Distributing Responsibilities for Exception Handling in JaCaMo

Demonstration Track

Matteo Baldoni[1], Cristina Baroglio[1], Olivier Boissier[2], Roberto Micalizio[1], Stefano Tedeschi[1]
[1] Università di Torino, Dipartimento di Informatica, Italy
[2] Laboratoire Hubert Curien UMR CNRS 5516, Institut Henri Fayol, MINES Saint-Etienne, France

ABSTRACT

We present an extension of the organizational model and infrastructure adopted in JaCaMo, that explicitly encompasses the notion of exception. We propose an exception handling mechanism for organization management in multi-agent systems. This mechanism relies on abstractions that are seamlessly integrated with organizational concepts, such as responsibilities, goals and norms.

KEYWORDS

Exception Handling; Multiagent Organizations; JaCaMo

ACM Reference Format:
Matteo Baldoni[1], Cristina Baroglio[1], Olivier Boissier[2], Roberto Micalizio[1], Stefano Tedeschi[1]. 2021. Distributing Responsibilities for Exception Handling in JaCaMo: Demonstration Track. In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), Online, May 3–7, 2021*, IFAAMAS, 3 pages.

1 INTRODUCTION

A software is *robust* when it is able to keep an acceptable behavior in presence of abnormal execution conditions, like unavailability of system resources, communication failures, invalid or stressful inputs [13]. One specific mechanism that supports robustness is *exception handling* which, roughly speaking, amounts to equipping the system with the capabilities needed to tackle classes of abnormal situations, identified at design time. An *exception* is an "event that causes suspension of normal program execution" [1]. Therefore, the purpose of an exception handling mechanism is to provide the tools to (i) identify when an exception occurs, and (ii) apply suitable handlers, capable of treating the exception and recover.

The need of exceptions emerges from the desire of structuring and modularizing software, separating concerns into components that interact. Following the seminal work of Goodenough on exceptions in programming languages [14–16], exceptions permit the user of an operation to extend the operation's domain (the set of inputs for which effects are defined) or its range (the effects obtained when certain inputs are processed). They allow tailoring an operation's results or effects to the purpose in using the operation, and they also allow generalizing operations, making them usable in a wider variety of contexts than would otherwise be the case. Consequently, an exception's *full significance* is known only outside the detecting operation: the operation is not permitted to determine unilaterally what is to be done after an exception is raised. The invoker controls the response to the exception that is to be activated. This increases the generality of an operation because the appropriate "fixup" will vary from one use of the operation to the next. To this aim, an invoker must be given enough information about the failure. This differentiates exception handling from the simple detection of norm violation or like contrary-to-duties [28], which do not systematically encompass the responsibilities of raising and dealing with the exception, see [2–4, 8, 9].

Multiagent organizations (MAOs) are widely used for the design and development of distributed, non-centralized, autonomous systems. Key features of many organizational models (see e.g., [5–7, 10, 19]), are a functional decomposition of the organizational goal and a normative system. Norms shape the scope of the responsibilities that agents take when joining the organization, capturing what they should do to contribute to the achievement of the organizational goal [12, 26, 27]. In particular, they allow to coordinate the distributed execution notifying agents when something must be done to discharge their responsibilities as members of the organization. This demonstration shows how exception handling can be grafted inside the normative system of a MAO and, consequently, how the normative system can be used to gain robustness in the execution. Specifically, we introduced exception handling in the well-known JaCaMo multi-agent platform [6], integrating it both at a conceptual level, within the high-level abstractions that are provided by the model of JaCaMo's organizational component, and at a software level, by enriching its infrastructure.

2 RESPONSIBILITY IN EXCEPTION HANDLING

Multi-agent systems bring software structuring, modularization, and separation of concerns to an extreme. Here, autonomous agents cooperate and rely on one another to pursue their aims. However, exceptions handling (besides a few attempts) has never been applied as postulated in the work by Goodenough. What explained in the first paragraph brings forward two important aspects of exception handling. *First*, it always involves two parties: a party that is *responsible* for raising an exception, and another party that is *responsible* for handling it. *Second*, it captures the need for some information/account from the former to the latter that allows coping with the exception. Since MAOs, in essence, are built upon responsibilities, we claim that MAOs are naturally suited to encompass an exception handling mechanism. Indeed, in a MAO, each agent has only a partial view of the organizational goal, whose achievement is distributed among the agents. From this perspective, we can interpret an exception as an event which denotes the impossibility, for some agent, to fulfill one of its responsibilities – e.g. the failure in the achievement of a goal or a missed deadline. As a consequence, the distributed achievement of the organizational goal will be suspended.

