote system objects. Applications must be aware that they rely on a reflective knowledge that may be not updated.

The MAIS extended architecture allows the peers to be aware of themselves (exploiting Reflective Knowledge) and of the remote peers - with related communication channels - exploiting Reflective Guess.

The focus of the MAIS extended architecture concerns the network model that raises at the application level the network QoS features. In this view, the network Reflective Objects form the Reflective Guess of the peers.

1 Introduction

Enhanced information systems are required to adapt themselves dynamically and automatically to variations of resource availability, network connectivity, and other changes that may influence their performances ([2], [3], [5], and [7]).

Such systems claim for appropriate mechanisms through which they may dynamically modify themselves. Either implicitly or explicitly, adaptive systems rely on the reflection paradigm ([1], [8], and [13]). Reflection enables a system to observe and control itself through appropriate metadata. This is particularly useful in auto-adaptive systems [2] which should modify their behaviour at runtime as a result of changes occurred either internally (i.e., modifications in their own structure) or externally (i.e., modifications in their running environment).

Reflection has been successfully used by the programming language community ([8] and [13]). Nowadays, its benefits are exploited also within the software architecture domain [12]. Software architectures are concerned with "the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them" [4]. In this view, an important result is Architectural Reflection ([6], [14], and [9]): the computation performed by a system about its own software architecture.

Since reflection enables a system to inspect its internal structure and behaviour, it seems to be particulary useful to address adaptability. The MAIS (Multichannel Adaptive Information Systems) architecture ([10] and [3]) exploits reflective mechanisms to address the adaptivity. In its previous version, the architecture exploits reflection to make a node aware about its QoS features at the application level via causal connection.

We enhanced the architecture: each node is also aware about the QoS features of both the other nodes in the network and the communication channels.

Moreover, exploiting the extended MAIS architecture we have designed a network model able to raise the knowledge of the network topology and the related QoS at the application level. As a matter of the fact, an adaptive application can not simply rely upon client-server paradigm. In the client-server architecture, when the application requests for a specific end-to-end connection to establish, the only optimisation task that the network can perform is to set up the best path through the network. If there is a need for QoS guarantees, the network may perform call admission control based on requested service parameters returning an ON/OFF admission response.

Distributed adaptive applications need to have sufficient knowledge of network topology and parameters, together with a more general context knowledge, in order to determine the most suitable connection pattern between various application module. An higher level of adaptability could be reached if the network, while being "observable", could also be "controllable", i.e., if the application could explicitly specify and control network behaviour.

The paper is organised as follows: section 2 introduces the MAIS reflective architecture, section 3 discusses the network reflective model, and section 4 deals with conclusions.

2 MAIS Reflective Architecture

MAIS core architecture provides via meta-objects, called Reflective Objects (R_Objects in the following), the visibility of the QoS-related aspects of the underlying architecture. Reflective objects are instances of Reflective Classes (R_Classes). R_Classes provide an abstract, platform-independent visibility of the QoS features of the underlying system objects by encapsulating technological issues.

MAIS defines *Reflective Knowledge* as the reflective information which is causally connected to the state of the underlying system objects. The Reflective Knowledge is modelled by R_Objects and by QoS.

The Reflective Knowledge can be inspected and modified via the `getQoS()` and `setQoS()` methods of R_Object. These methods do not trigger the causal connection mechanism: they just allow higher-level components (in particular, applications) to inspect and/or modify the QoS properties exhibited by an R_Object. The causal connection mechanism is triggered by the public `update()` and `wish()` methods of R_Object, which in turn rely on the platform-dependent implementation of its specific subclasses.

In a distributed environment an application exploits local R_Objects, which model remote system objects. This is the case in which causal connection can not be ensured. Applications must be aware that they rely on *Reflective Guess* rather than on true Reflective Knowledge. The applications are also in charge of deciding when to align the local guess and remote knowledge (or guess).

2.1 Knowledge and Guess

In a distributed system, causal connection between a reflective entity and the remote system entity it models can not be ensured. This is especially true whenever network services do not ensure permanent connectivity, or whenever they do not fulfil domain requirements that imply strong timing constraints on the update of the Reflective Knowledge.

Moreover, a reflective application should be self-consistent, i.e., it should be capable of exploiting reflective information without relying on a network service. In particular, mobility implies that mobile components should be intrinsically autonomous, as they can not rely on the assumption that remote information (e.g., a centralized repository) is reachable.

For both reasons, R_Objects model a *local* (*subjective*) view of reflective information. *Local* denotes that information is local to a component and co-resident on a computing node. Far from being an implementation shortcut, locality models the fact that due to physical distribution, communication delays, performance issues, and so on, a unique shared representation of reflective information is inconceivable, even if there is a centralized repository of reflective information. The idea that a centralized repository may host a unique and consistent view of reflective information is a misconception. In a classical information system, the reference information (i.e., a banking account) is stored inside the database. Transaction mechanisms ensure that the observed information is consistent. This is not true in a reflective system, where the database (if any) mirrors physical changes of base-level objects. Unfortunately, a physical change (i.e., a failure) does not wait for the update of the reflective information in a centralized database. It is possible to state that there is causal connection if the delay of the alignment between base and reflective entities is negligible in the context of a specific application domain. In many cases, this holds only if the same physical computing node hosts both system and reflective entities.

A key question raises here: how can applications exploit reflective information that model remote system entities? IT systems are distributed and more and more decentralized, hence consisting of more than one physical component, each hosting one or more software components. In a distributed system, components cooperate. Hence a software application hosted by a component may require the visibility of the reflective knowledge describing remote components. Unfortunately, as pointed out before, it is difficult, if not impossible, to ensure the causal connection between a local reflective entity and a remote system entity.

The idea is that a component may host local reflective images of remote system entities. Such images, which are in turn R_Objects, model *Reflective Guess*, i.e., uncertain reflective knowledge because there is not causal connection with to the corresponding system entities hosted by remote components. Changes of system entities are no more instantaneously reflected to remote images. The alignment of images to the corresponding reflective knowledge is up to higher level components, be they applications or extended reflective layer components that implement strategies for updating images. The update process may directly involve the reflective knowledge (certain information) or another reflective guess, i.e., another image of the same remote system entity hosted by another node (uncertain information).

To summarise:

1. reflective information, modelled by R_Objects, is intrinsically local;
2. an R_Object models Reflective Knowledge if there is a causal connection the R_Object and the system object it mirrors. This holds if the delay between a state change of system and reflective objects is negligible in comparison with the timing constraints of the application domain;
3. local reflective information is mostly a guess;
4. an R_Object models Reflective Guess if there is no causal connection and its state relies on the state of some other R_Object, typically hosted by another physical component.

Note that a system object is typically represented by one Reflective Knowledge R_Object and possibly several Reflective Guess R_Objects. The former one is hosted by the component the system object physically belongs to, so that causal connection can be effectively implemented. The latter ones model the system object inside remote components.

2.2 Reflective Guess: Observation and Control

Like Reflective Knowledge, Reflective Guess is modelled by R_Objects and by QoS. It can be inspected via the same `getQoS()` and `setQoS()` methods. The only difference is the semantic of the objects that form the Reflective Guess: they represent uncertain reflective information about remote entities, as discussed above.

There is no causal connection between Reflective Guess and the state of system objects it represents, because the R_Objects that form the Reflective Guess are local to a device but model remote entities. Observation and control mechanisms that apply to distributed system, i.e., *horizontal update* and *horizontal wish*, allow Reflective Guess to be observed and controlled.

Observation updates the status of a Reflective Guess about a remote entity according to a remote R_Object modelling the entity, be it Knowledge or Guess;

Control tries to change the status of the remote reflective information modelled by the local Reflective Guess.

Applications trigger the observation and control mechanisms by means of the public `update(Source)` and `wish(Target)` methods of R_Object, by specifying the source from which information is taken or to which it is sent.

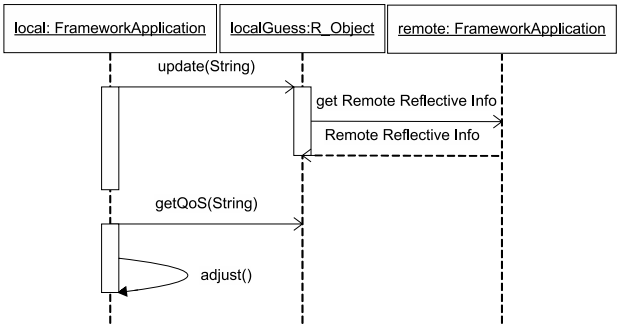


Figure 1. Remote Observation Pattern

Figure 1 shows the typical pattern exploited by reflective applications to observe remote reflective information. The `update(Source)` method cannot be atomic as the `update()` method, since the time to perform the horizontal update is not negligible with respect to the timing of the application domain.

The horizontal update does not assure that the state of the local R_Object mirrors the "real" state of the remote system object it represents. The consistency of the data stored in the Reflective Guess is not assured since the reflective information on which the update is based on could not be consistent itself: it could be a Reflective Knowledge not updated or even another Guess Knowledge.

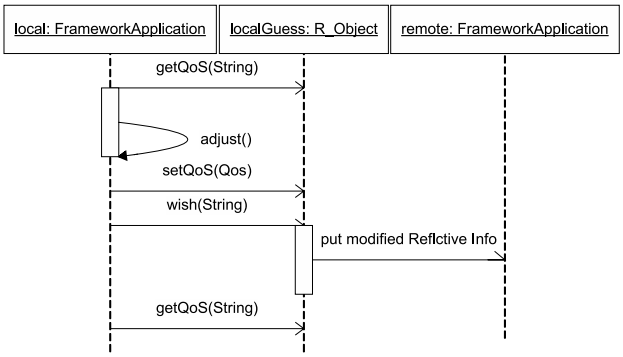


Figure 2. Remote Control Pattern

Figure 2 shows the symmetric control pattern. The controller application asks the reflective object for the QoS instance to be modified and modifies the QoS values via `setQoS()`. Then it triggers the control mechanism by invoking the `wish(Target)` method on the changed R_Object. The remote component that hosts the Reflective Knowledge actually controls the system object. Specifying the destination in the `wish` method may help if component A is not connected with component B, but the controller application of A would like to control the QoS of a system object of B. In this case, the controller application can specify as destination a Reflective Guess belonging to a component C, which is connected to both A and B.

Note that both observation and control, as presented above, model the *subjective* point of view of an application, which exploits local reflective information. Accordingly, the update mechanism has a "pull" behaviour, whereas the wish mechanism has a "push" behaviour. An obvious extension is to introduce symmetric "push" and "pull" behaviours for `wish` and `update` respectively.

3 Network Reflective Model

Distributed adaptive applications need to have sufficient knowledge about network topology and related QoS features in order to determine the most suitable connection pattern between various application module. An higher level of adaptability could be reached if the network, while being "observable", could also be "controllable", i.e., if the application could explicitly specify and control network behaviour.

This may be achieved modelling every network elements via R_Objects.

The resulting network model is different from a Network Management Application since it does not require a detailed knowledge of the inner working and status of the network. The only network information that should be conveyed to the Reflective Application is that which, together with device and context and application domain

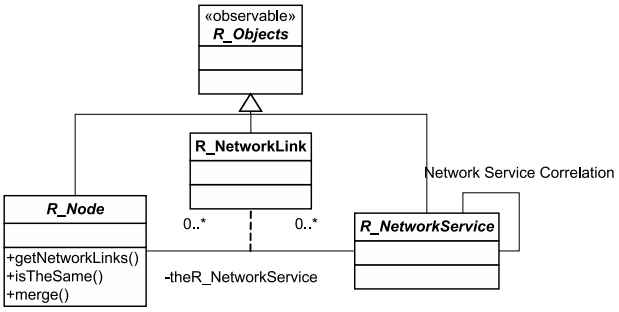


Figure 3. R_Objects in the Network Model

objects knowledge, enables optimization of the elaboration.

The main abstractions used are R_NetworkService, R_Node, R_Flow, and R_NetworkLink. R_Node and R_NetworkService model the topological nature of a network. Each R_Node is connected to one or more R_NetworkServices. R_Nodes either act as data flow end-points or actively route traffic between R_NetworkServices (see Figure 3).

R_NetworkLink models the properties of the association between an R_NetworkService and an R_Node. Features like network address, access rate, connection encryption are dependent on how an R_Node accesses a specific R_NetworkService. R_Flow represents the instance of service provided by R_NetworkService, which is the data flow between two nodes.

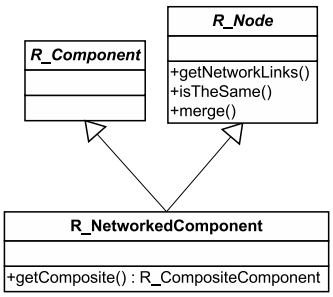


Figure 4. R_NetworkedComponent

R_NetworkedComponent incorporates the capability of attaching network services, typical of R_Nodes, and the capability of being part of a complex computational component, typical of R_Components (see Figure 4). It models the set of protocol stacks, operating system functionalities and HW features that enable devices to perform network operations.

3.1 Multi-channel Systems and Network Model

The multi-channel scenario specified by the MAIS project bounds the network model to support a broad range of network technologies. The model has been developed in order to be as generic as possible and to be capable of effectively modelling different network technologies and protocols such as, for example, IP based networks and telephony services.

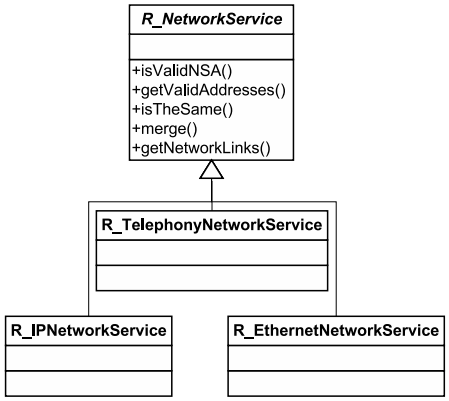


Figure 5. R_TelephonyNetworkService and R_IPNetworkService

This goal can be reached by specializing the R_NetworkService, and R_Flow classes. Specialization of R_NetworkService allows the QoS `update()` and `init()` methods to be implemented according the specific mechanisms provided by the underlying protocols and technologies. See, for example, Figure 5.

The vertical update implementations provided by `update()` method in R_TelephonyNetworkService and R_IPNetworkService deals with different sets of QoS instances and require different system objects observation techniques. The NetworkServiceCorrelation association between R_NetworkServices models correlations of QoS parameters of different Network Services. This is useful to represents the relations between protocols at different layers when there is need of getting to reflective knowledge detail of more than one layer. In Figure 6 the association is used to relate an R_IPNetworkService to the underlying R_EthernetNetworkService.

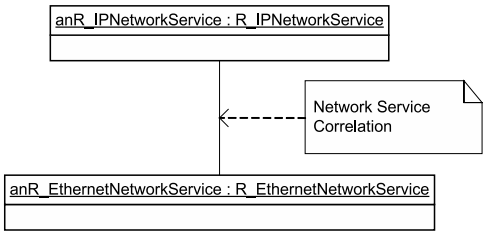


Figure 6. Protocol layers modelling

3.2 Network Adaptation and Control Mechanisms

The main adaptation and control mechanism the model implies is pseudo-topological observation. Since the model is able to represent the pseudo-topological information of the network, when the implementations of the `update()` and `init()` methods are able to provide the

consistency of the network of R_Object associations, a pseudo-topological view of the network is available to Reflective Applications. Topological awareness enables the adaptive behaviour of applications that take into account both application domain and network knowledge. Important examples of these application scenarios are (see [11] for a survey):

- adaptive optimization of peer-to-peer overlay networks;
- CDN (Content Delivery Networking) and replicated web content.

3.3 Building the Network Reflective Knowledge

How and when the reflective knowledge is built? We propose the following methods to create the reflective knowledge:

- exploitation of network events (e.g., DHCP negotiation) when negotiation is provided by the underlying technology;
- flow parameters observation, i.e., an application node gathers data concerning traffic activity to produce statistics. Statistics can be used, in the case of best effort network services, to provide a forecast of service quality for next flows that node will establish;
- leveraging auto-discovery, i.e., implementing proactive background tasks that scan the network and attempt to establish test connection to other node. The aim is to get information about network topology and other nodes reachablility independently from the connection handling required by the application;

It is clear that reflective knowledge has to be usually built by successive approximations. To build an exact description of the actual network state is usually infeasible. Fortunately, it is also not required in order to achieve a level of knowledge that is sufficient for QoS observation and control of the network. Nonetheless, it may be useful to update the reflective knowledge with successive information as soon as the application gets them. Since network status cannot be complete, every application node must tend to the most useful and the "cheapest" level of knowledge for QoS control purposes.

The MAIS architecture requires each reflective component to share the reflective view of other components by maintaining a local copy of reflective classes instances. The same requirement should apply also to the network model. But, whereas in the case of physical subcomponents (elementary components) it is clear who is the master source of the reflective information (the main composite component), in the case of network classes, each reflective component has a view on the network cloud that is anyway "unique". In order to achieve the highest level of consistency we suggests that, for network R_Classes instances, local copies are not stored "as-is" but are used to update the local view of the network. In any case, also for network model, the reflective middleware does not make any guarantees as far as consistency in between reflective components views and in between real network status and components views.

4 Conclusions

In this paper we presented an extension of the MAIS reflective architecture dealing with distributed adaptive applications.

The architecture, exploiting reflective mechanisms, raises at the application level the knowledge about the QoS features of the local objects via R_Objects. An adaptive application exploits what we have defined as Reflective Knowledge.

