Ontological Reasoning and Abductive Logic Programming for Service Discovery and Contracting

Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni

DEIS - Department of Electronics, Informatics and Systems
University of Bologna
viale Risorgimento, 2
40136 - Bologna, Italy
{federico.chesani, paola.mello, marco.montali, paolo.torroni}@unibo.it

Abstract. The Service Oriented Architecture paradigm, and its implementation based on Web Services, have been the object of an intense research and standardization activity. One of the most challenging open research issues is the discovery of Web Services on the base of the functionality they offer. Several proposals, including WSMO and OWL-S, rely on Semantic Web technologies to enrich service descriptions with semantic information about the offered functionalities. Current solutions however mainly focus on ontological aspects.
In this paper we focus instead on aspects related to web service interaction. We present an application framework that addresses the service discovery problem by accommodating both the ontological and the interaction perspectives. It does so by reasoning with a number of ingredients that specify how web services and users interact: web service semantic descriptions and declarative rules, and user goals, parameters and policies.
The aim of our framework is to discover and propose to user services which (1) ontologically fulfill the requested functionalities, and (2) allow the user to interact with the services and possibly achieve her goals, without violating her policies.

1 Introduction

Service Oriented Architecture (SOA) and Web Services are rapidly emerging as standard architectures for distributed application development. The adoptions of well-known network protocols and communication standards have solved interoperability and heterogeneity issues at the level of hardware and software. Eventually, the use of off-the-shelf solutions/services is becoming possible, although concerns about the adoption of such components have been raised. In particular, the search of software components on the basis of the functionality they provide, rather than on some syntactical property, is still an open research matter. To this end, some authors identify Semantic Web technologies as a promising way to address this issue [5, 9]. The idea is to augment web service

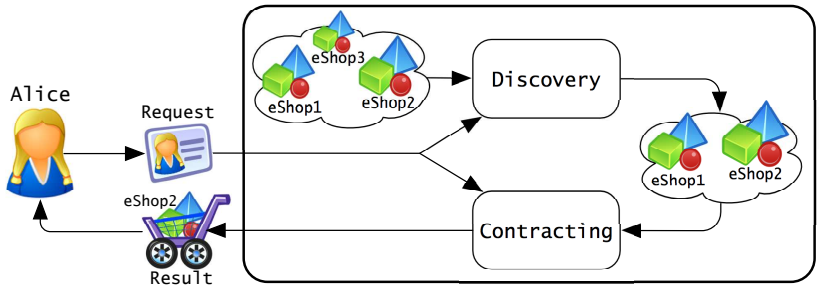

Fig. 1. Framework overview

descriptions by semantic information that can be used to search for *Semantic Web Services* (SWS).

In our view, searching for a service means to identify such components that *i*) can potentially satisfy the user needs in terms of outputs and effects, and *ii*) can be invoked by the customer and interact with her without violating her interaction policies. An example of interaction policy could be a user constraint that prevents providing a credit card number to a service which is not certified, or a service constraint that prevents to accept credit card payments for items out of stock or with more than 30 % discount. Hence, a user request contains not only a description (given in semantic terms) of the user desires, but also the user policies, which we call the "behavioural interface."

We consider the search of a SWS as the process of identifying, among a given set of services, those components that both satisfy the ontological requirements (i.e., they provide the requested functionality), and the constraints on interaction (i.e., they support the requested behaviour). Following [5], we propose a two-fold search process. A first phase, called *discovery*, consider a requester's desires, and using ontology-based reasoning, produces a selection of services that can potentially satisfy a request of such a kind. A second step, called *contracting*, matches the requester's interaction policies with those of every selected service, and establishes whether an interaction can be effectively achieved, and if the result matches the user goals.

In previous work [2] we focused on the contracting step only, exploring the possibility of using the $\mathcal{S}$CIFF language [3] to specify both the interaction policies of the requester and of the service. Then, by exploiting the $\mathcal{S}$CIFF proof procedure, we showed how our approach could be a feasible solution for reasoning upon interactions. The solution proposed, called SCIFF Reasoning Engine (SRE), is a framework able to establish if a given SWS and a requester can fruitfully inter-operate, taking as input the behavioural interfaces of both the participants, and producing as output a sort of a contract (a plan).