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, Online.

In the context of MAO, we propose to leverage on responsibility not only to model the duties of the agents in relation to the organizational goal, but also to enable agents to report about exceptions, occurring within the organization operation, and to identify those agents entitled for handling them. When agents join an organization, they will be asked to take on the responsibilities not only for the organizational goals, but also: 1) for providing accounts about the context where exceptions are detected while pursuing organizational goals, and 2) if appointed, for handling such exceptions once the needed information is available. Responsibilities, thus, define the scope of the exceptions, expressed with respect to the organizational state, that agents ought to raise or treat.

Related works. Differently than the approach by Platon et al. [22–25], where exception handling is seen as a tool that the individual agent can activate internally, to preserve control on itself despite the occurrence of exceptions, the proposal leverages on the distributed nature of exception handling, typical of programming languages and of the actor model [17], and suited to distributed systems made of cooperative parties, like MAO. In [11, 20, 21], an approach is proposed, that is based on a shared exception handling service. The service provides *sentinels*, that are equipped with handlers (inspired by research on management), to be plugged into existing agent systems. The service actively looks for exceptions in the system and prescribes specific interventions from a body of general procedures. Sentinels communicate with agents using a predefined language for querying about exceptions and for describing exception resolution actions. Agents, for their part, are required to implement a minimal set of interfaces to report on their own behavior and modify their actions according to the prescriptions given by the sentinels. As a difference, our proposal seamlessly integrates exception handling into the agents themselves, without centralizing it. In this way, it accommodates Goodenough's recommendation that appropriate "fixup" will vary from one use of the operation to the next.

3 EXTENDING JACAMO

JaCaMo [6] is a conceptual model and programming platform that integrates agents, environments and organizations. Moise [18, 19] implements the organizational programming model. It comprises a *structural* dimension, a *functional* dimension, including a set of schemes that captures how the organizational goals are decomposed into subgoals, grouped into missions, and a *normative* dimension binding the other two. Agents, in fact, are held to explicitly commit to missions, i.e., taking responsibility for mission goals.

We propose to enrich the schemes and missions of the functional specification of a organization with the following new concepts[1]. **Recovery Strategy** encodes when and how a given exception is to be raised and handled within the organization. It includes a notification policy and one or more handling policies. **Notification Policy** specifies when the exception must be raised. It includes a throwing goal, enabled when such circumstances hold and an *exception type* encoding the kind of information to be produced. **Handling Policy** specifies a way in which the exception must be handled, once the needed information is available. It includes a catching goal and specifies when such goal is to be enabled. **Throwing Goal** denotes the organizational goal of raising the exception, i.e., it will make the agent that is responsible for it to provide the information that is needed for recovery. **Catching goal** captures the course of action to handle the exception and remediate. Its aim is to restore the execution of the scheme after an exception is raised. Throwing goals and catching goals specialize the goal specification and are incorporated into mission just like standard ones. As a result, missions become a tool to distribute the responsibilities, not only concerning the normal execution, but also for the management of exceptional situations. Policies, in turn, delimit the scope of such responsibilities, specifying when and how they are to be discharged.

To illustrate, we rely on the *building-a-house* example, originally introduced in [6]. Here, the organizational goal is to build a house on a plot in a dynamic environment, where robustness of the organization coordinating the building of the house is important. Achievement involves the coordination of multiple companies executing the various subgoals, part of which can be executed in parallel, while part depends on others. For instance, site preparation must be completed before any other step. Should the agent in charge of it face a failure in the fulfillment of its responsibility, the whole house construction could not proceed. We can, then, extend the functional specification of the organization with the following recovery strategy, targeting a failure in the achievement of goal `site_prepared`:

```
1  <recovery-strategy id="rs1">
2    <notification-policy id="np1">
3      <condition type="goal-failure">
4        <condition-argument id="target" value="site_prepared" />
5      </condition>
6      <exception-type id="site_preparation_exception">
7        <exception-argument id="errorCode" arity="1" />
8      </exception-type>
9      <goal id="notify_site_preparation_problem" />
10   </notification-policy>
11   <handling-policy id="hp1">
12     <condition type="always" />
13     <goal id="handle_site_problem">
14       <plan operator="parallel">
15         <goal id="inspect_site" />
16         <goal id="notify_affected_companies" />
17       </plan>
18     </goal>
19   </handling-policy>
20 </recovery-strategy>
```