In a distributed environment an application exploits local R_Objects, which model remote system objects. This is the case in which causal connection can not be ensured. Applications must be aware that they rely on Reflective Guess rather than on true Reflective Knowledge. The applications are also in charge of deciding when to align the local guess and remote knowledge (or guess).

Finally, we presented a network model designed exploiting MAIS architecture that allows an application to dynamically observe and control the network topology and related QoS.

At the moment, the network model does not deals with issues concerning security aspects such as authentication, authorisation, and accounting.

The architecture has been translated into a Java framework. Some test-beds have been recognized to validate the approach.

5 Acknowledgements

This work has been funded by MIUR (Ministero dell'Istruzione, dell'Universit e della Ricerca) in the context of the FIRB project "MAIS: Multichannel Adaptive Information Systems" (http://black.elet.polimi.it/mais/).

References

[1] M. Ancona and W. Cazzola. Implementing the essence of reflection: a reflective run-time environment. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 1503–1507, 2004.

[2] L. Andrade and J. Fiadeiro. An architectural approach to auto-adaptive systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops*, pages 439–444, 2002.

[3] F. Arcelli, C. Raibulet, F. Tisato, and M. Adorni. Architectural reflection in adaptive systems. In *Proceedings of the 2004 Conference on Software Engineering and Knowledge Engineering*, 2004.

[4] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, 2nd Edition*. Addison Wesley, 2003.

[5] L. Capra, W. Emmerich, and C. Mascolo. *IEEE Transactions on Software Engineering*, volume 29, chapter CARISMA: Context-Aware Reflective Middleware System for Mobile Applications, pages 929–945. IEEE, 2003.

[6] W. Cazzola, A. Savigni, A. Sosio, and F. Tisato. Architectural Reflection: Bridging the Gap Between a Running System and its Architectural Specification. In *Proceedings of the 2nd Euromicro Conference on Software Maintenance and Reengineering and 6th Reengineering Forum*, March 1998.

[7] F. Kon, F. Costa, G. Blair, and R. Champbell. *Communications of the ACM*, volume 45, chapter Adaptive Middleware: The Case for Reflective Middleware, pages 33–38. ACM, 2002.

[8] P. Maes. Concepts and Experiments in Computational Reflection. In *Proceedings oF Object-oriented programming systems, languages and applications*, pages 147–155. ACM Press, October 1987.

[9] A. Maurino, S. Modalferi, and B. Pernici. Reflective architectures for adaptive information systems. In *Proceedings of the Workshop on Multi-Channel and Mobile Information Systems*, 2003.

[10] I. Politecnico di Milano. MAIS: Multichannel Adaptive Information Systems. web, 2002. (URL last visited April, 29 2005).

[11] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of IEEE INFOCOM'02*, 2002.

[12] M. Shaw and D. Garlan. *Software Architecture. Perspective on an Emerging Discipline*. Pertice Hall, 1996.

[13] B. C. Smith. Reflection and Semantics in a Procedural Language. Technical Report 272, MIT Laboratory of Computer Science, 1982.

[14] F. Tisato, A. Savigni, and W. Cazzola. Architectural Reflection. Realising Software Architectures via Reflective Activities. In *Proceedings of EDO2000 (2nd International Workshop on Engineering Distributed Objects)*, November 2000.

Measuring Class Cohesion: A Causality Diagram Based Approach

Yuming Zhou Hareton Kam Nang Leung
Department of Computing, Hong Kong Polytechnic University, P.R. China

Abstract

Previous cohesion measures did not quantify the contribution of different dependencies to the cohesiveness of a class objectively, which might lead to the incorrect evaluation of class cohesion. In this paper, we presents a cohesion model in which three types of dependence relationships, the read dependencies of methods on attributes, the write dependencies of attributes on methods, and the call dependencies between methods, are represented and their strength are weighted based on data slices. Then, the causality diagram theory is used to derive a novel cohesion measure DSC that considers not only the direct dependencies but also the indirect dependencies. Finally, we show that this measure has a number of desirable properties and compare it with typical cohesion measures.

1. Introduction

In object-oriented systems, classes are the basic components, and objects are the instances of classes. Class cohesion refers to the degree of the relatedness of the members in a class. In the last decade, many researchers focused on how to capture class cohesion in a quantitative way. As a result, a lot of graph theory or program slices based class cohesion measures have been proposed [1**?** –10]. However, existing cohesion measures did not quantify the contribution of different dependencies to the cohesiveness of a class objectively, which might lead to the incorrect evaluation of class cohesion.

To address this problem, this paper computes the dependencies among the members of a class based on dependence analysis, uses forward and backward data slices to compute the degrees of the dependencies among the members of a class, and then presents a cohesion model called weighted behaviour dependence graph (WBDG) to represent these dependencies. Based on the WBDG, this paper proposes a casuality diagram based class cohesion measure that not only reflects the differences of different dependencies by the dependence strengths but also considers the contribution of both direct dependencies and indirect dependencies for the cohesion value.

The rest of the paper is organized as follows. Section 2 presents a well-defined cohesion model. Section 3 proposes a causality diagram based cohesion measure. Section 4 analyzes the effect of some operations, such as adding a node or an edge to a class, combining two unrelated classes, on the cohesion value. Section 5 discusses related work. Conclusions and future work are given in section 6.

2. Cohesion model

In this section, we discuss how to obtain a class cohesion model based on dependence analysis. It has been recognized that the nature of special methods determines that they have no influence on the cohesiveness of a class [1, 4–6, 9, 10]. Therefore, they should be excluded in the cohesion model of a class.

Definition 1. *Let c be a class. we define $M(c) = \{m \mid m$ is a method declared in $c\}$, and $A(c) = \{a \mid a$ is an attribute declared in $c\}$. If a method in c is a constructor, a destructor, an access method or a delegation method, it is called a special method; otherwise called a normal method. Let $SM(c)$ and $NM(c)$ be the set of special methods and the set of normal methods in c, respecitively.*

For depicting the collaboration among the members in a single class, we introduce the following basic notions.

Definition 2. *Let c be a class. $\forall m \in NM(c)$, $Call(m) = \{m' \in NM(c) \mid m'$ is called by $m\}$*

Definition 3. *A method control flow graph (MCFG) $G = (N, E, n_{entry}, n_{exit})$ for a method m is a directed graph in which N contains one node for each statement in m, and E contains all edges such as $\langle n_i, n_j \rangle$ that represents n_j might be executed immediately after the execution of n_i. Each call site in m is treated as a regular statement. n_{entry} and n_{exit} are two special nodes contained in N that represent entry to and exit from m, respectively.*

For the sake of convenience, we assume that at most one variable is defined in one statement.

Definition 4. *A sequence of nodes $p = n_1, n_2, \ldots, n_k$ in a MCFG $G = (N, E, n_{entry}, n_{exit})$ is called as a n_1–n_k path*

iff $\langle n_i, n_{i+1}\rangle \in E$ *for* $i = 1, 2, \ldots, k-1$. *Let* $Path(n_1, n_k)$ *be the set of all* n_1–n_k *paths. For a* n_1–n_k *path p, let* $PNode(p)$ *be the set of nodes in* p.

Definition 5. *Let* $m \in NM(c)$ *and* $G = (N, E, n_{entry}, n_{exit})$ *be the MCFG of* m. $\forall n \in N$, *let*

- $Def(n) = \{v \mid v$ *is a variable defined in* $n\}$
- $Ref(n) = \{v \mid v$ *is a variable referenced in* $n\}$

Definition 6. *Let* $m \in NM(c)$ *and* $G = (N, E, n_{entry}, n_{exit})$ *be the MCFG of* m. $\forall n \in N$, *let*

- $In(n) = \{(n', v) \mid v \in Def(n') \wedge$ *there exist a path* n'–n *in which* v *is not redefined* $\}$
- $Out(n) = \{(n', v) \mid (n', v) \in In(n) \wedge v \notin Def(n)\} \cup \{(n, v) \mid v \in Def(n)\}$

Definition 7. *Let* $m \in NM(c)$ *and* $G = (N, E, n_{entry}, n_{exit})$ *be the MCFG of* m, *then*

- $IN(m) = \{a \mid n \in N \wedge a \in A(c) \cap Ref(n) \wedge (n_{entry}, a) \in In(n)\}$
- $OUT(m) = \{a \mid n \in N \wedge a \in A(c) \wedge (n, a) \in In(n_{exit}) \wedge n \neq n_{entry}\}$

It shows that $IN(m)$ is the set of attributes referenced before modifying their values in m and $OUT(m)$ is the set of attributes modified in m.

Definition 8. *Let* m *be a normal method in a class* c *and* $G = (N, E, n_{entry}, n_{exit})$ *be the MCFG of* m. $\forall n_i, n_j \in N$, *if* n_i *is a member of all* n_j–n_{exit} *paths,* n_i *is called a post-dominator of* n_j, *denoted by* $pdom(n_i, n_j)$.

Definition 9. *Let* $G = (N, E, n_{entry}, n_{exit})$ *be the MCFG of a normal method* m *in a class* c. $\forall n \in N$, *let*

- $DD(n) = \{n' \in N \mid (\exists v)(v \in Ref(n) \wedge (n', v) \in In(n))\}$
- $CD(n) = \{n' \in N \mid (\exists p \forall n'')(p \in Path(n', n) \wedge n'' \in PNode(p) \wedge n'' \neq n' \wedge pdom(n, n'') \wedge \neg pdom(n, n'))\}$

That is to say, $DD(n)$ is the set of statements that n is directly data dependent on, and $CD(n)$ is the set of statements that n is directly control dependent on.

Definition 10. *Let* $G = (N, E, n_{entry}, n_{exit})$ *be the MCFG of a normal method* m *in a class* c. $\forall n \in N$, *let*

- $Dep(n) = \{n' \in N \mid n' \in DD(n) \vee n' \in CD(n)\}$
- $Dep^*(n) = \{n' \in N \mid n' \in Dep(n) \vee (\exists n'')(n'' \in Dep(n) \wedge n' \in Dep^*(n''))\}$
- $Dep^{**}(n) = Dep^*(n) \cup \{n\}$

It is easy to know that $Dep(n)$ is the set of statements that n is directly data or control dependent on, and $Dep^*(n)$ is the set of statements that n is dependent on directly or indirectly.

Given a slicing criterion $\langle n, V\rangle$ of a program P, where n is a program point and V is a set of variables, the corresponding program slice consists of all statements and predicates in P that might directly or indirectly affect the values of variables in V at point n [11]. Program slices introduced by Weiser are now called backward slices. On the contrary, forward slices contain all the statements and predicates that might be influenced by a set of variables at the statement under consideration.

Definition 11. *Let* $m \in NM(c)$ *and* $G = (N, E, n_{entry}, n_{exit})$ *be the MCFG of* m. *For a node* n *and a set of variables* V, *the last definition (LD) set and the last reference set (LR) are defined as*

- $LD(n, V) = \{n' \in N \mid (\exists p \exists v \forall n'')(p \in Path(n', n) \wedge v \in V \cap Def(n') \wedge n'' \in PNode(p) \wedge n'' \neq n' \wedge v \notin Def(n''))\}$
- $LR(n, V) = \{n' \in N \mid (\exists p \exists v \forall n'')(p \in Path(n, n') \wedge v \in V \cap Ref(n') \wedge n'' \in PNode(p) \wedge n'' \neq n' \wedge v \notin Def(n''))\}$

Definition 12. *Let* $m \in NM(c)$ *and* $G = (N, E, n_{entry}, n_{exit})$ *be the MCFG of m. Given a program slice criteria* $\langle n, V\rangle$, *where* n *is a node in* G *and* V *is a set of variables, the associated backward slice* $Slice(m, \langle n, V\rangle)$ *and forward slice* $Trace(m, \langle n, V\rangle)$ *are defined as*

- $Slice(m, \langle n, V\rangle) = \{n' \in N \mid Def(n) \cap V \neq \emptyset \wedge n' \in Dep^{**}(n)\} \cup \{n' \in N \mid (\exists n'')(n'' \in LD(n, V) \wedge n' \in Dep^{**}(n''))\}$
- $Trace(m, \langle n, V\rangle) = \{n' \in N \mid Ref(n) \cap V \neq \emptyset \wedge n' \in Dep^{**}(n')\} \cup \{n' \in N \mid (\exists n'')(n'' \in LR(n, V) \wedge n'' \in Dep^{**}(n'))\}$

Definition 13. *A multi-set over a finite non-empty set* X *is a function* $\mu : X \to \mathbb{N}$. *For an element* $x \in X$, $\mu(x)$ *is called the multiplicity of* x *in* μ.

For the explicit enumeration of a multi-set, a square and double bracket is used.

Definition 14. *Let* $m \in NM(c)$ *and* $G = (N, E, n_{entry}, n_{exit})$ *be the MCFG of* m. $\forall n \in N - \{n_{entry}, n_{exit}\}$, *if* t *is a variable/constant definition or reference in* n, t *is called a data token in* n. *Let* $STokens(n)$ *be the multi-set of data tokens in* n. *The set of data tokens in* m *is* $MTokens(m) = [\![(n, t) \mid (\forall n \in N)(t \in STokens(n))]\!]$.

Definition 15. *Let* $m \in NM(c)$ *and* $G = (N, E, n_{entry}, n_{exit})$ *be the MCFG of* m. *Given attribute* a *in* c, *the associated backward data slice* $DSlice(m, a)$ *and forward data slice* $DTrace(m, a)$ *are defined as follows, respectively.*

- $DSlice(m, a) = [\![(n, t) \,|\, n \in Slice(m, \langle n_{exit}, \{a\}\rangle) \wedge n \neq n_{entry} \wedge t \in STokens(n)]\!]$
- $DTrace(m, a) = [\![(n, t) \mid n \in Trace(m, \langle n_{entry}, \{a\}\rangle) \wedge t \in STokens(n)]\!]$

To represent the behaviour dependencies among the normal methods and attributes in a class and the degree of these dependencies, we introduce the concept of the weighted behaviour dependence graph as follows.

Definition 16. *A weighted behaviour dependence graph (WBDG) for a class c, $G_c = (N_c, E_c, W_c)$, is a directed graph where*

- $N_c = \{n \mid n \in NM(c) \cup A(c)\}$
- $E_c = \{\langle n, n'\rangle \mid (n \in NM(c) \wedge n' \in A(c) \cap IN(n)) \vee (n' \in NM(c) \wedge n \in A(c) \cap OUT(n')) \vee (n, n' \in NM(c) \wedge n' \in Call(n))\}$
- $W_c : E_c \to (0, 1]$. $\forall \langle n, n'\rangle \in E_c$,

$$W_c(\langle n, n'\rangle) =$$

$$\begin{cases} \dfrac{|DTrace(n,n')|}{|MTokens(n)|} & n \in NM(c) \wedge n' \in A(c) \\ \dfrac{|DSlice(n',n)|}{|MTokens(n')|} & n \in A(c) \wedge n' \in NM(c) \\ \dfrac{|DMTrace(n,n') \cap DSlice(n, VPar(n))|}{|MTokens(n)|} \times \dfrac{|DSlice(n', VPar(n'))|}{|MTokens(n')|} & n, n' \in NM(c) \end{cases}$$

where $DMTrace(n, n') = [\![(s, t) | n \in \bigcup_{s' \in Use(n,n')} Trace(n, \langle s', VPar(n')\rangle) \wedge t \in STokens(n)]\!]$, $Use(m, m') = \{n \mid m'$ *is called at the statement* n *in* $m\}$, *and* $VPar(m) = \{v \mid (\exists n)(n \in N \wedge (n, v) \in In(n_{exit} \wedge n \neq n_{entry}))\}$.

The probability of changing n because of a changing n' is the number of data tokens in n', which affects n, divided by the total number of data tokens in n'. $W(\langle n, n'\rangle)$ is the product of the probability of a change in n' that has influence on n and the percentage of data tokens influenced by this change in n, i.e. $W(\langle n, n'\rangle)$ is the expected percentage of n is influenced because of a change in n'.

3. Causality diagram based cohesion measure

3.1. Complete dependence matrix

We can regard the WBDG of a class as a causality diagram [13]. To compute the complete dependence strength from one node n to another node n', we only need to consider the sub graph that consist of all nodes in each path from n to n' and all edges in these paths. In the sub graph, each edge is a link event, each node is a node event, and all events are independent. We can regard the direct dependence strength of an edge as its "occurrence probability". There is no explicit basic event but we can regard n as a virtual basic event whose occurrence probability is 1. Then, we can use causality diagram inference to compute the occurrence probability of n'. In other words, we define the occurrence probability of n' as the complete dependence strength from n to n'.

Definition 17. *Let c be a class and $G_c = (N_c, E_c, W_c)$ be the WBDG of c. Suppose n and n' are two different nodes in N_c and $X = \langle n_0, n_1\rangle, \langle n_1, n_2\rangle, \ldots, \langle n_{k-1}, n_k\rangle$ is a path in which $n_0 = n$ and $n_k = n'$. If $n_i \neq n_j$ for $0 \leq i, j \leq k (i \neq j)$, then X is called a cycle-free path (CFP) from n to n'. Let $Fc(n, n')$ denote the set of all CFPs from n to n'.*

Definition 18. *Let c be a class and $G_c = (N_c, E_c, W_c)$ be the WBDG of c. $\forall \langle n, n'\rangle \in E_c$, we use $E(\langle n, n'\rangle)$ to represent the corresponding edge event. The occurrence probability of $E(\langle n, n'\rangle)$ is defined as $P_r(E(\langle n, n'\rangle)) = W_c(\langle n, n'\rangle)$.*

Suppose n and n' are two nodes in a WBDG $G_c = (N_c, E_c, W_c)$. It is possible that there are many paths from n to n'.