Building on [2], in this paper we describe an implemented, application framework that encompasses both the discovery and the contracting steps, in a unified search process. In particular, we accommodate service discovery by ontology-based reasoning, specifically by matching the user inputs, outputs and effects,

with "similar" inputs, outputs and effects of potentially suitable services. Similarity here can be intended as subsumption. Once a set of software components has been shortlisted from all the available services, in a second step the SRE evaluates which services can effectively (and successfully) interact with the user. Ontology-based reasoning is again used, also in this second stage, to solve terminological mismatches in the user and service policy descriptions.

1.1 An example scenario

User **Alice** forgot to buy her brother a Christmas present and now she is desperately searching the Internet for an online shops that sells the last crime fiction novel featuring Inspector Montalbano. She is particularly worried because she needs to find a shop that delivers in 3 days, and offers a gift wrapping service. She can pay by Mastercard, PayPal, or bank transfer.

eShop1 is the biggest Internet book seller, and through its semantic web services it provides every type of books. Its services are advertised with the generic term "book". Moreover, it has some policies about the delivery: fast delivery (one working day) is allowed only if payment is performed by means of bank transfer; otherwise, standard delivery (one week) is the default option. Gift wrapping is possible, but only with standard delivery.

eShop2 is a small internet seller, specialized in crime fiction books only. Its service advertisements use again the generic term "book," and it accepts "credit card" payment. The shop offers fast delivery, and gift wrapping without restrictions.

eShop3 is a huge, consumer electronics chain, which advertise its internet service as "selling anything." It accepts any payment method, supports only fast delivery and gift wrapping.

We envisage a scenario in which Alice queries our search engine, looking for a solution for her problem. Our application performs the discovery step, and the shops **eShop1** and **eShop2** are selected as possible services providing what Alice is looking for. However, this is not enough to guarantee that an interaction is possible. Due to its policies, gift wrapping and fast delivery are incompatible options for **eShop1**. Our application, during the contracting phase, reasons upon the policies of the services and the customer. It tries to generate a possible interaction plan where all the constraints are respected, and if it fails, the service is simply discarded. In this example, the only service that has a chance to fulfill Alice's requirements is **eShop2**, which is returned as a result from the search process.

2 The description of a Semantic Web Service

Several, different solutions have been proposed in order to describe a Semantic Web Service, and a vast literature is available on the topic. However, up to now no solution has been widely accepted, and a proper standard for defining the

semantics of a Web Service is still a matter of research. This is indeed, in our opinion, one of the most limiting factor for the adoption of SWS standards.

The two major proposals, the Web Service Modeling Ontology (WSMO [13]) and the Semantic Markup for Web Services (OWL-S [8]) address both the ontological aspects and the behavioral issues, when describing a SWS. However, WSMO proposes a rigid structure, and the behavioural aspects are mainly defined on abstract state machines semantics. OWL-S instead allows a great flexibility, leaving the users the possibility of extending it following their needs: behavioural aspects are supported by allowing their definition using at least two languages (Knowledge Interchange Format, KIF [6] and Semantic Web Rule Language, SWRL [14]), plus the possibility of adding any required language. WSMO offer a complete suite of tools for editing SWS descriptions, while OWL-S partially leaves this burden to the single users.

In the current implementation of our application framework, we made a compromise, by choosing to keep ontological and behavioural aspects separate from each other. They are represented by two different sets of information and stored in two different files. Ontological aspects are represented by means of an OWL-S 1.1 profile, while behavioural properties are defined using the SRE language. This allows us to keep our architecture open to other SWS description solutions, without giving up the powerful SRE formalism for representing the interaction issues.