Notification policy `np1` specifies that, should a goal failure concerning `site_prepared` occur (see the condition at Lines 3-5), the throwing goal `notify_site_preparation_problem` is to be enabled. Its purpose is to make the agent responsible for it provide the information needed for recovery. To this end, an exception type `site_preparation_exception` (Lines 6-8) specifying an `errorCode` is defined. Handling policy `hp1`, in turn, expresses what needs to be done to solve the site preparation exception, once it has been raised and the error code provided. In this case, the catching goal is a complex one (Lines 13-18): the site should be inspected and, at the same time, the other companies involved in the house construction notified. It's worth noting that agents in charge of these goal will likely leverage the information provided beforehand to achieve them. Site inspection, e.g., will be performed in different ways in case the error code denotes a flooding rather than the finding of archaeological remains. Corrective actions undertaken by the agents will be different, as well.

[1]The full code of Moise extended with exception handling, together with some examples, is available at http://di.unito.it/moiseexceptions.

REFERENCES

[1] 2010. ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary. *ISO/IEC/IEEE 24765:2010(E)* (Dec 2010), 1–418. https://doi.org/10.1109/IEEESTD.2010.5733835

[2] Matteo Baldoni, Cristina Baroglio, and Roberto Micalizio. 2020. Fragility and Robustness in Multiagent Systems. In *Post-Proc. of the 8th International Workshop on Engineering Multi-Agent Systems, EMAS 2020, Revised Selected Papers (LNAI, 12589)*, C. Baroglio, J. F. Hubner, and M. Winikoff (Eds.). Springer, Auckland, New Zealand, 61–77. https://doi.org/10.1007/978-3-030-66534-0_4

[3] Matteo Baldoni, Cristina Baroglio, Roberto Micalizio, and Stefano Tedeschi. 2020. Is Explanation the Real Key Factor for Innovation?. In *Proc. of Italian Workshop on Explainable Artificial Intelligence, XAI.it 2020*, C. Musto, D. Magazzeni, S. Ruggieri, and G. Semeraro (Eds.), Vol. 2742. CEUR, Workshop Proceedings, Online Event, Italy, 87–95. http://ceur-ws.org/Vol-2742/

[4] Matteo Baldoni, Cristina Baroglio, Roberto Micalizio, and Stefano Tedeschi. 2021. Robustness based on Accountability in Multiagent Organizations. In *Proc. of 20th International Conference on Autonomous Agents and Multiagent Systems, Engineering Multiagent Systems Track, AAMAS 2021*, U. Emdriss, A. Nowé, F. Dignum, and A. Lomuscio (Eds.). IFAAMAS, Richland, SC, Online.

[5] B. Bauer, J.P. Müller, and J. Odell. 2001. Agent UML: A formalism for specifying multiagent software systems. *Software Engineering and Knowledge Engineering* 11, 3 (2001), 207–230.

[6] Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, Alessandro Ricci, and Andrea Santi. 2013. Multi-agent Oriented Programming with JaCaMo. *Sci. Comput. Program.* 78, 6 (2013), 747–761. https://doi.org/10.1016/j.scico.2011.10.004

[7] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. 2004. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems* 8, 3 (2004), 203–236. https://doi.org/10.1023/B:AGNT.0000018806.20944.ef

[8] Amit K. Chopra and Munindar P. Singh. 2016. From social machines to social protocols: Software engineering foundations for sociotechnical systems. In *Proc. of the 25th Int. Conf. on WWW*.

[9] Amit K. Chopra and Munindar P. Singh. 2018. Sociotechnical Systems and Ethics in the Large. In *AIES '18: Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*. ACM, 48–53.