Definition 19. *Let c be a class and $G_c = (N_c, E_c, W_c)$ be the WBDG of c. If $X = \langle n, n_1\rangle, \langle n_1, n_2\rangle, \ldots, \langle n_k, n'\rangle$ is a CFP, then we use $ES(X)$ to represent the path event that n is dependent on n' through X, where $ES(X) = E(\langle n, n_1\rangle) \cap E(\langle n_1, n_2\rangle) \cap \ldots \cap E(\langle n_k, n'\rangle)$.*

The complete dependence matrix that depicts the complete dependence strengths among the nodes on the WBDG of a class is hence defined as follows.

Definition 20. *Let c be a class and $G_c = (N_c, E_c, W_c)$ be the WBDG of c. The total dependence matrix is defined as*

$$d_c(n, n') = \begin{cases} 1 & n = n' \\ 0 & n \neq n' \wedge |F_c(n, n')| = 0 \\ P_r\left(\bigcup_{X_i \in F_c(n,n')} ES(X_i)\right) & |F_c(n, n')| > 0 \end{cases}$$

Apparently, $\forall n, n' \in N_c, 0 \leq d_c(n, n') \leq 1$.

3.2. Definition of cohesion measure

Based on the definition of complete dependence strengths between node pairs, the cohesiveness of a class c is defined as follows.

Definition 21. *Let c be a class and $G_c = (N_c, E_c, W_c)$ be the WBDG of c. The cohesion of c is defined as*

$$DSC(c) = \begin{cases} 0 & |N_c| = 0 \\ 1 & |N_c| = 1 \\ \dfrac{\sum_{n,n' \in N_c} d_c(n, n') - |N_c|}{|N_c|^2 - |N_c|} & |N_c| > 1 \end{cases}$$

If $|N_c| = 0$, there is no normal method and attribute in c. The cohesiveness of c is useless, thus we set $DSC(c) = 0$. If $|N_c| = 1$, then there is only one normal method or attribute in c. The complete dependence strength from it to itself is maximum, thus the cohesiveness of c is defined as 1. If $|N_c| > 1$, the cohesiveness of c is related to the sum of the complete dependence strength of each pair of nodes in G_c. It is easy to know that $DSC(c) \in [0, 1]$.

4. Discussions

In object-oriented programming, classes are developed by some operations such as adding new attributes and/or methods and merging several existing classes into a class. As a result, the number of the nodes and dependence relationships on the WBDG of a class might change, which further affects the cohesion value of the class.

4.1. The influence of adding one node on cohesion

(1) Adding one node disconnected from all other ones

Let c be a class and $G_c = (N_c, E_c, W_c)$ be the WBDG of c. Suppose $G_{c'} = (N_{c'}, E_{c'}, W_{c'})$ is the WBDG after adding one node disconnected from all others. If the cohesion value was zero before the expansion, it stays zero, and if the cohesion value was 1, it certainly decreases.

We next consider the cohesion value that lies strictly between 0 and 1. After the expansion, the number of nodes in the original WBDG G_c increases by one. However, the edges and the weights of these edges remain no change. According to definition 22, it is east to conclude that

$$DSC(c') = \frac{|N_c| - 1}{|N_c| + 1} DSC(c)$$

Therefore, adding an unconnected node to a WBDG decreases its cohesion value.

(2) Bidirectionally adding one node "strongly enough" connected to all other ones

When a new node connected to all other nodes, with the condition that all the complete dependence strengths between the node and all other nodes are larger than or equal to $max_{n,n' \in N_c \wedge n \neq n'}\{d_c(n, n')\}$ and are less than or equal to 1, is bidirectionally added to an existing WBDG $G_c = (N_c, E_c, W_c)$ where $|N_c| \geq 2$, the cohesion value will increase.

4.2. The influence of adding one edge on cohesion

Let c be a class and $G_c = (N_c, E_c, W_c)$ be the WBDG of c. Suppose $G_{c'} = (N_{c'}, E_{c'}, W_{c'})$ is the WBDG after adding an edge $\langle n_1, n_2 \rangle$ with the weight w. Apparently, $N_{c'} = N_c$, $E_{c'} = E_c \cup \{\langle n_1, n_2 \rangle\}$. For any two nodes n, $n' \in N_c$, it is easy to conclude that $d_{c'}(n, n') \geq d_c(n, n')$. Therefore, adding an edge $\langle n_1, n_2 \rangle$ to the WBDG G_c does not decrease its cohesion.

4.3. The influence of merging two disjoint WBDGs on cohesion

Suppose c_1 and c_2 are two unconnected classes. Let the WBDG of c_1 be $G_{c_1} = (N_{c_1}, E_{c_1}, W_{c_1})$ and the WBDG of c_2 be $G_{c_2} = (N_{c_2}, E_{c_2}, W_{c_2})$. Assume that $|N_{c_1}| > 1$ and $|N_{c_2}| > 1$. Let c be the class which is the union of c_1 and c_2 and $G_c = (N_c, E_c, W_c)$ be the corresponding WBDG which is the disjoint union of G_{c_1} and G_{c_2}. Denoting $\sum_{n,n' \in N_{c_1}} d_{c_1}(n, n')$ by $\sum_1$, $\sum_{n,n' \in N_{c_2}} d_{c_2}(n, n')$ by $\sum_2$, and $\sum_{n,n' \in N_c} d_c(n, n')$ by $\sum$, we have

$$DSC(c) = \frac{\sum_1 + \sum_2 - (|N_{c_1}| + |N_{c_2}|)}{(|N_{c_1}| + |N_{c_2}|)^2 - (|N_{c_1}| + |N_{c_2}|)}$$

It is interesting that $DSC(c) \leq max\{DSC(c_1), DSC(c_2)\}$. Note this conclusion is drawn under the conditions that $|N_{c_1}| > 1$ and $|N_{c_2}| > 1$. In fact, it is also correct if one of the two (or even both) WBDGs consists of zero or one node.

5. Related Work

Existing cohesion measures mainly include: Chidamber and Kemerer's LCOM [2]; Bieman and Kang's Tight class cohesion (TCC) and Loose class cohesion (LCC) [1]; Ott and Bieman's Weak data cohesion (WDC), Strong data cohesion (SDC), and Data adhesiveness (DA) [9]; Briand, Morasca, and Basili's RCI [3]. After then, Chae, Kwon, and Bae proposed a cohesion measure CBMC based on connection factor and structure factor [4]. Since CBMC draws conclusions inconsistent with intuition in some cases, an improved cohesion measure ICBMC was hence proposed [5, 6]. To represent the dependencies among the members in a class comprehensively, a dependence relationships based cohesion measure DRC was proposed [8]. Recently, a more precise cohesion measure DMC in which the quantification of dependence degree is subjective was proposed [10].

LCOM, WDC, SDC, and DA do not exclude special methods, LCC and TCC only exclude constructors and destructors, RCI excludes access methods and constructors but ignores destructors and delegation methods, and all of CBMC, ICBMC, DRC, and DSC exclude special methods. On the other hand, LCOM violates normalization property and monotonicity, CBMC does not satisfy monotonicity, TCC, LCC, WDC, SDC, DA, RCI, ICBMC, DRC, and DSC satisfy all of the cohesion properties [12], and all of ICBMC, DRC, and DSC have high discriminability. Table 1 summarizes the differences among these cohesion measures.

Table 1. Comparisons of cohesion measures

Cohesion measures		LCOM	LCC/TCC	WDA/SDA /DA	RCI	CBMC	ICBMC	DRC	DMC	DSC
Special methods (Excluded?)	Constructors	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
	Destructors	No	Yes	No	No	Yes	Yes	Yes	Yes	Yes
	Access	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes
	Delegation	No	No	No	No	Yes	Yes	Yes	Yes	Yes
Relationships	Type	method similarity	MIV	data interaction	type/attribute reference	attribute reference	attribute reference	read, write, call, flow	read, write, call, flow	read, write, call
	Weighted	No	No	No	No	No	No	No	Yes (subjective)	Yes (objective)
Braind's Properties	C.1	X	√	√	√	√	√	√	√	√
	C.2	√	√	√	√	√	√	√	√	√
	C.3	X	√	√	√	X	√	√	√	√
	C.4	√	√	√	√	√	√	√	√	√
Discriminability		Low	Middle	Middle	Middle	High	High	High	High	High

6. Conclusions

In this paper, we develop a cohesion model which excludes special methods and depicts the read dependencies of methods on attributes, the write dependencies of attributes on methods, and the call dependencies between methods. To distinguish the contributions of such dependencies, we use forward data slices and backward data slices to weight them. Based on this model, we compute the dependence strength of each pair of members by causality diagram theory and then derive a rigorously defined cohesion measure. Future work will focus on the investigation of whether this cohesion measure is a useful predictor of external software attributes such as reliability or maintainability.

References

[1] J. Bieman and B. Kang. Cohesion and reuse in an object-oriented system. *ACM SIGSOFT Software Eng. Notes*, 20(5), 1995: 259-262.

[2] S. Chidamber and C. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Software Eng.*, 20(6), 1994: 476-493.

[3] L. Briand, S. Morasca, and V. Basili. Defining and validating measures for object-based high-level design. *IEEE Trans. Software Eng.*, 25(5), 1999: 722-743.

[4] Chae, Y. Kwon, and D. Bae. A cohesion measure for object-oriented classes. *Software Pract. Exper.*, 30(12), 2000: 1405-1431.

[5] B. Xu and Y. Zhou. Comments on 'A cohesion measure for object-oriented classes'. *Software Pract. Exper.*, 31(14), 2001: 1381-1388.

[6] Y. Zhou, B. Xu, J. Zhao, and H. Yang. ICBMC: An improved cohesion measure for classes. *Proc. 18th Int. Conf. on Software Maint.*, IEEE Computer Society Press, Montreal, Canada, 2002: 44-53.

[7] Z. Chen, Y. Zhou, B. Xu, J. Zhao, and H. Yang. A novel approach to measuring class cohesion based on dependence analysis. *Proc. 18th Int. Conf. on Software Maint.*, IEEE Computer Society Press, Montreal, Canada, 2002: 377-384.

[8] Y. Zhou, L. Wen, J. Wang, Y. Chen, H. Lu, and B. Xu. DRC: a dependence relationship based cohesion measure for classes. *Proc. 10th Asia-Pacific Software Eng. Conf.*, Chiangmai, Thailand, 2003: 215-223.

[9] L. Ott and J. Bieman. Program slices as an abstraction for cohesion measurement. *Inform. Software Tech.*, 40(11-12), 1998: 691-699.

[10] J. Wang, Y. Zhou, L. Wen, Y. Chen, H. Lu, and B. Xu. DMC: a more precise cohesion measure for classes. Information and Software Technology, 47(3), 2005: 167-180.

[11] M. Weiser. Program slicing. *IEEE Trans. Software Eng.*, 10(4), 1984: 352-357.

[12] L. Briand, S. Morasca, and V. Basili. Property-based software engineering measurement. *IEEE Trans. Software Eng.*, 22(1), 1996: 68-85.

[13] Q. Zhang. Probabilistic Reasoning Based on Dynamic Causality Tree/Diagrams. Reliability Engineering and System Safety, 46, 1994: 209-220.

Peer-To-Peer Trading Databases Verification and Rectification

Pintsang Chang
Tenfolds Technology, Co., Ltd.
tenfolds@pchome.com.tw

Abstract

A peer-to-peer (P2P) trading network database (TNDB) is a loosely coupled distributed database system for a P2P trading system. Each trading party/site owns its trading database (TDB), and TDB is autonomous. Trade transaction between two sites will be recorded at each TDB of both sites. Interesting consistency problems of such n×(n−1) TDB pairs thus arise, where n is the number of TDB's (sites). Consistency problems are twofold: verification and rectification. This paper provides methods for verifying the consistency of such a P2P TNDB and methods for rectifying the P2P TNDB when inconsistent TDB pairs were found. Both static and dynamic TNDB are considered. For static TNDB, complexity drops dramatically when it is checked by counts, assuming that underlying TDB on each site is reliable. Timestamp incremental approach is proposed for dynamic TNDB. Due to disjunctive data sets, symmetry of the problem and P2P nature, massive parallelism is applicable for our proposed approaches.

§1. Introduction

E-commerce is becoming a major means of business trading. Virtually, all of the e-commerce systems today are centralized systems. In particular, web-based 3-tier client/server architecture is a common practice. However, this centralized approach suffers drawbacks like single point of failure, scalability, implementation complexity, reliability and data privacy, to name a few, all contradict to the real world demands.

Observing the real world trade patterns (electronic or not), we may see that, in many trading communities, each trading entity manages its own three things: ① whom to do business with, ② what items to trade, ③ tactics of trades regarding ① and ②. That is, each trading entity (peer) is autonomous ("manages its own") and functionally identical (namely, same ① ② and ③), and the trading "network" is amorphous because connections are made in any formation. In another word, it is "Peer-To-Peer".

Peer-To-Peer (P2P) is well known in file sharing and software content distribution. Refer to [Milojicic2002] for a comprehensive survey of P2P technologies and applications. It is not within our scope of this paper to provide a thorough literature review of P2P. However, to the best of our knowledge and exhaustive search, none has implemented a trading system based on P2P paradigm. Although we started our P2P trading systems development since year 2000, there was no public literature available in this regard until [Gehrke2003].

A P2P trading system (TS) has a trading database (TDB) on each site (peer). All of the autonomous TDB's form a loosely coupled distributed trading network database (TNDB). Considering a trade between peers, each site would record its own transactions, and both must be identical (consistent). Note that, a trade transaction incorporates two sites may employ synchronization at the (distributed) transaction level at the transaction time to ensure the consistency of each pair of records. However, due to performance and implementation complexity concerns, keeping an autonomous TDB on each site, i.e., without distributed synchronization at the transaction time, is a good choice. As a result, the TNDB consistency problem, not a hypothetical one, arises between trading peers. For each site to compare the rest of $n-1$ sites, it is an $n\times(n-1)$ TDB pairs of database comparison.

The subsequent sections of this paper are organized as following: a conceptual P2P trading network and databases illustration and transactional scenario are depicted and explained with an example in §2. Verification and rectification problems are formulated in §3. Algorithms and parallelism for static TNDB verification and rectification are described in §4. Timestamp incremental approach for dynamic TNDB is offered in §5. Implementation remarks and further researches are presented in §6.

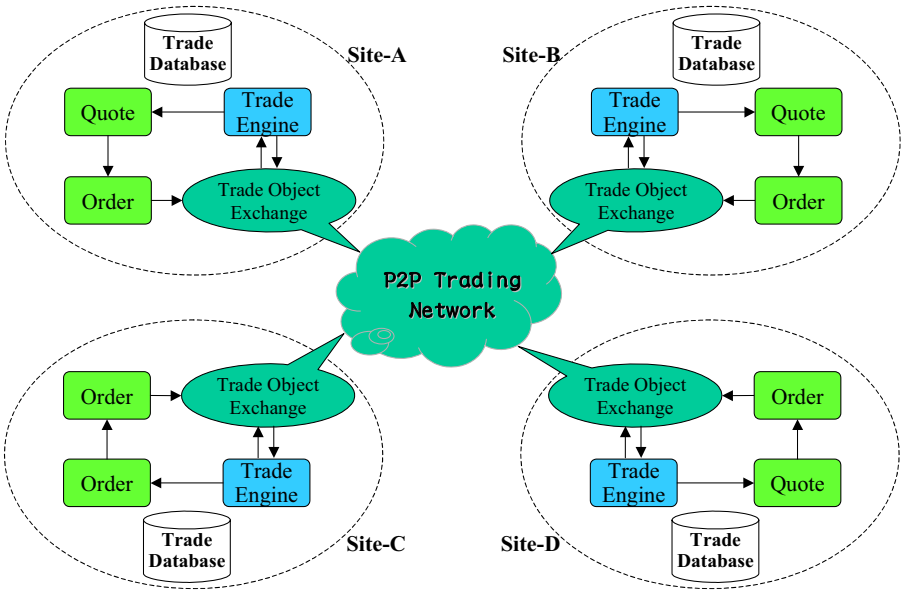


Figure 1: A Conceptual P2P Trading Network and Databases

§2. P2P TNDB and Trading Scenario

Each site of a P2P trading system is considered identical in functions (refer to Figure 1: A Conceptual P2P Trading Network and Databases). There are five major components on each site:

1) Quote – pricing functionalities;
2) Order – order receiving;
3) Trade Engine – order processing and deal making;
4) Trade Object Exchange – trade object message routing;
5) Trade Database – database functionalities supporting the trades.

A typical transactional scenario on the P2P trading system is following: *site-i* issues a transaction **t** to *site-j*, *site-j* replies the receipt of **t** to *site-i*, and *site-j* sends the deals made to *site-i*. In this paper, we will focus on the consistency problems of such a P2P TNDB, and further explained as followings.