2.1 Representing Interaction Issues: the SRE Approach

The SCIFF Reasoning Engine (SRE, [2]) describes behavioural aspects of a Semantic Web Service in terms of logical triplets in the form: $\langle s, ws, \mathcal{S}_{ws} \rangle$, where s identifies a service, ws is the name of a web service that provides s, and $\mathcal{S}_{ws} \equiv (\mathcal{KB}_{ws}, \mathcal{P}_{ws})$ is the *web service behavioural specification* that ws associates to s. In particular, for a given provider ws providing s, a set of policies (rules) $\mathcal{P}_{ws}$ describes ws's behaviour with respect to s, and a knowledge base $\mathcal{KB}_{ws}$, in the form of a logic program, contains information that ws wants to disclose to potential customers. Client's queries are also defined in terms of the service that the client (c) needs, and a behavioural description of her own interaction preferences $\mathcal{S}_c \equiv (\mathcal{KB}_c, \mathcal{P}_c)$.

Policies $\mathcal{P}_{ws}$ describe a web service's observable behaviour in terms of *events* (e.g., messages). SRE considers two types of events: those that one can directly control (e.g., if we consider the policies of a web service ws, a message generated by ws itself) and those that one cannot (e.g., messages that ws receives, or does not want to receive). Atoms denoted by **H** denote "controllable" events, those denoted by **E** and **EN** denote "passive" events, also named *expectations*. Since SRE reasons about possible future courses of events, both controllable events and expectations represent *hypotheses* on possible events. We restrict ourselves to the case of events being messages exchanged between the two parties in play. The notation is:

- $\mathbf{H}(ws, ws', M, T)$ denotes a message with sender ws, recipient ws', and content M, which ws expects to be sending to ws' at a time T;

- $\mathbf{E}(ws', ws, M, T)$ denotes a message with sender ws', recipient ws, and content M, which ws expects ws' to be sending at a time T;
- $\mathbf{EN}(ws', ws, M, T)$ denotes a message with sender ws', recipient ws, and content M, which ws expects ws' *not* to be sending at a time T.

In other words, **H** denotes messages that the *sender* expects to be sending, and **E**/**EN** denote messages that the *recipient* expects (not) to be receiving. For example, if **eShop1**'s policies contain $\mathbf{H}(alice, eShop, hello, 10pm)$, the interpretation is: "*alice* expects to be saying *hello* to me at 10pm," or, more simply, since *alice* is in fact responsible for producing such an event, "*alice* says *hello* to me at 10pm."

Note that message content, sender, recipient, and time can also be partially or totally left (un)specified. For example, the notation allows us to express that service ws expects a message from some other web service, but does not know the message content, sender, nor time yet. Moreover, it is possible to pose constraints on such elements, for example it is possible to say that a message must match with a certain pattern, or that its time must fall within a certain interval.

More formally, web service specifications in SRE are relations among expected events, expressed in the $\mathcal{S}$CIFF [3] Abductive Logic Programming (ALP) language.[1] A program in ALP is a triplet $\langle \mathcal{KB}, \mathcal{A}, \mathcal{IC} \rangle$, where $\mathcal{KB}$ is a logic program, $\mathcal{A}$ (sometimes left implicit) is a set of literals named *abducibles*, and $\mathcal{IC}$ is a set of *integrity constraints*. In the domain of web services, we will use ALP as a reasoning paradigm that combines backward, goal-oriented reasoning with forward, reactive reasoning [7]: two aspects that frequently, in the context of web services, are treated separately from each other, but which share important links. The ALP reasoning process will consider as knowledge base the $\mathcal{KB}_{ws}$ associated to the web service. The set of abducibles $\mathcal{A}$ includes all possible **E**/**EN** expectations, **H** events, and predicates left undefined by $\mathcal{KB}_{ws}$, while $\mathcal{IC}_{ws}$ will correspond to $\mathcal{P}_{ws}$, containing ws's policies.