[10] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. 1993. Goal-directed requirements acquisition. *Science of Computer Programming* 20, 1 (1993), 3 – 50. https://doi.org/10.1016/0167-6423(93)90021-G

[11] Chrysanthos Dellarocas and Mark Klein. 2000. An experimental evaluation of domain-independent fault handling services in open multi-agent systems. In *Proceedings Fourth International Conference on MultiAgent Systems*. IEEE, 95–102.

[12] Christophe Feltus. 2014. *Aligning Access Rights to Governance Needs with the Responsability MetaModel (ReMMo) in the Frame of Enterprise Architecture*. Ph.D. Dissertation. University of Namur, Belgium.

[13] Jean-Claude Fernandez, Laurent Mounier, and Cyril Pachon. 2005. A Model-based Approach for Robustness Testing. In *Proceedings of the 17th IFIP TC6/WG 6.1 International Conference on Testing of Communicating Systems* (Montreal, Canada) *(TestCom'05)*. 333–348.

[14] John B. Goodenough. 1975. Exception Handling Design Issues. *SIGPLAN Not.* 10, 7 (July 1975), 41–45. https://doi.org/10.1145/987305.987313

[15] John B. Goodenough. 1975. Exception Handling: Issues and a Proposed Notation. *Commun. ACM* 18, 12 (Dec. 1975), 683–696. https://doi.org/10.1145/361227.361230

[16] John B. Goodenough. 1975. Structured Exception Handling. In *Proceedings of the 2nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (Palo Alto, California) *(POPL '75)*. Association for Computing Machinery, New York, NY, USA, 204–224. https://doi.org/10.1145/512976.512997

[17] Carl Hewitt, Peter Bishop, and Richard Steiger. 1973. A Universal Modular ACTOR Formalism for Artificial Intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence* (Stanford, USA) *(IJCAI'73)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 235–245.

[18] Jomi F. Hübner, Olivier Boissier, Rosine Kitio, and Alessandro Ricci. 2010. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems* 20, 3 (1 5 2010), 369–400.

[19] Jomi F. Hübner, Jaime S. Sichman, and Olivier Boissier. 2007. Developing Organised Multiagent Systems Using the MOISE+ Model: Programming Issues at the System and Agent Levels. *Int. J. Agent-Oriented Softw. Eng.* 1, 3/4 (2007), 370–395. https://doi.org/10.1504/IJAOSE.2007.016266

[20] M. Klein and Chrysanthos Dellarocas. 1999. Exception handling in agent systems. In *AGENTS '99*.

[21] Mark Klein and Chrysanthos Dellarocas. 2000. A knowledge-based approach to handling exceptions in workflow systems. *Computer Supported Cooperative Work (CSCW)* 9, 3-4 (2000), 399–412.

[22] Eric Platon. 2007. *Modeling exception management in multi-agent systems*. Ph.D. Dissertation. Université Pierre et Marie Curie, France.

[23] Eric Platon, Nicolas Sabouret, and Shinichi Honiden. 2007. Challenges for Exception Handling in Multi-Agent Systems. In *Software Engineering for Multi-Agent Systems V*, Ricardo Choren, Alessandro Garcia, Holger Giese, Ho-fung Leung, Carlos Lucena, and Alexander Romanovsky (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 41–56.

[24] Eric Platon, Nicolas Sabouret, and Shinichi Honiden. 2007. A Definition of Exceptions in Agent-Oriented Computing. In *Engineering Societies in the Agents World VII*, Gregory M. P. O'Hare, Alessandro Ricci, Michael J. O'Grady, and Oğuz Dikenelli (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 161–174.

[25] Eric Platon, Nicolas Sabouret, and Shinichi Honiden. 2008. An architecture for exception management in multiagent systems. *International Journal of Agent-Oriented Software Engineering* 2, 3 (2008), 267–289.

[26] Ian Sommerville. 2007. *Models for Responsibility Assignment*. Springer London, London, 165–186.

[27] Ian Sommerville, Tim Storer, and Russell Lock. 2009. Responsibility modelling for civil emergency planning. *Risk Management* 11, 3 (2009), 179–207.

[28] Leendert W. N. van der Torre and Yao-Hua Tan. 1999. Contrary-to-duty reasoning with preference-based dyadic obligations. *Ann. Math. Artif. Intell.* 27, 1-4 (1999), 49–78. https://doi.org/10.1023/A:1018975332469