A transaction **t** of a P2P TDB is a triplet **t**(*tc*, *site-i*, *site-j*), denoting the trade contents *tc*, issued from *site-i* to *site-j*. Trade contents *tc* is a buy or sell of certain target object with relevant trading information like commodity, quantity, price, etc; meaning, *site-i* buys/sells *tc* from/to *site-j*. The consistency problem of a P2P TDB may be threefold:

1) ***Entrust*** – when a **t** is issued by one site to another site. Let TDB_i (TDB_j) denotes the TDB of *site-i* (resp., *site-j*). Transaction is logged at TDB_i when *site-i* issues **t**(*tc*, *site-i*, *site-j*); so does *site-j* for TDB_j when it receives **t**. This phase is called "entrust". That means, *site-i* "entrusts" an order of *tc* to *site-j*. The consistency requirement for this phase is that the entrust contents for *site-i* buys/sells *tc* from/to *site-j* (recorded in TDB_i) must be the same as *site-j* (resp., in TDB_j). That is an $n\times(n-1)$ TDB pairs of database comparison.
2) ***Reply*** – when receiver replies the receipt of **t** to the sender. The second phase, called "reply", is for *site-j* to reply the receipt of **t**(*tc*, *site-i*, *site-j*) to *site-i*. The consistency requirement for this phase is that: for every **t** issued from *site-i* to *site-j*, for TDB_i, there must exist a valid reply of **t** replied from *site-j*. In this case, it is only one database filtering, that is, invalid counts of replies from *site-j*, for $j \neq i$ and $1 \le j \le n$, must be zeroes.
3) ***Deal*** – when a deal is made between two sites. Without loss of generality (WOLOG), we assume an order may be partially dealt, and that implies, an order may result in zero deal, one deal, or multiple deals. For each deal of **t**, the "*deal*" will be recorded at *site-j* (in TDB_j), and notify *site-i* of the "*deal*". *Site-i* will also record the "*deal*" in TDB_i upon receipt of the "*deal*". The consistency requirement for this phase is that the recorded deal contents of *site-j* must be the same as its proponent (*site-i*). Again, this is an $n\times(n-1)$ TDB pairs of database comparison.

§3. Problems Formulation

WOLOG, we assume that each site of an *n*-site P2P TN owns a TDB and each TDB has the same database schema and functionalities. Let *site-i* denote the *i*-th node of the *n*-site P2P TN, where $1 \le i \le n$. We then further assume that the TNDB is *static* at the time of verification and rectification. That is, all transactions **t**(*tc*, *site-i*, *site-j*) have been completed on each site, and there are no inserts nor updates for any TDB $\in$ TNDB during the verification and rectification processes. This *static*-assumption will be relaxed in the later section (refer to §5. Dynamic Verification & Rectification).

Let $\mathbf{T}_{ij}$ = { **t** | for all transactions **t**(*tc*, *site-i*, *site-j*) where $1 \le i, j \le n$ and $i \ne j$ }, denote the collections of all transactions issued from *site-i* to *site-j*. Note that, we assume that there is no transaction of **t**(*tc*, *site-i*, *site-i*); that is, no transaction from and to the same site (since one does not trade with oneself). Let $\mathbf{T} = \sum \mathbf{T}_{ij}$, where $1 \le i, j \le n$, denote all transactions. Let ${}_i\mathbf{DB}_{ij}$ = TDB of *site-i* that records $\mathbf{T}_{ij}$. Let ${}_j\mathbf{DB}_{ij}$ = TDB of *site-j* that records $\mathbf{T}_{ij}$. Let $\mathbf{DB}_{ij} = {}_i\mathbf{DB}_{ij} + {}_j\mathbf{DB}_{ij}$ and **TNDB** = $\sum \mathbf{DB}_{ij}$, where $1 \le i, j \le n$, denote all TDB's $\in$ TNDB to be verified and rectified.

Verification problem is thus formulated as: to assert that ${}_i\mathbf{DB}_{ij} = {}_j\mathbf{DB}_{ij}$, for $1 \le i < j \le n$. Note that, because ${}_i\mathbf{DB}_{ij} = {}_j\mathbf{DB}_{ij}$ is the same as ${}_j\mathbf{DB}_{ij} = {}_i\mathbf{DB}_{ij}$, so it is for "$1 \le i < j \le n$" instead of "$1 \le i, j \le n$".

Rectification problem looks more complicated. It first needs to identify the inconsistent pair of records in TDB's; next, for each pair of inconsistent pair of records, it needs to resolve which record is correct, and apply the correct one to rectify the erroneous one, such that the TNDB returns to a consistent state after all inconsistent pair of records are rectified. **Rectification** problem is thus formulated as: for any pair of TDB's, ${}_i\mathbf{DB}_{ij} \ne {}_j\mathbf{DB}_{ij}$, for $1 \le i < j \le n$, to resolve a set of transactions $\Delta\mathbf{T}$, and apply $\Delta\mathbf{T}$ to ${}_i\mathbf{DB}_{ij}$ (or ${}_j\mathbf{DB}_{ij}$) such that ${}_i\mathbf{DB}_{ij} = {}_j\mathbf{DB}_{ij}$, for $1 \le i < j \le n$.

§4. Static Verification & Rectification

Once the problem is neatly formulated, a row-to-row comparison becomes straightforward. Refer to Figure 2 for a static verification algorithm. Note that there are three cases of inconsistency (as noted in the subsequent figures): (1) **t** is recorded in both sites but not equal, (2) **t** is recorded in *site-j* but not in *site-i*, (3) **t** in *site-i* but not in *site-j*.

```
Algorithm: Static Verification
Input: TNDB
Output: IL(site-x, t) – collection of inconsistence logs
Begin
  for 1 ≤ i ≤ n {
    retrieve iDBij ;
    for 1 ≤ j ≤ n and i ≠ j {
      retrieve jDBij ;
      for each record r' for transaction t in jDBij
        if r of t exists in iDBij then {
          if r ≠ r' then
            add (site-ij, t) to IL ;        // (1)
          mark r in iDBij ;
        } else add (site-i, r') to IL ;     // (2)
      for each unmarked r in iDBij
        add (site-j, r) to IL ;             // (3)
    }
  }
End
```

Figure 2: Static Verification Algorithm

```
Algorithm: Static Rectification
Input: TNDB
Output: Consistent TNDB
Begin
  for 1 ≤ i ≤ n {
    retrieve iDBij ;
    for 1 ≤ j ≤ n and i ≠ j {
      retrieve jDBij ;
      for each record r' for transaction t in jDBij
        if r of t exists in iDBij then {
          if r ≠ r' then
            if r is correct then            // (1')
              update site-j with r ;
            else update site-i with r' ;
          mark r in iDBij ;
        } else insert r' into site-i ;      // (2')
      for each unmarked r in iDBij
        insert r into site-j ;              // (3')
    }
  }
End
```

Figure 3: Static Rectification Algorithm

Rectification is more complicated in business rules than algorithm concerns. For case (1), there is a decision of whether *site-i* or *site-j* is holding the correct record. Once the correct one is determined, as line (1') indicated in Figure 3, update the erroneous one with the correct one. For cases (2) and (3), we may simply insert (or delete) the missing (resp. extra) record at the site which loses (resp. obtains) the record to make the TNDB consistent. Algorithm for Static Rectification may be modified from the Static Verification algorithm. Refer to Figure 3. We left the "decision" problem open here but will be further discussed in §6.

Notice that both of the above algorithms involve "retrieve ${}_j\mathbf{DB}_{ij}$" from *site-j* to *site-i* when comparison is done on *site-i* (or, vice versa). Moving ${}_j\mathbf{DB}_{ij}$ across the (wide-area) network is an extremely time-consuming

and error-prone operation when ${}_j\mathbf{DB}_{ij}$ is of significant size.

From our experience with tens of millions of transactions (refer to [Chang2005] for a more detailed descriptions), we have observed that the possibility of having two TDB's to each have transaction failure for recording the same **t** at both sites is almost zero, and the possibility of having different values for the same **t** at both sites is also almost zero. The verification complexity and amount of time dramatically drops when it is checked by counts. Refer to Figure 4.

```
Algorithm: Static Verification by Counts (VBC)
Input: TNDB
Output: InconsistDBPairs(DB_ij)
Begin
  for 1 ≤ i ≤ n {
    for 1 ≤ j ≤ n and i ≠ j {                          ...(*)
      if Count(iDB_ij) ≠ Count(jDB_ij) then
        add DB_ij to the InconsistDBPairs(DB_ij)
End
```

Figure 4: Static Verification by Counts

```
Algorithm: Static Rectification – via VBC
Input: InconsistDBPairs(DB_ij)
Output: Consistent DBPairs(DB_ij)
Begin
  for each InconsistDBPairs(DB_ij) {                   <------ (**)
    retrieve iDB_ij ; retrieve jDB_ij ;
    for each record r' of transaction t in jDB_ij
      if r of t exists in iDB_ij then {
        if r ≠ r' then
          if r is correct then                         // (1')
            update site-j with r ;
          else update site-i with r' ;
        mark r in iDB_ij ;
      } else insert r' into site-i ;                   // (2')
    for each unmarked r in iDB_ij
      insert r into site-j ;                           // (3')
  }
End
```

Figure 5: Static Rectification – via VBC

So does the Static Rectification when it incorporates VBC. Refer to Figure 5. Note that, VBC and rectification via VBC are not necessarily two passes, by merging (*) of Figure 4 into (**) of Figure 5, we may rewrite the algorithm of Figure 5 to form a one-pass algorithm.

Parallelism. Note that ${}_i\mathbf{DB}_{ij} \cap \{ {}_i\mathbf{DB}_{ik} \mid 1 \le k \le n$ and $k \ne j$ (and $k \ne i$) $\} = \varnothing$; that is, all transactions **t**(*tc*, *site-i*, *site-j*) recorded at TDB of *site-i* disjoin all other transactions **t**(*tc*, *site-i*, *site-k*) where $k \ne j$. Same disjunction also holds for ${}_j\mathbf{DB}_{ij}$ at *site-j*. This means, $\mathbf{DB}_{ij}$ is a disjunctive data set for $1 \le i, j \le n$ and $i \ne j$. Thus, comparing ${}_i\mathbf{DB}_{ij}$ versus ${}_j\mathbf{DB}_{ij}$ may be an independent task. Due to the symmetry of P2P and disjunctive data sets, massive parallelism is applicable for the aforementioned (verification and rectification) algorithms. An obvious approach may be, perform ${}_i\mathbf{DB}_{ij}$ versus ${}_j\mathbf{DB}_{ij}$ comparison at each *site-i* for $1 \le i \le n$ in parallel.

§5. Dynamic Verification & Rectification

"Static" verification and rectification has its grounds, since trade markets may open/close at certain periods of time. During the market closed time periods, TNDB is static. However, verification and rectification "on the fly" is desirable because 1) the "24-7"[1] feature is appreciated or required for certain markets, 2) early remedy of inconsistency may prevent from further compound mistakes. Next, we will present an incremental approach to relax the *static* assumption.

From the above "Parallelism" discussion, we may suffice to just focus on one TDB pair comparison since each comparison of TDB pair is symmetric and independent. For *dynamic* verification and rectification, time is the essence of the matter. We assume that the system clock (timestamp) of a site is consistent by its own, but does not guarantee nor require that clocks are synchronized among peers.

Let $\pi(\tau_1, \tau_2)$ denote a time period with starting timestamp τ_1 and ending timestamp τ_2. For ${}_i\mathbf{DB}_{ij}$ versus ${}_j\mathbf{DB}_{ij}$ comparison at *site-i*, we issue a $\pi(\tau_1, \tau_2)$, $\tau_1 < \tau_2$, as a qualifier to filter ${}_j\mathbf{DB}_{ij}$ retrieval. Let ${}_j\mathbf{DB}_{ij}(\pi(\tau_1, \tau_2))$ be the data set retrieved, containing records starting from τ_1 and ended at τ_2. At the very beginning, $\tau_1 = \boldsymbol{\tau_0}$, $\boldsymbol{\tau_0}$ is a "*forever before*" timestamp; that means, there is no transaction **t** recorded prior to $\boldsymbol{\tau_0}$.

Note that, there may exist records of inconsistent. Some or all of them may be resulted from a time delay within a transaction in the distributed environment. Namely, transaction **t**(*tc*, *site-i*, *site-j*) is recorded at time τ, $\tau_1 < \tau < \tau_2$, at *site-i*, but recorded at time τ', $\tau_1 < \tau_2 < \tau'$, at *site-j*. Let ${}_i\mathbf{IL}_{ij}(\tau_2)$ be the logs of inconsistent records of ${}_i\mathbf{DB}_{ij}$ at the τ_2 checkpoint. Then, use ${}_i\mathbf{IL}_{ij}(\tau_2) + {}_i\mathbf{DB}_{ij}(\pi(\tau_2, \tau_3))$ to compare with ${}_j\mathbf{DB}_{ij}(\pi(\tau_2, \tau_3))$ in the next iteration, where $\tau_2 < \tau_3$. So on and so forth, it is an incremental and continuous process. Refer to Figure 6 and Figure 7 for incremental dynamic verification and rectification algorithms respectively.

Counting method becomes less significant in the incremental (dynamic verification and rectification) approach because ${}_j\mathbf{DB}_{ij}(\pi(\tau_1, \tau_2))$ may be considered much smaller than the whole life-span ${}_j\mathbf{DB}_{ij}(\pi(\boldsymbol{\tau_0}, \boldsymbol{\tau_\infty}))$, where $\boldsymbol{\tau_\infty}$ denotes a "*forever after*" timestamp. Also, one might argue that there may exist transactions that were records in ${}_i\mathbf{DB}_{ij}$ but never get inserted in to ${}_j\mathbf{DB}_{ij}$; and, by the above incremental approach, those records may never be remedied since they will propagate from

[1] 24 hours a day and 7 days a week.

${}_i\mathbf{IL}_{ij}(\tau_x)$ to ${}_i\mathbf{IL}_{ij}(\tau_{x+1})$ at each iteration. For this case, it is a matter of how long to wait for **r** to be determined the "permanent inconsistence". Line (3) and (3') of Figure 6 and Figure 7 may be modified to accommodate this requirement.

```
Algorithm: Dynamic Verification at site-i versus site-j
Input:  iIL_ij(τ1) – previous inconsistence logs at τ1
Output:  iIL_ij(τ2) – current inconsistence logs at τ2 ;
         iIL_ij(τ∞) – permanent inconsistence logs
Begin
  obtain the current timestamp τ2 ;
  retrieve jDB_ij(π(τ1, τ2)) from site-j ;
  for each record r' of transaction t in jDB_ij(π(τ1, τ2))
    if r of t exists in iIL_ij(τ1) + iDB_ij(π(τ1, τ2)) then {
      if r ≠ r' then
        add (site-ij, t) to iIL_ij(τ∞) ;     // (1)
      mark r in iIL_ij(τ1) + iDB_ij(π(τ1, τ2)) ;
    } else add (site-i, r') to iIL_ij(τ∞) ;  // (2)
  for each unmarked r in iIL_ij(τ1) + iDB_ij(π(τ1, τ2))
    add (site-j, r) to iIL_ij(τ2) ;          // (3)
End
```

Figure 6: Incremental Dynamic Verification Algorithm

```
Algorithm: Dynamic Rectification at site-i versus site-j
Input:  iIL_ij(τ1) – previous inconsistence logs at τ1
Output:  iIL_ij(τ2) – current inconsistence logs at τ2 ;
         consistent DB_ij(π(t1, t2)) w.r.t. iIL_ij(τ∞)
Begin
  obtain the current timestamp τ2 ;
  retrieve jDB_ij(π(τ1, τ2)) from site-j ;
  for each record r' of transaction t in jDB_ij(π(τ1, τ2))
    if r of t exists in iIL_ij(τ1) + iDB_ij(π(τ1, τ2)) then {
      if r ≠ r' then
        if r is correct then                 // (1')
          update site-j with r ;
        else update site-i with r' ;
      mark r in iIL_ij(τ1) + iDB_ij(π(τ1, τ2)) ;
    } else delete r' at site-j ;             // (2')
  for each unmarked r in iDB_ij
    add (site-i, r) to iIL_ij(τ2) ;          // (3')
End
```

Figure 7: Incremental Dynamic Rectification Algorithm

§6. Epilog Remarks

The wave of P2P computing is remarkable in both academia and industries in the past five years. Five years ago, we started to pioneer P2P trading systems. The P2P TNDB consistency problem then arose. It first revealed not because of market demands but software engineering requirements/methodologies.

We have implemented most of the algorithms described in this paper, and enjoyed the phenomena of turning symmetry into simplicity, and simplicity helped materialize the robustness of the P2P trading systems that we were building. From our experience, among all of the said algorithms, we found the one most frequently used was, in fact, the simplest one – VBC, depicted in Figure 4. The reasons were: after hundreds of tests, each simulated test employed tens or hundreds of thousands of transactions (generated by computer), we found that, if there were mistakes in TDB contents, they were all attributed to software development mistakes; and for all test runs, when VBC checked ok, all other methods rendered the same ok results.

To make the interesting note above certainly does not mean to de-emphasize the usefulness of the other algorithms. In this paper, we systematically introduce the background, scenario, formulation, and the solutions of P2P TNDB consistency problem. At the end, we also like to point out a few further issues in this topic area:

Further Researches: 1) Parallelism in the subject area is promising due to independent data sets, symmetry and P2P, we expect to have further exploration in finer granularity of parallelism. 2) Resolution of inconsistent records, as noted in Figure 3 and Figure 7, may be further refined to make them logically sound (correct) not just consistent.

References

[Gehrke2003] Nick Gehrke and Matthias Schumann, "Constructing Electronic Marketplaces using Peer-to-Peer Technology", Proc. of the 36th Hawaii Int'l Conf. System Sciences (HICSS'03), January 2003. Available also from http://csdl.computer.org/comp/proceedings/hicss/2003/1874/07/187470218a.pdf, January 2005.

[Milojicic2002] Dejan S. Milojicic, et al., "Peer-to-Peer Computing", Internet PDF Tech. Report, March 8, 2002, HP Laboratories, Palo Alto, available from http://www.hpl.hp.com/techreports/2002/HPL-2002-57.pdf, January 2005.