Each integrity constraint in $\mathcal{P}_{ws}$ is a rule in the form *Body* $\rightarrow$ *Head*. The *Body* of an IC is a conjunction of events, literals and CLP constraints; the *Head* is either a disjunction of conjunctions of events, literals and CLP constraints, or *false*. The operational reading of ICs is similar to that of forward rules: whenever the body becomes true, the head is also made true.

For example, **eShop1**'s policy about gift wrapping being available only with standard delivery can be expressed as follow:

$$\begin{aligned} &\mathbf{H}(Customer, eShop1, request(Book\_id), T_r) \\ &\mathbf{H}(Customer, eShop1, select\_option(gift\_wrap), T_o) \\ \rightarrow &\mathbf{E}(Customer, eShop1, select\_delivery(standard), T_d) \wedge T_d < T_o \\ &\wedge \mathbf{H}(eShop1, Customer, deliver(Book\_id, delivery(standard)), T_a) \wedge T_a > T_o. \end{aligned} \quad \text{(eShop1)}$$

Policy eShop1 has the following meaning: if a *Customer* has requested an item with id *Book_id*, and the *gift_wrap* option, then **eShop1** expects (i.e., requires) that the Customer has beforehand selected *standard* delivery, and it expects

[1] For an introduction to ALP, see [4].

to tell *Customer* that he will send her the book. Note the two events in the head: the first one is an expectation about an external event, representing a requirement, while the second one is an expectation about a controllable event, representing the specification of a behaviour. The body of this specific constraint instead contains expectations about events controllable by other services, which in a way amount to "perceptions" of occurred events.

SRE reasons by considering a goal, expressed as a conjunction of events and constraints (e.g., "alice expects (wants) to buy a book, and it wants it to be wrapped and delivered within 3 days"), and by firing integrity constraints whose body is verified by matching events.

3 Search for Semantic Web Services

As in [5], we distinguish between the discovery step and the contracting step. During the discovery step, the user request for a service is compared with each SWS description, and possible services are selected. In this phase the main problem is that the terms/concepts used in the request could differ from those used in the service description. At the end of the discovery phase, a set of services that fulfill the user requirements is provided as result. Such a set will be the input for the next phase.

The contracting phase then focuses its attention on the interaction policies, i.e., on the set of rules that each partner has declared as representing its behavioural interface. Here the problem consists of deciding whether an interaction can effectively happen, with the further constraint that such interaction should also achieve the user goals. Besides the interaction issue, again an ontological problem must be faced: user and service policies could be defined using different terms (referring to different ontologies or to related but different concepts). Our framework uses ontological reasoners to cope with this problem.

3.1 The Discovery Phase

In our framework the discovery phase has been implemented following the algorithm proposed by Paolucci et al. in [10]. In that work, the authors conceived the discovery issue as the problem of selecting those services whose input and output parameters "match" the input and output parameters provided in the user request. Hence, the user request is confronted with every service description available.

More specifically, request and a service outputs match if for each output parameter of the request there is a matching output parameter in the service description. Inputs match if for each service input there is a matching request input. Each output parameter in the request is checked against all the output parameters of the service; similarly, each input parameter of the service is checked against all the input parameters in the user request.

Paolucci et al. propose to analyze user request and service description parameters pairwise, and to classify them into four categories according to their

matching level (for example, exact match is stronger than subsumption). The levels are: *exact*, *plugin*, *subsume* and *fail*.

Our implementation slightly differs from the algorithm described in [10], due to some practical considerations we made on the search process. In fact, while the original algorithm has been developed to support a scenario where a service queries the search engine looking for a service with certain characteristics, our application envisages an initial situation where a human user queries the search engine looking for a service. So we decided to relax some constraints: in particular, if a service provides more outputs than the user required, such a service is selected anyway. Moreover, if a service requires more input than the user specified in her request, again the service is selected nonetheless. We believe in fact that a user might be interested to a service that requires more input that she specified in her request: we want to give the user the opportunity of deciding if she is willing to provide more inputs. Similar motivation for the outputs: we want to leave it to the user to decide if a service that provide more outputs than requested, is still of interest for her.