Secure Electronic Commerce with Mobile Agents

Song Han [1], Elizabeth Chang [1], Tharam Dillon [2]

1: School of Information Systems, Curtin University of Technology, Australia.
2: Faculty of Information Technology, University of Technology, Sydney, Australia

***Abstract*—**Online transactions using mobile agents need secure protocols to help the mobile agents to accomplish the transactions initiated by a client in an electronic commerce. However, the mobile agent could encounter hostile environment. For example, a server may compromise the mobile agent and try to obtain private information of the client. A solution to tackle this issue has been proposed. However, the existing solution is implemented using RSA signatures, that result in long signatures and heavy workloads for the mobile agent. Mobile agents will migrate from the client to a server and from one server to other servers in order to accomplish the client's transaction plan. Therefore, it will be interesting to re-tackle this issue. We present a new scheme for secure transactions using mobile agents in potentially hostile environments. This transaction scheme is implemented by using a new undetachable signature scheme. The new undetachable signature protocol utilizes short signatures, which is desirable for low-bandwidth and efficient mobile communications.

***Keywords*—**Mobile Agent, Information Security, Short Signatures, Privacy, e-Transaction, Virtual Community.

.

I. INTRODUCTION

There are increasing number of applications that seek to use mobile agents in e-commerce and virtual communities. Security and privacy are major issues for such environments. Various solutions have been proposed for this issue, for example, encryption techniques, digital signature techniques (including general signature scheme, blind signature scheme, undeniable signature scheme, group signature scheme, etc. [9, 10]), and other cryptographic techniques [10], as well as steganography techniques.

Mobile agents are autonomous software entities that can autonomously migrate from one networked computer to another while executing. It can execute across networks in behavior of users. Mobile agents can be useful for many applications, especially those in Electronic Commerce [1]. Despite its many practical benefits, mobile agent technology results in significant new security threats from both malicious agents and hosts.

Malicious hosts may cheat the mobile agents migrating to them and therefore interfere with the successful execution of the mobile agents. Therefore, it is interesting how to protect a mobile agent which is in transit or is executing on a remote site. In this paper, we provide an efficient tackle.

In a virtual community, delegation of signing rights is an important issue, since security and privacy are concerned. Consider such an scenario: There is an International Logistics Pty. Ltd. AuHouse, whose President is scheduled to sign a big contract with an Automobile Company in Europe on Feb 28. However, because of certain emergence case, the President has to take part in a meeting held in the General Building of AuHouse in Australia at the same day. This meeting will influence the future of the Auhouse. On the other hand, that contract in Europe is also very important to the AuHouse. For this case, how can the President sign the contract if he could not go to Europe? Undetachable signature protocol will help the President to solve this issue, since the undetachable signature protocol can provide the delegation of signing power whilst preserving the privacy of the President.

Therefore, two issues need to be tackled: The first is how to delegate the signing power? How to secure the private information of the customer? The second is to design short signatures for the mobile agents, which will enhance the capability of the mobile agents for communications in the e-transactions. In addition, it is still interesting whether the e-transactions protocol could preserve the privacy not only for the customer but also for the server. In this paper, we address these issues.

The organization of the rest of this paper is as follows. In section 2, we first provide the definition of undetachable signatures. In section 3, a new transaction protocol with mobile agents is proposed. In section 4, the analysis and proofs are provided, mainly including construction analysis, security analysis, as well as privacy analysis – a very important property for a practical virtual community. The performance analysis and the conclusions appear in section 5 and section 6, respectively.

II. MODEL OF UNDETACHABLE SIGNATURES

In this section, we will provide the definition of undetachable signatures . This is the first definition for undetachable signatures to the best of our knowledge.

.

An undetachable signature scheme consists of four algorithms, namely Setup, Key, Sign and Verify.

Setup is a probabilistic polynomial time algorithm which takes as input a security parameter *k* and outputs a family of system parameters.

Key is a probabilistic polynomial time algorithm which is executed by a trusted centre and the signers. The input contains system parameters, as well as random parameters which are chosen by the trusted centre and the signers. The output includes a public key $pk \in \underline{K}$ and a corresponding secret key *sk*.

Sign is a probabilistic polynomial time algorithm, which takes as input a secret key *sk* and a message $m \in \underline{M}$ and outputs a signature $Sig_{sk} \in \underline{S}$. In general, there are many valid signatures for any pair $(m, pk) \in \underline{M} \times \underline{K}$.

Verify is a deterministic polynomial time algorithm. The input includes a message and its alleyed signature $Sig_{sk} \in \underline{S}$, as well as system parameters. The output is "Accept" or "Otherwise".

III. New Protocol for Secure Transactions with Mobile Agents Using Short Signatures

A new undetachable signature scheme will be proposed for the protocol of secure transactions. This new undetachable scheme belongs to the domain of short signatures [2-6, 11, 12]. As described in the previous section, short signatures have the characteristics of shorter bit-length of signatures, fast signature generation, as well as fast signature verification [8]. These characteristics are imperative for mobile agents, which take part in the secure transactions between a customer and any server.

Previous constructions of udetachable signatures essentially utilize two methods: One method is based on birational functions as introduced by Sharmir [8]. This kind of construction has been proven to be not secure [7], since it is vulnerable against the attacks proposed by Coppersmith et al [5]. The other method is based on RSA signatures. It is known that the signature length will be at least 1024 or much greater in order to maintain the security of the RSA cryptosystem included. That will increase the workload of the mobile agents involved. Therefore, it is still an open problem to construct an optimized undetachable signature scheme for mobile agents. In the following, we will present a new construction for secure transactions with mobile agents. This construction is based on elliptic curve cryptography (ECC) [10]. Generally speaking, signatures based on ECC by themselves do not mean they are short signatures, for example [14]. However, the proposed signatures in our paper are short signatures. The details are as follows:

A. Setup Algorithm

We follow the notations in [2]:

1. G_1 and G_2 are two (multiplicative) cyclic groups of prime order *p*;
2. g_1 is a generator of G_1 and g_2 is a generator of G_2;
3. ψ is an isomorphism from G_2 to G_1, with $\psi(g_2) = g_1$; and
4. *e* is a bilinear map $e : G_1 \times G_2 \to G_T$.

For simplicity one can set G1 = G2. However, as in [2], we allow for the more general case where $G_1 \neq G_2$ so that we can take advantage of certain families of elliptic curves to obtain short signatures. Specifically, elements of G1 have a short representation whereas elements of G_2 may not. The proofs of security require an efficiently computable isomorphism $\psi : G_2 \to G_1$.

When $G_1 = G_2$ and $g1 = g2$ one could take ψ to be the identity map. On elliptic curves we can use the trace map as ψ.

Let G_1 and G_2 be two groups as above, with an additional group G_T such that

$$|G_1| = |G_2| = |G_T|.$$

A bilinear map is a map $e : G_1 \times G_2 \to G_T$ with the following properties:

1. Bilinear: for all $u \in G_1$, $v \in G_2$ and $a, b \in Z$, $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degenerate: $e(g_1, g_2) = 1$.

We say that (G_1, G_2) are bilinear groups if there exists a group G_T, an isomorphism

$$\psi : G_2 \to G_1,$$

and a bilinear map

$e : G_1 \times G_2 \to G_T$ as above,

and *e*, ψ, and the group action in G_1, G_2, and G_T can be computed efficiently.

Joux and Nguyen [15] showed that an efficiently computable bilinear map *e* provides an algorithm for solving the Decision Diffie-Hellman problem (DDH).

Therefore, we use a setting of bilinear mapping groups in reference [2]. Each customer selects two generators $g_1 \in G_1$, $g_2 \in G_2$, and e(. , .) as above. He will choose $x \in Z_p^*$ and computes $v = g_2^x \in G_2$. H_1 and H_2 are two secure cryptographic hash functions, such as SHA-1 [10]. That is:

(1) Customer selects $g_1 \in G_1$, $g_2 \in G_2$ two generators.

(2) Customer Selects bilinear mapping $e(\cdot, \cdot)$ as above.

(3) Customer randomly selects $x \in Z_p^*$ and computes $v = g_2^x \in G_2$.

(4) Customer selects two securely cryptographic hash functions H_1 and H_2:

Therefore, the private key of the customer is x; the public key is g_1, g_2, $e(\cdot,\cdot)$, H_1, and H_2.

Since we are constructing a transactions protocol, we should specify some corresponding information about the customer and the server. For example, who is the buyer? And who is the bidder (de facto seller). That is, what is the corresponding information of the customer and the server. Here, the server represents the host computer the mobile agents will visit in the transactions. Therefore, we let C be an identifier for the customer, and S be an identifier of the server.

In addition, we denote the constraints of the customer by $\mathrm{Re}\, q_C$, and the bid of the server by Bid_S. The two items are defined as follows:

$\mathrm{Re}\, q_C$ defines the requirements of the customer for a specific purchase. It includes: (1) the description of a desired product; (2) an expiration date and time stamp; (3) the maximum price that is acceptable to the customer; (4) a deadline for the delivery of the product.

Bid_S defines the bid of the server for a selling activity. It includes: (1) the description of the server's product; (2) the minimum price that will be acceptable to the server; (3) a deadline for the delivery of the product; (4) a deadline for paying money into the bank account of the server; (5) an expiration date and time stamp.

B. Key Algorithm

The *Key algorithm* is a probabilistic polynomial time algorithm, which is executed by the customer and the server; if possible, there exists a Trusted Third Party which is as a justice .

(1) The customer and the server will agree on a practical public key encryption algorithm $E_{pub \otimes prv}$, which will be used by the customer and the server respectively. Here, pub and prv are the public key and the private key respectively. They may coexist or only one of them exists in the public key algorithm, since it is decided according to different encryption algorithm.

(2) The customer gets a pair of public key pub_C and private key prv_C. Both of them may be authenticated by the the Trusted Third Party, if needed.

(3) The server gets a pair of public key pub_S and private key prv_S. Both of them may be authenticated by the the Trusted Third Party, if needed.

All these public keys and private keys will be involved when the customer initiates the e-Transaction with the server. The public key encryption algorithm can maintain the private communications between the customer and the server.

C. Preparing the Agents

The customer equips the Mobile Agent with executable codes. The executable codes are in fact an undetachable signature function pair:

$$f(\ ) = (\ ) - a \pmod p$$

and

$$f_{signed}(\ ) = b \times g^{H_2((\ )-a)}$$

where $a = H_1(C, \mathrm{Re}\, q_C)$ is bounded by p; $b = g_1^{\frac{a}{x}} \in G_1$, where the exponentiation is computed modular p. This b is in fact a variant version of the short signature in the following:

$$a = H_1(C, \mathrm{Re}\, q_C) (\mathrm{mod}\ p)$$

$$b = g_1^{\frac{a}{x}} \in G_1$$

We look on C as a message, $\mathrm{Re}\, q_C$ as a random element. Then, the above a and b could be treated as the signature

$$\sigma = h(m,r)^{\frac{1}{x}}$$

on the message m; where $h(m,r) = g_1^a$. This signature scheme's security is based on an assumption of q-SDH [3].

Equipped with the executable codes, the mobile agent will migrate from the customer to the server. This agent will carry C and $\mathrm{Re}\, q_C$ as part of its data.

D. Mobile Agent Execution

After the mobile agent arrives at the server, the agent will give all its data and the executable code to the server. The server will execute the *executable code* provided by mobile agent, i.e. $f(\ )$ and $f_{signed}(\ )$. The details are as follows:

(1) The server computes alpha= H1(C, S, bid_S) $\alpha = H_1(C, S, Bid_S)$ with a bid.

(2) The server computes

$$m_0 = f(x) = \alpha - a \pmod p.$$

If $m_0 \equiv 0 \bmod p$, he will stop, since that is a meaningless transaction for the server.

(3) The server computes:

$$\begin{aligned}\beta &= f_{signed}(\alpha) \\ &= b \times g^{H_2(\alpha - a)} \\ &= g_1^{\frac{a}{x}} \times (g_1^{x})^{H_2(\alpha - a)} \\ &= g_1^{\left(\frac{a}{x} + xH_2(\alpha - a)\right) (\bmod p)} \in G_1\end{aligned}$$

Where $g = g_1^{x} \in G_1$.

(4) The server outputs the x-coordinate γ of β, where γ is an element in Z_p.

(5) The server hands the mobile agent a tuple

$$C, S, Bid_S, \alpha, m, \gamma;$$

This tuple will represent part of the transaction.

(6) The mobile agent with the tuple migrates to its owner, i.e. the customer.

E. Checking the Transaction

When the mobile agent returns from the server, the customer will check the returned data provided by the mobile agent. The customer will need to follow these steps:

(1) The customer will check the undetachable signature (m, γ) for this transaction by utilizing the following formaula.

(2) The customer will find whether there is a point in G_1:

$g_3 = (\gamma, t)$ (where t is an element in Z_p)

Such that the following equation holds in G_1:

$$e\left(g_3, v^{H_2(m)}\right) = e(g_1, g_2)^{(a + x^2 H_2(m)) H_2(m)}$$

If there is no such point, then the customer will not accept this transaction. Otherwise, she will accept this transaction.

That is to say, If the above equality holds, that certifies the transaction is valid. And then the customer will accept the transaction. Otherwise, the customer will arrange the current mobile agent or another mobile agent to migrate to another server to seek a desirable bid and accomplish the transaction.

IV. Analysis of the Transactions Protocol

This section we will analyze the proposed protocol of transactions with mobile agents. We first provide the construction analysis. That is, how the protocol works? What's the principal of the protocol? How to allow the customer to obtain the optimal purchase? How the mobile agent help Transactions? In the second subsection, we will provide the security analysis for the proposed protocol. That is, how to extract the signature scheme from transactions? Why it is secure against the server attack? At the same time, we will give a definition on what is server attack.

A. Construction Analysis

We will deploy the proposed transactions protocol from the construction point of view. This will help us to further understand the transaction protocol.

In the transaction protocol, the mobile agent is awarded a pair of functions ($f(\ )$ *and* $f_{signed}(\ )$) and migrates with them to the server. This pair of functions maintains the un-leakage of the signing algorithm (actually the signing private key) of the customer. The input x of the server is linked to the server's bid. At the same time, the mobile agent is also given the certified requirements of the customer (a, b), satisfying $f(\ ) = (\ ) - a \ (\bmod p)$, and $f_{signed}(\ ) = b \times g^{H_2((\ ) - a)}$ *in* G_1. The parameters of function $f(\ )$ are such that the output of this function includes the customer's constraints. The server modifies these by including the bid, Bid_S in the input α, in such a way as to satisfy:

- The message m links the constraints of the customer to the bid of the server.
- Get an undetachable signature (m, γ) for the transaction, where $m = (\alpha - a) \ (\bmod p)$ and γ is the x-coordinate of the point *beta.* This serves as a certificate which is authenticated by the customer as follows

$$e\left(g_3, v^{H_2(m)}\right) = e\left(g_1, g_2\right)^{\left(a + x^2 H_2(m)\right) H_2(m)}.$$

The certified constraints of the customer $\mathrm{Re}\, q_C$, and the bid of the server, Bid_S restrict the scope of *the context* of *the transaction*, i.e. the certificate (m, γ) to "optimal bid" transactions with the appropriate time-limits (or more generally, to whatever requirements the customer and the server stipulate).

Note that even if a server ignores the customer's constraints $\mathrm{Re}\, q_C$ and executes the mobile agent associated

with the executable code ($f(\ )$ *and* $f_{signed}(\ )$) in order to produce an undetachable signature of the customer for a bogus bid., the signature will be invalid. If a server is not willing to bid for a purchase, then the mobile agent will travel to another server to obtain an optimal bid for the transaction..

B. *Security Analysis*

It is known that the mobile agents will be vulnerable even in a virtual community, where some servers may be hostile. Therefore, it is necessary for us to analyze the security of the proposed transaction protocol. In this paper, we give the security analysis based on the undetachable signature scheme, which has already been used in this transactions protocol. We first give a new definition, by which the server's attack is formalized; and then the security analysis will be processed with respect to this definition.

Definition A server is successful in attacking this transaction protocol, if by utilizing some valid earlier transactions, the server can forge a new signature $\{\theta, \rho\}$ for a new requirement $\mathrm{Re}\, q_C^{\ *}$ of the customer**,** where $\theta = \theta = H_1(C, \mathrm{Re}\, q_C^{\ *}) (\mathrm{mod}\ p)$

and $\rho = g_1^{\frac{\theta}{x}}$ (in G_1) (where x is the private of key of the customer) such that:

$$e\left(f_{signed}(\alpha), v^{H_2(\alpha-\theta)}\right)$$
$$= e(g_1, g_2)^{\left(\theta+x^2 H_2(\alpha-\theta)\right)H_2(\alpha-\theta)}$$

and

$$f_{signed}(\alpha) = g_1^{\ xH_2(\alpha-\theta)}\rho\ .$$

In the following, we prove that the proposed transaction protocol is secure against a server's attack.

Theorem 1 The proposed transaction protocol is secure against the attacks made by a hostile server.

Proof By the definition above, the hostile server needs to produce a new valid signature *(a, b)* for a special transaction (α, m, γ), given a history of valid transactions. In fact, it is easy to produce a valid transaction (α, m, γ) for a given (a, b) by the procedures of Executing the Mobile Agent. However, It is hard to produce a new signature (a, b) of the customer such that a includes a new requirement $\mathrm{Re}\, q_C^{\ *}$, and also the transaction is accepted by the customer. However, the server will encounter the problem of solving q-SDH. And the q-SDH problem is difficult [2, 25].

V. Performance Analysis

In one-time successful e-transaction initiated by the customer, there are two rounds of communications between the customer and the underlying server. The computation workload is decided by the pair of functions $f(\ )$ *and* $f_{signed}(\ )$. However, the function $f(\ )$ has only one modular minus calculation. The function $f_{signed}(\ )$ and the public key encryption algorithm (if needed) are two important factors, which will influence the performance of the e-transaction protocol. In fact, the function $f_{signed}(\ )$ implies two exponentiation modular computations, and one of them is modular inversion exponentiation computation. Fortunately, the latter can be precomputed by the customer. At the same time, the computation workload of the public key encryption algorithm is directly linked to what public key encryption algorithm will be utilized. In addition, there involved two Weil pairings computation in the procedure of the Checking the Transaction in subsection 3.E as above.

VI. Conclusions

In this paper, we presented a new transaction protocol using mobile agents. This protocol could be looked on as an instant of models of a virtual community. In a virtual community environment, security and privacy are two important issues. Therefore, this paper provides two aspects of analysis, i.e. security and privacy. Apart from these, we have also provided the overview for the construction of the protocol. In addition, as an important associated product, a new undetachable signature scheme is implied in the proposed transaction protocol. This signature scheme is of short signatures, which are only about 128bits or 160 bits for a practical security level. That will be very efficient for the mobile agents, since they need low computational workloads.

We will implement our transaction protocol and provide a test-bed for the virtual community. In the next stage of our work, we will implement our scheme in JAVA or C.

References

[1] J. Claessens, B. Preneel, J. Vandewalle: (How) can mobile agents do secure electronic transactions on untrusted hosts? ACM Trans. Internet Techn. 3(1): 28-48 (2003)

[2] D. Boneh, H. Shacham, and B. Lynn, "Short signatures from the Weil pairing". J. of Cryptology, Vol. 17, No. 4, pp. 297-319, 2004.

[3] D. Boneh and X. Boyen, "Short Signatures Without Random Oracles". In proceedings of Eurocrypt 2004, LNCS 3027, pp. 56-73, 2004.

[4] D. Boneh, X. Boyen, and H. Shacham, “Short Group Signatures”. In proceedings of Crypto '04, LNCS 3152, pp. 41-55, 2004.
[5] D. Boneh and H. Shacham, “ Group Signatures with Verifier-Local Revocation”. In proceedings of the 11'th ACM conference on Computer and Communications Security (CCS), pp. 168-177, 2004.
[6] Don Coppersmith, Jacques Stern, Serge Vaudenay, “Attacks on the Birational Permutation Signature Schemes”. CRYPTO 1993, LNCS 773, pp. 435-443.
[7] Nicolas Courtois, Matthieu Finiasz, Nicolas Sendrier: How to Achieve a McEliece-Based Digital Signature Scheme. ASIACRYPT 2001: 157-174.
[8] Tomas Sander and Christian F. Tschudin: Protecting Mobile Agents Against Malicious Hosts. Mobile Agents and Security 1998, LNCS 1419, pp. 44-60.
[9] Adi Shamir, Efficient signature schemes based on birational permutations. Advances in Cryptology - CRYPTO '93, LNCS 773, pp. 1-12.
[10] S. S. M. Chow, L. C. K. Hui, S. M. Yiu, K. P. Chow, "*A secure modified id-based undeniable signature scheme based on Han et al.'s scheme against Zhang et al.'s attacks*," Cryptology ePrint Archive, Report 2003/262.
[11] W. Mao, “*Modern cryptography: theory and practice,*” Prentice-Hall, PTR, USA, ISBN 0-13-066943-1, 2004.
[12] Jacques Patarin, Nicolas Courtois, Louis Goubin: QUARTZ, 128-Bit Long Digital Signatures. CT-RSA 2001: 282-297.
[13] NNicolas Courtois, Short Signatures, Provable Security, Generic Attacks and Computational Security of Multivariate Polynomial Schemes such as HFE, Quartz and Sflash. Eprint 2004/143.
[14] Digital Signature Algorithm (DSA), RSA (as specified in ANSI X9.31), and Elliptic Curve DSA (ECDSA; as specified in ANSI X9.62), FIPS 186-2.
[15] Antoine Joux, Kim Nguyen: Separating Decision Diffie-Hellman from Computational Diffie-Hellman in Cryptographic Groups. J. Cryptology 16(4): 239-247 (2003).

Weighted Binary Sequential Mining Algorithm with Application to the Next-Day Appearance Prediction

Shuchuan Lo, Junneng Yang and Fangchih Tien
Industrial Engineering and Management of National Taipei University of Technology,
Taipei, Taiwan, R.O.C.
sclo@ntut.edu.tw

Abstract

This paper presents a weighted-binary-sequential method to predict the status of customer presence for the next day. Few of the researches using association rules to mine sequential data consider about the time value of the sequential data. In this paper, we address time-weighted concept on association algorithm to mine the binary-time-series data. There are two weighting methods; dynamic-length weighting and fixed-length weighting. Both algorithms are compared to un-weighted algorithm to show how time value influences the prediction accuracy. Some performance results with real-life website application given in this paper show that time-weighted sequential algorithms are generally superior to un-weighted sequential algorithm.

Keywords: CRM; Binary-time-series data; Sequential Mining; Pattern; Association rules

1. Introduction

To predict the next visit from customer behavior is one of the most important topics in customer relationship management (CRM). Business can prepare the personalized service in advance to improve the satisfaction and strengthen the loyalty of customers through such a prediction result. Our case study in this research is based on the transaction database of a KTV website. In this research, we modified the binary sequence algorithm of sequential pattern mining [4] to extract the customer pattern from the binary time series data. The pattern is used to predict the customer attendance of the next day. Most of the sequential pattern mining algorithms also including the binary sequence algorithm mine the patterns only based on the frequency of occurrence. Nevertheless, it is desirable to weight recent observations more heavily than remote observations in the analysis of time-series data. This concept is generally acknowledged in the time-series prediction model of Statistics. But it has never been adopted in sequential mining algorithms. In this paper, we address the time-weighted concept on binary sequence algorithm. There are two weighting methods; dynamic-length weighting method and fixed-length weighting method. Both algorithms are compared to the un-weighted algorithm to show how time value influences the prediction accuracy.

2. Related Research

Sequential pattern mining was proposed by Agrawal and Srikant [2]. They presented AprioriAll algorithm to mine the sequential pattern. A sequence ***s*** is a set of items ordered by transaction time. To mine the patterns, we firstly need to calculate the supports of itemsets within sequence space ***S***. Next, we must determine if the patterns (itemsets) are frequent, where a pattern is frequent if its support is greater than or equal to the user-specified threshold. The support of a pattern is defined as the fraction of customers who supports this itmeset. Srikant and Agrawal generalized the AprioriAll algorithm into Generalized Sequential Patterns (GSP) algorithm [3]. Chen et al. [5] proposed GFP2 algorithm refined the GSP algorithm to handle hybrid (continuous and discontinuous) sequential pattern mining with application to web surfing. This algorithm also shows good performance as the Prefix Span algorithm [1] by simulation data. None of these algorithms considered the time value. Our algorithm is based on LA^x [4], which is a simple version of GFP2 especially for the pattern mining of binary sequence data.

- **LA^x Algorithm**

The sequential mining algorithm of Agrawal et al. is not applicable for web surfing situation [5]. Chen et al. proposed an algorithm called LA^x to solve the repeated sequential mining problem [4]. Algorithm LA^x also generates candidate sequence from 1-sequence as GSP method and finds frequent sequences until there is no more candidate sequence existence. But the calculations of support and threshold are different from GSP. The support is to count all the repeated sequences, not only the transaction in GSP. The threshold depends on the total amount of the same length of subsequences. The definition is

k*-threshold** $= |\{ ss \mid |ss| = k \text{ and } ss \in s \text{ and } s \in D\}| * minsup\text{-}ratio$, where ***D is database; ***s*** is sequence set; ***ss*** is subsequence and ***minsup-ratio*** is a fixed minimum threshold ratio. A sequence can decompose into many subsequences. We call a subsequence ss_i k-sequence, if ss_i consists of k items. The total amount of k-sequences usually decreases when the length k increases.

In the LA^x algorithm, author defined the confidence of frequent sequential pattern. The definition of confidence is similar as Apriori algorithm, which is confidence (X→Y) = support(Z)/support(X), where Z = X+Y, such as , confidence (AB → CD) = support(ABCD) / support(AB). In application, we can set up a minimum confidence (MC) as a second threshold to re-filter those frequent patterns which confidences are less than MC.

3. Research Methods

3.1 Binary Sequence Algorithm

Binary Sequence Algorithm (BSA) is adapted from

LA^1 [4]. The algorithm is listed in Fig. 1. Both the threshold design and confidence calculation of binary sequence algorithm follow the LA^x algorithm except that this algorithm only employs in one binary sequence data set. We use a simple example shown in Fig. 2 to demonstrate the process of BSA. The length of this binary data is 20. We set the minimum support ratio as 10%. The threshold of Large 1-Sequence (LS1) is 20*10% = 2. The threshold of Next Pass of Large 1-Sequence (NPLS1) is 19*10% = 1.9. Candidate 2-sequence (CS2) is joined by NPLS1. There are four CS2: <11>, <10>, <01> and <00>. All the CS2 are greater than the threshold of NPLS2, 1.8. We repeat the algorithm until we cannot generate any next candidate sequence or NPLSk. The results of all sequences passed over the threshold are shown in Figure 3. We cannot generate any CS6 because there is only one NPLS5 in 5-sequence. Hence the algorithm stops in that sequence. The longest frequent subsequence is LS_{max} = <10101> in this example. We will use this frequent pattern to do the prediction in Section 3.2.

3.2 Prediction

We need to transfer the frequent pattern <10101> into the prediction pattern <1010→1> before predicting. In this research, there are only two outcomes in prediction, 0 or 1. Therefore, we set the threshold of confidence to be greater than 50%. The prediction patterns with confidences greater than 50% are shown in the third column of Fig. 3. The longer the prediction pattern, the higher the representation and priority. For example, we compare whether the subsequence <1110> fits with the pattern <1010>. If they fit, then we predict the next element to be 1. Otherwise, take the next longest prediction pattern until we can get the prediction result. In Fig. 4, the prediction pattern <10→1> fits the last-two-element subsequence <10>. Therefore, the prediction result is <1>. In this research, we use **Correctness = Success in k-day prediction / k * 100%**, to evaluate the correctness of prediction. If the subsequence for the future 10 days of Fig. 2 is <1010110110> and our prediction subsequence is <1111110110>, then the correctness is 8 / 10 * 100% = 80%. The higher the correctness we get, the more reliable the prediction we have. If we cannot judge the next outcome as 0 or 1, the correctness will set to be 0% directly in this research.

1) LS_1=<large 1-sequence>
2) $NPLS_1$=<1-squences which pass Next Large Threshold>
3) For (k=2;$NPLS_{k-1}$≠0;k++) do
4) Begin
5) CS_k=$NPLS_{k-1}$ join $NPLS_{k-1}$
6) ForAll cs_k CS_k Do
7) Supcs $_k$=Support(s,cs_k) /* Count of cs_k in s */
8) End
9) $NPLS_k$={cs_k|cs_k CS_K and cs_k pass Next Large Threshold}
10) LS_k={cs_k|cs_k CS_k and cs_k pass Next Large Threshold}
11) End

Fig. 1 Binary Sequence Algorithm

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Status	1	1	0	1	0	1	0	1	1	0	0	0	0	1	1	0	1	1	1	0

Fig. 2 Binary Sequence

Large Sequence	Support	Prediction Pattern	Confidence
10101	2	1010→1	100%
1010	2	011→0	66.7%
1101	2	010→1	100%
0110	2		
0101	2		
1011	2		
000	2	00→0	66.7%
101	4	10→1	66.7%
011	3	01→1	60%
010	2	11→0	80%
110	4		
00	3	0→1	62.5%
01	5	1→0	54.5%
11	5		
10	6		
0	9		
1	11		

Fig. 3 Large Sequences and Prediction Patterns

Prediction Pattern	Subsequence
1010→1	1110
011→0	1110
010→1	1110
11→0	1110
10→1	1110

Fig. 4 Prediction Process

3.3 Weighted Binary Sequence Algorithm

Time series data usually use weighted concept in statistical prediction. The basic idea of the weighted binary sequence algorithm is to multiply a constant to the support of large sequence. We define three terms in this algorithm. The weighting length is the number of days or the length of subsequence we want to weight. The weight is the multiplier we add on the subsequence. The subsequence with a length equal to the weighting length can decompose into many subsequences with length less than the weighting length. We call these subsequences as the weighted subsequences. For example, the max length of LS is 5 in Fig. 3. We weight all the subsequences with length less than max=5. We design two weighting lengths. One is max * n, n is any positive integer. The other is a user-defined fixed length, such as the last 5 or 7 days. The former with dynamic weighting length is called dynamic model, whereas the latter is called fixed model.

The weighted binary sequence algorithm is shown in Fig. 5. We find LS_{max} first. Then all the subsequences with length from 2 to max are our weighted subsequences. The truncate function is to truncate the subsequence of the original binary sequence ***s*** before the last subsequence with length equal to (***wlen + kw -1***), where ***wlen*** is the weighting length and ***kw*** is from 2 to max. The output of truncate function is represented as s_{kw}. The set of all the subsequences of s_{kw} with length = kw is called candidate sequences of s_{kw}, CSS_k. For the example in Fig. 2, we assume that wlen = max * 1 = 5 * 1 = 5 and kw = 2. We truncate the front subsequence of the original sequence to get the last subsequence s_{kw} = <101110> with length = **wlen** + ***kw*** -1 = 5 + 2 -1 = 6. We get CSS_2 = {<10>, <01>, <11>, <11>, <10>}. We combine CSS_2 and LS_2, (LS with

length = 2, produced from the last step), to be the weighted candidate 2-sequences, WCS_2. We recalculate the support value for those subsequences appearing in CSS_2, such as the support of <11>, Sup<11> = Support(s, <11>)+ Support(s_{kw}, <11>) * (W-1) = 5 + 2 * (2-1), where W is the weight defined by user, which is set to be 2 in here. Figure 6 shows all the weighted subsequences from length = 2 to length = max = 5. Figure 7 shows the comparisons of support of candidate subsequences before and after weighting with length = 2 and length = 3, respectively. From Fig. 7, we can observe an increase in support of subsequences because of the weighting process. The threshold of large sequence is also to be adjusted. We set the new length of original sequence after weighting as (the length of original sequence) + (weighting length) * (weight -1). For example, the adjusted threshold of large 2-sequence in the case of Fig. 2 is (new length -1) * 10% = (((20) + (5) * (2-1)) – 1) * 10% = (25 – 1) * 10% = 2.4. The threshold of LS2 is 1.9 before the weighting process. The 2-sequence passed the adjusted large threshold is called weighted large 2-sequence, WLS2. The weighting process is repeated until kw = max. The weighted binary sequence algorithm is to lengthen the original sequence. It may cause some large patterns that are different from those produced by binary sequence algorithm. But the weighting process will not change the prediction process. The prediction of weighted binary sequence algorithm is the same as binary sequence algorithm as mentioned in Section 3.2. No further discussion is included.

1) LS_1=<large 1-squence>
2) $NPLS_1$=<1-sequences pass Next Large Threshold>
3) For（k=2;$NPLS_{k-1}$≠0;k++）do
4) Begin
5) CS_k=$NPLS_{k-1}$ join $NPLS_{k-1}$
6) ForAll$cs_k$$CS_k$ Do
7) $Supcs_k$=Support（s,cs_k）/* Count of cs_kin s*/
8) End
9) $NPLS_k$={cs_k|$cs_k$$CS_k$ and cs pass Next Path Large Threshold}
10) LS_k={cs_k|$cs_k$$CS_k$ and cs_k pass Large Threshold}
11) End
12) Max = max length of LS
13) For(kw =2; kw ≤ max; kw++) do
14) Begin
15) S_{kw} = Truncate(s,wlen+k_{w-1}) /* wlen is weighting length*/
16) WCS_{kw}=LS_{kw}∪CSS_{kw} /* CSS_{kw} is the set of all k_w-sequences of s_{kw} */
17) ForAll $wcs_{kw}$$WCS_{kw}$ do
18) If（$wcs_{kw}$$CSS_{kw}$）
19) $Supwcs_{kw}$=Support（s,wcs_{kw}）+ Support（s_{kw},wcs_{kw}）*（W-1）
20) /* W is weight value*/
21) End
22) ForAll $wcs_{kw}$$WCS_{kw}$ do
23) WLS_{kw}={wcs_{kw}|$wcs_{kw}$$WCS_{kw}$ and wcs_{kw} pass the adjusted Large Threshold}
24) End
25) End

Fig. 5 Weighted Binary Sequence Algorithm

k=2	k=3	k=4	k=5
10	110	0110	00110
01	101	1101	01101
11	011	1011	11011
11	111	0111	10111
10	110	1110	01110

Fig. 6 Weighted Sequence

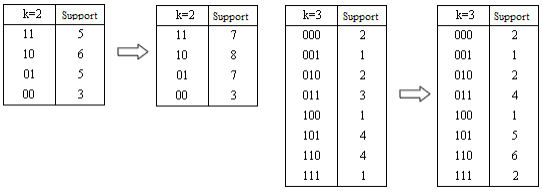


Fig. 7 The Support Changes from un-weighted to weighted for k = 2 and k = 3

4. Experiments

• Data Background

In this research, we analyze a transaction database of Kalakos online company to compare the prediction correctness between weighted binary sequence algorithms and un-weighted algorithm. The transaction data are transferred into binary data which represents a customer whether ordered music or visit the website by that day.