3.2 The Contracting Phase

Our application framework exploits the SRE framework in the contracting phase. Roughly speaking, the SRE engine tries to establish if there exists a possible sequence of events (exchanged messages) that respect the constraints of both the service and the user. If such interaction is possible, SRE comes up with a sort of a plan indicating the messages that should be exchanged. Note that the reasoning process is driven by the user goals: not all the possible interactions are of interest, but only those that lead to satisfying the user's needs.

We extend the SRE architecture presented in [2]. First, we assume that each service description, given in terms of $\mathcal{S}$CIFF rules, contains also a reference to one (or more) ontologies. This is required to support the alignment of the terms adopted by different behavioural policies.

Second, a simple (and not complete) alignment process is carried out over the terms used in the rules for describing the interaction policies. To this end, the knowledge base of the service description is augmented with a set of predicates `is_a/2`, and the underlying unification mechanism of the $\mathcal{S}$CIFF proof procedure has been slightly modified to accommodate ontological subsumption relations.

4 Framework Architecture

The framework has been organized as a set of web services, each of them providing one functionality. We aimed to realize a modular architecture, with the intention of using the framework as a proof of concept of our solutions, and in particular of the SRE approach. Our framework provides two facilities: registering and querying. The first if provided by a component used by service providers, which can register a service description given as the pair OWL-S profile / SRE profile. The second facility is provided by a component which enables the users

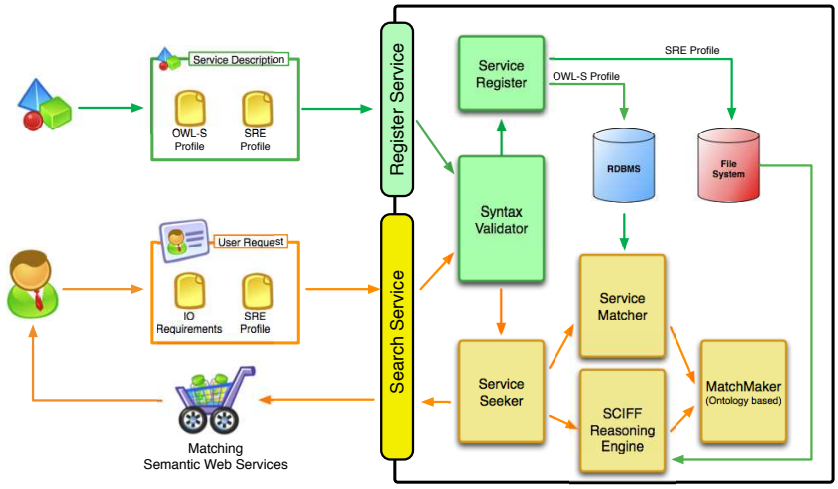

Fig. 2. The Framework architecture

to submit their request, and which returns as an output a list of SWS that fulfill the user requirements.

The various web services composing our system are shown in Figure 2. Any interested Web Service can register itself at our application, by providing an OWL-S description together with a SRE description of the provided functionality and its behavioural policies. After a syntactic validation performed by the *Syntax Validation* module, the pair OWL-S description / SRE description are sent to the *Service Register* component who manages the storing procedures. In particular, it stores OWL-S profiles by means of a Relational Database, while SRE profiles are directly stored in the file system. OWL-S profiles are pre-processed, and a summary of the profile is extracted for each SWS: in particular, information about the referred ontology and the terms used for the inputs and the outputs description are explicitly stored in the database. This choice allows to simplify the matching algorithm described in Section 3.1 by means of ad-hoc, dynamically created SQL queries on the DBMS. If the storing procedure terminates successfully, an acknowledge is returned back to the service that required the registration of its profiles.