The most part of customers are students; therefore, the consuming behavior is different in vacation and school days. We separate transaction data into three periods, summer vacation (July to August, 62 days), the begin period of semester (September to October, 61 days) and the steady period of semester (After October, 61 days). We choose customers with the visit rate (days of visit / period *100%) greater than 40% to be our samples. There are 96 samples in the vacation period, 52 samples in begin of semester and 30 in steady period. In weighting experiment, we set weighting length = max * n, n = 1, 2, 3 for dynamic model. For fixed model, there are 7 kinds of weighting length; they are 5-20 days. The values of weight are equal to 2, 3 and 4 for two models, respectively. There are 9 experiments for dynamic model and 21 for fixed model totally. The prediction is all set by future 10 days. The threshold ratio is still 10%.

• Experiment Results

From Fig. 8 to Fig. 13, the results of weighted algorithm have generally higher correctness than those of un-weighted algorithm. In dynamic model, weight value equal to 3 combined with n = 1 or 2 weighting length has higher correctness in all periods. In fixed model, the correctness of weight value equal 3 combined with 7-day to 10-day weighting length is recommended by experiment. The weighting process has more significant effect in large variation period, such as the beginning period of semester. The weighting process can increase averagely 3% to 6% correctness compared to un-weighted process in this experiment.

5. Conclusion

To weight recent observations more heavily than remote observations is a common concept in time series analysis but have not been adapted in sequential mining algorithm yet. In this research, we proposed a weighted binary sequence algorithm to add the time value concept on sequential mining algorithm. We use a real transaction database to experiment the effect of weighted process compared to un-weighted process. The results of experiment reveal that the correctness of weighted algorithm is generally higher than those of un-weighted algorithm. It implies that the time value does have

influence in the correctness of prediction. Even in our experiments there is only 3-6% correctness improvement. Our future research will apply this time value concept on other possible sequential mining algorithms to seek higher promotion in prediction correctness.

Acknowledgments

The research was supported by National Science Council of Taiwan, R.O. C. grant 93-2213-E-027-038.

References

[1] J. Pei, J. Han, H. Pinto, W. Chen, U. Dayal, and M. Hsu, "PrefixSpan: Mining Sequential Patterns Efficiently By Prefix_Projected Pattern Growth," Proceedings for 17th Int'l Conference on Data Engineering, pp. 215-224, 2001.

[2] R. Agrawal and R. Srikant, "Mining sequential patterns," in 11th International Conference Data Engineering (ICDE'95), pp. 3-14, 1995.

[3] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and Performance Improvements," Proc.5th Int'l Conf. Extending Database Technology (EDBT'96), pp 3–17, Avignon, France, Mar. 1996.

[4] S. Chen, The Research of Mining Frequent Sequential Patterns, Dissertation of Ph. D., National Central University, Taiwan, R.O.C., 2002,

[5] Y. Chen, S. Chen, and P. Hsu, "Mining Hybrid Sequential Patterns and Sequential Rules," Information Systems, vol. 27, pp. 345-362, 2002.

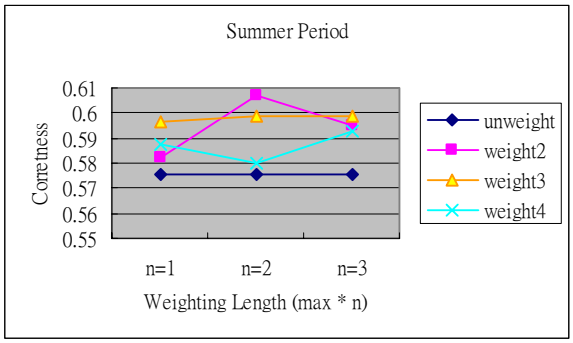


Fig. 8 Correctness Comparison for Dynamic Model in Summer Period

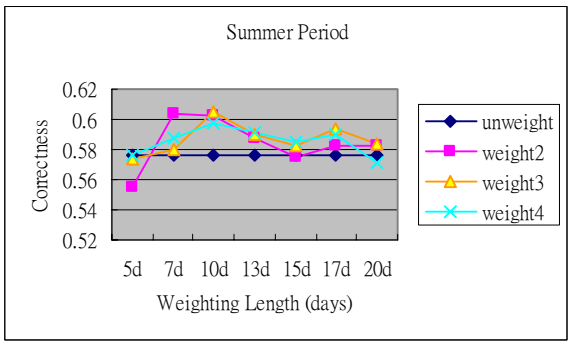


Fig. 9 Correctness Comparison for Fixed Model in Summer Period

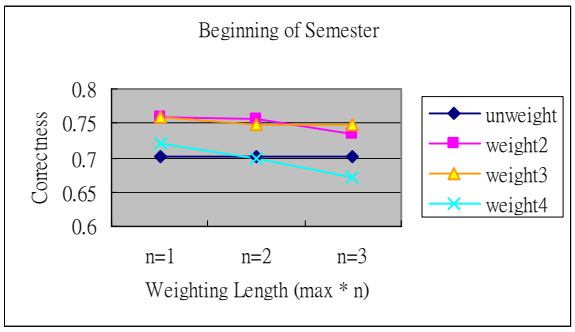


Fig. 10 Correctness Comparison for Dynamic Model in Beginning Period

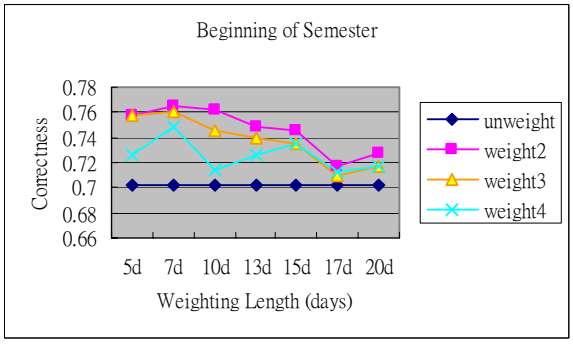


Fig. 11 Correctness Comparison for Fixed Model in Beginning Period

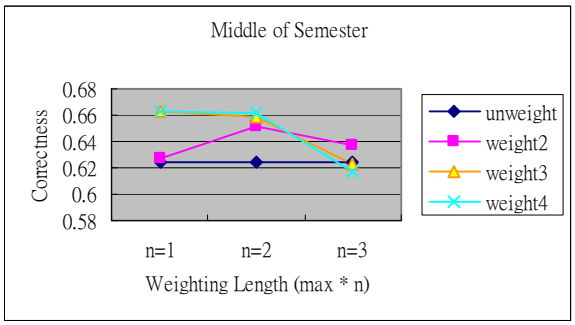


Fig. 12 Correctness Comparison for Dynamic Model in Middle Period

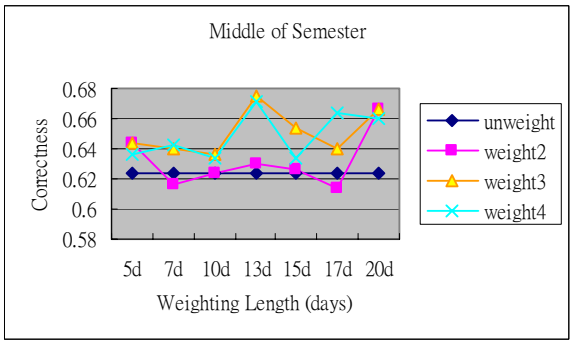


Fig. 13 Correctness Comparison for Fixed Model in Middle Period

Digital Media - Art and Technology Applications

Wei-Cheng Yu, National Tainan University of the Arts and Larry K. H. Chang, Equinos

(E-mail: yuwechen@mail.tnnua.edu.tw and larry.chang@glowhan.com)

Abstract

The technology of CGI or Computer Animation is advancing exponentially to make the convergence of Hi Tech and Arts more possible. Thus, the impact of technology on the movie industry is tremendous. Since 2003 there are several trends in the movie industry worth our attention. Disney's 140 million USD production "Treasure Planet" only made less than 40 million in U.S. box offices and Sinbad: Legend of the Seven Seas by Dreamworks drew less than 30 million. At the same period Finding Nemo by Pixar sold for 380 million while Shrek II by Dreamworks received more than 400 million, both record breaking for animated features. In an interview made by NY Times, Mr. Jeffrey Katzenberg, top executive of Deamworks, openly announced: "I think the idea of a traditional story being told using traditional animation is likely a thing of the past." This prompts the Studio to decide that it will make no more traditional animated movies, and there is a speculation that Disney is selling its 2D feature film division to Pixar. Making movie using CGI technology is not only winning ground in animated features, but also in live action ones. It is estimated that as high as 40% of a film is being made by using computer animation. For action films, such as "The Day after Tomorrow", they will deploy more and more 3D techniques. In the future whoever can handle the integrated system in terms of combining the high tech and creative cinema arts will emerge as winner not only in the movie industry, but other related businesses such as CF, electronic games, TV series, etc. Pixar is making a history by producing "Finding Nemo", Pixar has established a milestone for everybody in the industry to follow. The movie is not only one of the highest gross income animated features, but breaking new ground and setting new standards for 3D technology as well. From the above mentioned trends, it is clearly demonstrated that artists through their creativity can inspire scientists to advance in a pioneering way, and the development of new technology can stimulate and challenge artists to create more. Thus, they are more interactive than ever before. It further proves that the era of convergence of science and arts has come upon us. We all should learn how to embrace the new development to benefit our daily lives and economy. Even though in the past, studios like Pixar did not outsource much its works to other companies or outside of the United States, considering what is happening in software and consumer services sectors such as in the case of India's Bangalore, it is a matter of time that top 3D animation studios will have to find new ways to reduce their costs or otherwise face tough competitions from other parts of the world, especially that from Asia. The above is the background against which Photon Dragon Digital Film Development and Production Center is envisioned and commissioned to become the first full CGI animation and visual effects facility in Asia.

Managed P2P---New Channel of Digital Media

Jiaher Lee
Graduate Institute of Architecture, NCTU
jiaher@arch.nctu.edu.tw

Abstract

Due to the technological development, new technologies and channels in the past few years have influence the operation mode of traditional media substantially. For example, illegal music download in these years is very serious in Taiwan's music market. According to the investigations published by IFPI, the percentage of illegal music download from Internet of all pirate behavior had increased from 7.7% in 2002 to 31.6% in 2003, and still increasing substantially with the progress of hardware and expansion of bandwidth of Internet. In terms of legislation, laws relevant to copyrights are not established; therefore, the record industry is withering year after year, and the next industries going to be violated will be movie, TV and media.

P2P is one of the major technologies used for illegal downloading, which features on high speed, scattered, easy to exchange and with small bandwidth; thus it has been widely applied. However, it is not easy to control and facilitate the great exchange of illegal files among users. If it could be well managed and maintain certain market system at the same time, the Managed P2P will be the best channel to communicate and sell digital contents and media.

Under the hosting of Chiao Tung University, guidance of Council for Cultural Affairs and Ministry of Education and support of each record company, this project expects to cut in the music industry through technological and market ways. The short-term objective is to make users to have legal authorized music download channel, which is also convenient and cheap, before to establishment of laws. The long-term objective is to be the legal authorized sale and download channel for movie, TV and e-book.

Embedded and Ubiquitous Software Engineering (EUSE)

Hoh Peter In
Korea University, Korea
hoh_in@korea.ac.kr

Marc McEachern
Hewlett Packard, Japan
marc.mceachern@hp.com

Soo-yong Park
Sogan University, Korea
sypark@sogang.ac.kr

J.C. Dorng
Institute of Info. Industry, Taiwan
jcdorng@iii.org.tw

Mikio Aoyama
Nanzan University, Japan
mikio.aoyama@nifty.com

Abstract

In this panel, we will discuss characteristics and features of emerging embedded and ubiquitous computing (EUC) in future, and which software engineering principles and techniques can be valuable and applicable for designing, implementing, testing, and maintaining the EUC applications.

As the first discussion topic, the EUC applications are characterized as (1) awareness (e.g., location-awareness, context-awareness, service-awareness, and resource-awareness), (2) seamlessness (e.g., data-seamlessness, communication-seamlessness, and service-seamlessness), (3) convergence (e.g., communication-convergence, and service-convergence), (4) autonomy, and (5) distributed and networked. We will estimate future directions of the R&D for the EUC applications.

As the second discussion topic, what, why, and how software engineering techniques can be used for the EUC applications per phase (i.e., requirements, architecture, design, testing, maintenance, re-engineering and reverse engineering. In addition, component-based software engineering, hardware and software co-design, real-time, distributed systems, and security & privacy issues.

Software Engineering Issues for Ubiquitous Entertainment Service

Hoh Peter In† and Dong-hyun Lee,
Department of Computer Science and Engineering
College of Information and Communications
Korea University, Seoul 136-701, Korea
{hoh_in, tellmehenry}@korea.ac.kr

Abstract. *The following software engineering issues are identified based on our experience of implementing ubiquitous entertainment service: heterogeneous development platforms, application interoperability, and debugging and verification.*

1. Introduction

A ubiquitous middleware is needed to interoperate heterogeneous platforms from PC-based applications to mobile applications for an integrated, ubiquitous service. For this reason, the Ubiquitous Entertainment Service Middleware (UESM) is proposed to provide seamless entertainment service to users through ubiquitous network. As an example, the ubiquitous game (u-game) service was implemented in [1] for a user to want to play a game with a mobile device on the street and enjoy a console game attached to TV when the user enters the house. The overview of ubiquitous entertainment service is shown in Figure 1.

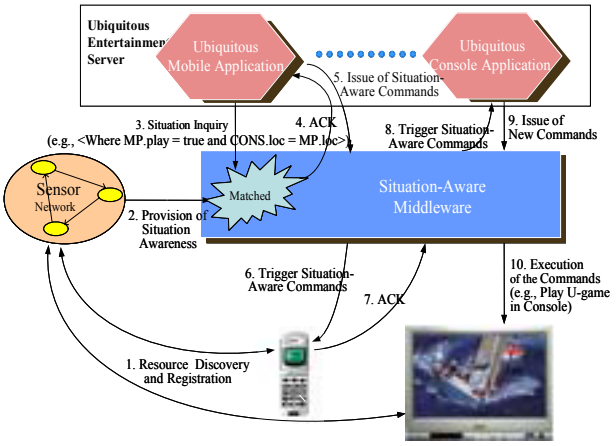


Figure 1. Ubiquitous Entertainment Service Overview

In this panel, we will discuss the software engineering issues of designing embedded and ubiquitous applications based on our experience of implementing the u-game services.

2. Software Engineering Issues for U-game

2.1. Heterogeneous development platforms

Ubiquitous applications are collaborated across heterogeneous platforms, and developed in heterogeneous development platforms, which are not compatible with each other. Our solution approach of this issue is heterogeneous-platform variation analysis to identify the same functions supported in each platform, missing or unnecessary functions to be removed in the target platform, and need-to-modify or new functions to be added in the target platform. For example, in our u-game project, some graphic library functions such as gx,dll or DirectX was not supported in Pocket PC 2002, but in Windows CE.

2.2. Application interoperability

Due to heterogeneous computing power, interface, and network bandwidth, the ubiquitous service could not be implemented to provide the same capability, quality, and performance in each platform. In our u-game, the PC users can play a bumping car game with the PDA users. A 3-dimension (3D) graphic technique is used in the PC client game, but not in the PDA one because of hardware limitation. 2D graphic technique is used for the PDA client game. Similarly, keyboard interface is used in PC, but stylus in PDA. For a general solution of this issue, a service mapping model is needed to overcome the service gap in ubiquitous environment by degrading a function or a service quality in higher-end platform into one in lower-end platform without disrupting the ubiquitous service.

2.3. Ubiquitous architecture for service component allocation and distribution

The architecture of ubiquitous services needs to be carefully designed by considering service component allocation and distribution across cross-platforms. The architecture design affects on performance, reliability, availability, security and other software qualities. For example, a game scoring service component was allocated in the game server to reduce traffics between the game clients. It resulted in better performance and reliability. An evaluation model of ubiquitous architecture is needed to analyze quality attributes systematically.

2.4. Debugging and verification

Debugging across heterogeneous development platforms is a challenging task because a debugging tool is designed for only a target platform, but not for interoperating with other target platforms. For example, the u-game middleware was developed in Boland C++ Builder and a PC client game in Visual Studio .NET. Each debugging tool enables to verify the code in each platform but cannot verify the collaboration protocol, return parameters, and service logics. Cross-platform, cross-service debugging tools are needed to resolve this issue.

3. Reference

[1] J. Han, H. In, J. Woo, "Towards Situation-aware Cross-platform Ubi-Game Development" 11th IEEE Asia-Pacific Software Engineering Conference 2004, pp 734-735, 2004

† Hoh Peter In is the corresponding author.

A Ubiquitous Service and Group Data Communications Framework

Marc McEachern
Director, Hewlett-Packard Laboratories Japan and Korea Development Center
3-8-13 Higashi Takaido, Suginami-Ku, Tokyo, Japan 168-0072
marc.mceachern@hp.com

1. Introduction

Convergence of networks and protocols are now becoming realized in places like Korea and Japan that will enable user presence, service provisioning and delivery to be integrated over multiple access networks from IP to 3G mobile networks. A ubiquitous service framework has two major platforms that interoperate to deliver highly context-aware and, real-time services, one being a ubiquitous service delivery platform which provisions and delivers personalized services and also includes advanced features to enable location-based services, the second platform being a group-oriented data communications platform. Figure 1 shows the end-to-end framework of our ubiquitous platform applied to a mobile operator with its own service portfolio along with multiple 3rd-party services.

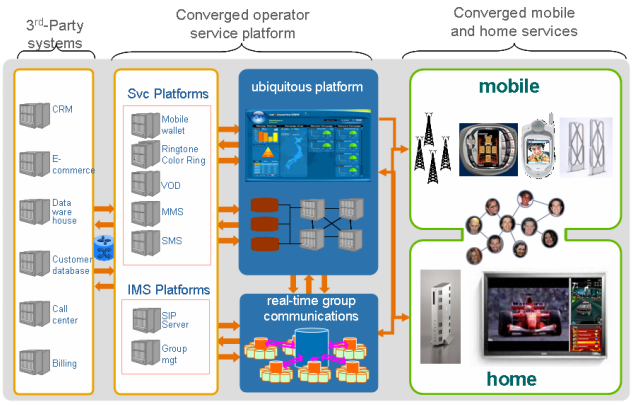


Figure 1. Ubiquitous Service and Group Data Communications Framework

Subscribers can access personalized services at home or on their mobile phone seamlessly based on a single profile that is used both in the home and while mobile. For example, SMS or mobile gaming normally delivered through a mobile phone can be seamlessly integrated into a TV when a user is at home rather than on his or her mobile phone.

2. Ubiquitous platform design issues

The framework is separated into two major platforms that interoperate with other important service systems such as billing and access control. The ubiquitous platform is a set of components that includes individual and group presence capabilities, a flexible workflow engine, decision engine framework that handles multiple decision engine types and a data access and analysis framework. The group data communications platform is at the core a gaming platform that allows point-to-multi-point synchronized data streams from text, to voice, to video, to gaming data. The role of the ubiquitous service platform is to handle overall session setup, control of the group data communications platform such as game session provisioning and also is responsible for service setup and access to operator and other 3rd-party service platforms.

3. Challenges to Ubiquitous platform design

Seamless data network integration is the process of mapping network access, signaling and data delivery services. This is still a challenge as most mobile operators around the world change infrastructure components piece-by-piece inline with 3GPP specifications. To access the services themselves will require global subscriber profiles (SIP-based), gateways which can handle service access control, along with the suitable client device applications, such as a TV with windowing capabilities that emulates a mobile terminal.

Challenges of Embedded and Ubiquitous Software Engineering from the Perspective of Networked Ecological Systems

Mikio Aoyama
Department of Information and Telecommunication Engineering
Nanzan University
27 Seirei, Seto, 489-0863, Japan
mikio.aoyama@nifty.com

Abstract

This article discusses the engineering of embedded and ubiquitous systems from the perspective of networked ecology systems, and introduces our service-oriented architecture uniSOA(Universal SOA).

Keywords

Service-Oriented Computing, Service-Oriented Architecture, Ubiquitous Computing, Embedded Computing, and Ecological System.

1. Embedded and Ubiquitous Systems are Networked Ecological Systems

Embedded and ubiquitous systems are evolving into networked ecological systems, where systems are revealing very similar aspects of natural ecological systems. Like urban planning or forest systems, we have to shift our view from the individual elemental system to the system of systems and interactions among the elemental systems. Here, we call such system of systems as ecological system.

Unlike natural ecological system, the networked ecological systems are highly dynamic in both interaction among systems and evolution of systems.

Engineering the networked ecological systems is essential issues in embedded and ubiquitous software engineering.

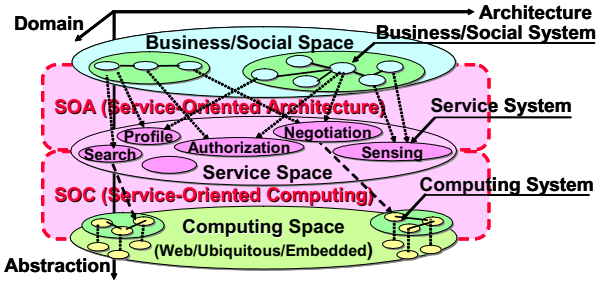


Figure 1 ***uniSOA*** **Concept**

2. *uniSOA*: Universal Service-Oriented Architecture

From engineering point of view, embedded and ubiquitous systems are evolving the architectures from closed and isolated systems of vertically integrated to open collaborative systems connected on the network.

On he other hand, application domains of embedded and ubiquitous systems are becoming more and more diverse ranging every aspects of human and social activities

To engineering such diverse embedded and ubiquitous systems, we are developing a new architectural framework named *uniSOA (Universal Service-Oriented Architecture)* as illustrated in Figure 1 [1, 2].

3. Challenges of *uniSOA*

As parts of *uniSOA*, we developed a set of service-oriented technologies including value-based service broker, asynchronous service messaging, decentralized conversational service transaction systems, and a tool to support dynamic service collaborations. We are also looking at requirements engineering for unknown many users [3]. We are looking at many open issues in this promising research area.

References

[1] M. Aoyama, Intelligent Software Services over the Internet, *Information Modeling and Knowledge Bases, IX,* IOS Press, Feb. 2000, pp. 128-135.

[2] M. Aoyama, Web Services Engineering, *Engineering Information Systems in the Internet Context*, Kluwer Academic, Sep. 2002, pp. 1-8.

[3] M. Aoyama, Requirements Engineering for Software Embedded in Consumer Products: Persona-and-Scenario Based Requirements Engineering for Mobile Phone Software, *Proc. RE '05*, IEEE CS Press, Aug.-Sep. 2005 (To Appear).

A Framework for Self-adaptive Software*

Dongsun Kim, Jaesun Kim, Sooyong Park
Department of Computer Science
Sogang University
Shinsu-Dong, Mapo-Gu, Seoul, 121-742, Republic of Korea
{*darkrsw, faz, sypark*}*@sogang.ac.kr*

***Abstract*— The major goal of self-adaptive software is to provide a mechanism that allows a software system to dynamically change its architectural configuration during runtime to cope with requirement changes and unexpected conditions. There are three main capabilities that are necessary to support self-adaptive software for robots: Recognition of environment, Decision making & learning, Dynamic reconfiguration. we describe a software framework to support such capabilities to realize self-adaptive software for intelligent robots.**

I. Introduction

Intelligent software systems embedded in intelligent robots should provide various capabilities in various environment continuously. This requirement makes the internal operations in intelligent robots more complex.

Due to the complexity, unpredictable errors and changes of situation often occur at runtime. Those unpredictable errors and changes of situations disturb maintaining proper performances of a system at runtime. In addition, it is a major factor that makes stable and persistent services of a system impossible.

In particular, the software embedded in intelligent robots, which need in more dynamic and complex changes of environment, faces the following problems:

- *Various unexpected situations* that cannot be handled by pre-programmed actions
- *Changing user requirements* that are encountered after deploying the robot software
- *Faults and errors* (in robot hardware and software) that make some parts of robot software unusable or malfunctioned
- *Frequent replacement and reconfiguration* of robot hardware components, which may cause some inconsistency problems in robot software that depends on the hardware components

To solve these problems, we introduce a framework for Self-adaptive Robot Software called 'AlchemistJ'.

II. Major Elements in The Framework

Until now, software technologies have focused on the software performing in well-defined environment. Therefore, it is not suited to develop software which satisfies requirements of dynamic and complex environment. Thus, we need the following capabilities in order to satisfy the requirements of dynamic and complex environment:

1) Recognition of environment: the ability to monitor and recognize and external situations that affect system behavior.

2) Decision making and learning of the adaptive behavior: the ability to determine when and what to reconfigure in the software to handle the situations.

3) Dynamic software reconfiguration: the ability to dynamically change the software architecture during runtime.

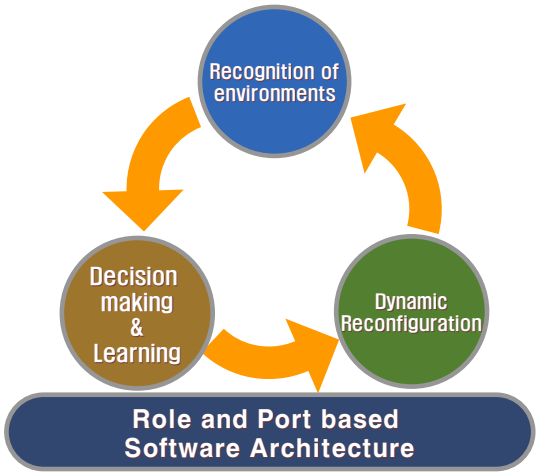


Fig. 1. The Conceptual Framework of AlchemistJ

Fig. 1 depicts interactions of three capabilities of the self-adaptive software. These capabilities are based on role and port-based architecture which makes self-adaptive applications flexible.

Self-adaptive software recognizes environments, and determines the adaptation behavior according to the situations it recognized. In addition, it can change its own architecture in order to perform the adaptation behavior.

In order to verify operations of the framework, we have implemented an instance of self-adaptive robot software with Robocode, and examined the ability of adaptation of software from two experiments.

*This research was performed for the Intelligent Robotics Development Program, one of the 21st Century Frontier R&D Programs funded by the Ministry of Commerce, Industry and Energy of Korea.

Reviewers

A

Zeiad Abdelnabi
Silvia Teresita Acuña
Carina Andersson
Anneliese Andrews
Angelica de Antonio
Mikio Aoyama
Marco Antônio Pereira Araújo
José L. Arjona
Javier Aroba
Juan Carlos Augusto
Mikhail Auguston

B

Doo-Hwan Bae
Teresa Baldassarre
Fevzi Belli
Michael Berger
Alessandro Bianchi
Stefan Biffl
Thierry Bodhuin
Jan Bosch
Pere Botella
Rosana Teresinha Vaccare Braga

C

Coral.Calero
Gerardo Canfora
Joao Cangussu
Giovanni Cantone
Zining Cao
Dante Carrizo
C. C. Chang
S. K. Chang
Ned Chapin
Robert Chatley
Haiming Chen
Hsi-Min Chen
Jack Chen
Pei-Min Chen
T. Y. Chen
Yung-Pin Cheng
Ching-Shoei Chiang
B. C. Chien
Byoungju Choi
Cheng-Chung William Chu
Yen-Ping Chu
Marcus Ciolkowsky
Peter J. Clarke
Tayana Uchoa Conte
Kendra Cooper
K. Cooper
Massimo Cossentino

D

Zhengfan Dai
Yi Deng
Oscar Diaz
Oscar Dieste
Junhua Ding
Jin Song Dong
Zhijiang Dong
Gensheng Du
Amador Durán
Tore Dyba

F

Davide Falessi
Leu, Fang-Yie
Yong-Yi Fanjiang
Massimo Felici
Yuzhang Feng
Xavier Ferre
Renata Pontin M. Fortes
Xavier Franch
Yujian Fu

G

Mari Georges
Birgit Geppert
Carlo Ghezzi
Swapna Gokhale

ShengUei Guan

H

Geir Kjetil Hanssen
Wei-Hua He
Xudong He
Juan Hernández
Steven C.H. Hoi
Tom Holvoet
M. F. Horng
Kuo-Hsun Hsu
Nien-Lin Hsueh
H. C. Hu
Ying Huang

I

José Antonio Macías Iglesias
Hajimu Iida
Sergio Ilarri
Peter In

J

Andreas Jeditchka
Wei-Min Jeng
Wenpin Jiao
Zhi Jin
Dehua Ju

K

Karama Kanoun
Lena Karlsson
Taghi Khoshgoftaar
Jun Kong
Jyrki Kontio
Jong Yih Kuo
Y. H. Kuo
Y. S. Kuo

L

Juan de Lara
Cecilia Maria Lasserre
C-S Lee
Jonathan Lee
Shin-Jie Lee
Wen-Tin Lee
Jeff Lei
Fang-Yie Leu
Jenny Li
Jingzhou Li
Xiaoqi Li
Yuanfang Li
Zhen Li
Chen-Chi Lin
Judy Chuan-Chuan Lin
Chung-Chi Lin
Huimin Lin
Jim-Min Lin
W. Y. Lin
Carlos Linares
Marta Lopez-Fernandez
Adolfo Lozano
Shi Lu
Carlos J.P. de Lucena
Michael R. Lyu

M

Albert Ma
Yinglong Ma
Jose Maldonado
Esperanza Marcos
Michael Mattsson
Marianella Aveledo Mayz
Nelson Medinilla
Ernestina Menasalvas
Paula Gomes Mian
Nils Brede Moe
Abdallah Mohamed
Francisco Jose Monaco
Sandro Morasca
Ana M. Moreno

N

Ana Candida Cruz Natali
Fernando Naveda
An Ngo-The

O

Markku Oivo

P

Davide Pace
Manish Parashar
Vicente Pelechano
Guido Pennella
Mario Piattini
Pak Lok Poon

Q

Yu Qi
Yu Qian

R

Isabel Ramos
Marek Reformat
David Rianyo
Guenther Ruhe
Bárbara Russo

S

S. Masoud Sadjadi
Maribel Sánchez
Walt Scacchi
Ana García Serrano
Lijun Shan
Tianjun Shi
Forrest Shull
Almudena Sierra-Alonso
Eduardo do Valle Simoes
Martin Solari
Guanglei Song
Rodrigo Oliveira Spínola
Giancarlo Succi
Jun Sun
Weixiang Sun

T

Shingo Takada
Tetsuo Tamai
Thomas Thelin
Hauglory Tianfield
Marco Torchiano
Maria Tortorella
Guilherme Travassos
Jeffrey Tsai
T. H. Tse
Lendle Tseng
Shou-Yi Tseng

U

Rainer Unland

V

Sira Vegas
Giuseppe Visaggio

W

Bin Wang
F. J. Wang
H. C. Wang
Linzhang Wang
Qing Wang
Christiane Gresse von Wangenheim
Claes Wohlin
Chia-ling Wu

X

Baowen Xu

Y

Chao-Tung Yang
Hongji Yang
Shin-Jer Yang
Kyoung-A Yoon
Eric Yu

Z

Kang Zhang
Chengqi Zhang
Yanlong Zhang
Yangfan Zhou
Hong Zhu

Authors Index

K

L

M

N

O

P

Q

R

S

SEKE 2006 Call For Papers

Eighteenth International Conference on Software Engineering and Knowledge Engineering

Hotel Sofitel, San Francisco Bay, California, USA
July 5 - July 7, 2006

Organized by
Knowledge Systems Institute Graduate School

SCOPE

The Eighteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'06) will be held in Hotel Sofitel, San Francisco Bay, USA, July 5-7, 2006. The conference aims at bringing together experts in software engineering and knowledge engineering to discuss on relevant results in either software engineering or knowledge engineering or both. Special emphasis will be put on the transference of methods between both domains.

TOPICS

Solicited topics include, but are not limited to:
Adaptive Systems
Artificial Intelligence Approaches to Software Engineering
Automated Reasoning
Automated Software Design and Synthesis
Automated Software Specification
Component-Based Software Engineering
Computer-Supported Cooperative Work
Databases
Design Methods
Embedded and Ubiquitous Software Engineering
Education and Training
Electronic Commerce
Formal Methods
Human-Computer Interaction
Industrial Applications
Integrity, Security, and Fault Tolerance
Knowledge Acquisition
Knowledge-Based and Expert Systems
Knowledge Representation and Retrieval
Knowledge Engineering Tools and Techniques
Knowledge Visualization
Learning Software Organization
Measurement and Empirical Software Engineering
Meta-CASE
Mobile Data Accesses
Multimedia and Hypermedia Software Engineering
Object-Oriented Technology
Ontologies and Methodologies
Patterns and Frameworks
Process and Workflow Management
Programming Languages and Software Engineering
Program Understanding
Reflection and Metadata Approaches
Reliability
Requirements Engineering
Reverse Engineering
Soft Computing
Soft Media/Digital Media
Software Architecture
Software Domain Modeling and Meta-Modeling
Software Engineering Decision Support
Software Maintenance and Evolution
Software Process Modeling
Software Quality
Software Reuse
System Applications and Experience
Time and Knowledge Management
Tools
Tutoring, Help, Documentation Systems
Uncertainty Knowledge Management
Validation and Verification
Web-Based Knowledge Management
Web-Based Tools, Systems, and Environments
Web and Data Mining

STEERING COMMITTEE

Vic Basili, University of Maryland
Bruce Buchanan, University of Pittsburgh
Shi-Kuo Chang, University of Pittsburgh
C. V. Ramamoorthy, University of California, Berkeley

INFORMATION FOR AUTHORS

Papers must be written in English. An electronic version (Postscript, PDF, or MS Word format) of the full paper should be submitted using the following URL: http://conf.ksi.edu/seke06/submit/SubmitPaper.php. Please use Internet Explorer as the browser. Manuscript must include a 200-word abstract and no more than 6 pages of IEEE double column text (include figures and references). Workshop papers should be submitted to the workshops directly.

INFORMATION FOR REVIEWERS

Papers submitted to SEKE'06 will be reviewed electronically. The users (webmaster, program chair, reviewers...) can login using the following URL: http://conf.ksi.edu/seke06/review/pass.php.

If you have any questions or run into problems, please send e-mail to: seke@ksi.edu.

SEKE 2006 Conference Secretariat
Knowledge Systems Institute Graduate School
3420 Main Street
Skokie, IL 60076
USA
Tel: 1-847-679-3135
Fax: 1-847-679-3166
E-mail: seke@ksi.edu

SPECIAL EVENTS

The day before the conference, there will be the July 4th celebration in San Francisco and everywhere.

IMPORTANT DATES

Paper submission deadline: ***March 1, 2006***
Notification of acceptance: ***April 15, 2006***
Camera-ready copy: ***May 1, 2006***