A user starts the query process by providing a request, composed of a description of the functionality she is looking for, together with her behavioural policies. The description of the desired service is given in terms of inputs and outputs: however we assume such lists as a sort of "indication" of the needs of the user, and a certain flexibility is adopted, as explained in Section 3.1. After an initial syntactic validation step, is passed to the *Service Seeker* component, who manages and coordinates the search process orchestrating the other components. The input/outputl list is passed to the *Service Matcher* component that selects, among the registered services, only those ones that could satisfy the user request. To this end, the *Service Matcher* implements the algorithm explained

in Section 3.1. The ontology subsumption relation is evaluated by the *MatchMaker* component, a simple wrapper for the Pellet reasoner [11]. This choice was because Pellet is open source, and it is written in Java: the same language used to implement most of our framework's components.

The list of services selected by the *Service Matcher*is returned back to the *Service Seeker*, that in turn gives it to the SCIFF Reasoning Engine module. Such module reasons about the possibility and the existence of an interaction that could effectively satisfy the user needs, as explained in Section 3.2. Its outcomes consists of a restricted list of services together with, for each service, a possible interaction plan that *i*) justify why that service has been selected, and 2) shows how the user can successfully interact with the service. To overcome terminological issues, also the SCIFF Reasoning Engine module uses the *MatchMaker* and its integrated Pellet reasoner. Finally, the list of selected services is returned back to the user by the *Service Seeker* module.

5 Discussion and Conclusion

Many authors tackled the service matching problem, considering both the ontological and the interaction perspective. Among them, Kifer et al. propose a comprehensive solution to the discovery and contracting problem in [5], by tackling both the aspects of discovery and contracting, but only considering user goals. In our framework instead the user can specify their policies (their intended behavioural interface) by means of declarative rules. The use of SRE, and in particular of the underlying $\mathcal{S}$CIFF language and proof procedure, allows a great expressivity when defining the policies, with the typical advantages of declarative approaches and of a solid underlying a computational counterpart. In [12], Ragone et al. use the idea of Concept Covering and Concept Abduction to overcome some of the limits of previous matching approaches, and to address also the composition problem. In this work we focus on discovering a SWS able to satisfy the user requests, and we concentrate our efforts instead on reasoning about the interaction aspects: in this perspective, we understand a SWS as a complex agent, for which the interactions aspects play an important role.

The architecture introduced in this paper is a first prototype, and from the software development viewpoint it is still in an alpha stage. Moreover, it suffers from many limitations we introduced to support flexibility and extensibility. E.g., we currently store service profiles using a relational DBMS. This enables a certain flexibility, but does not permit to take advantage of ontology-aware storing systems. We assumed also that all the service profiles and the user requests refer to the same set of ontologies, and that no alignment among ontology terms is required.

Nevertheless, the results of our preliminary test, conducted using OWL-S profiles available at `http://projects.semwebcentral.org/`, are encouraging. They show that our proposal for the service discovery and contracting problem is a viable solution, with the advantages of offering a powerful yet simple, declarative language for expressing service policies.

In the future, we intend to run a more comprehensive comparison with other solutions, by considering both expressive power and computational performance. We also plan to extend our application by providing support to other service description languages, such as WSMO, SAWSDL and WSMO-Lite. Another research direction we intend to pursue regards the integration between rules and ontologies.This is perceived to be an important issue by the whole Semantic Web community. Some initial work in this direction is presented in [1]. Finally, we are interested in encoding SRE rules using emerging standards for rule interchange such as the Rule Interchange Format (RIF) and its Framework for Logic Dialects (RIF-FLD).

Acknowledgements

This research has been partially supported by the Italian MIUR PRIN 2007 project No 20077WWCR8, *"Le Forme di Correlazione tra Italian Style, Flussi di Turismo e Trend di Consumo del Made in Italy"*, and by the Italian FIRB project *TOCAI.IT*. The prototype framework has been partially developed by F. Succi, M. Cavina, L. Acerbo and D. Pierangeli during their MEng project.

References

1. M. Alberti et al. Exploiting semantic technology in computational logic-based service contracting. Submitted to SWAP'08.
2. M. Alberti et al. Web service contracting: Specification and reasoning with SCIFF. In *Proc. ESWC 2007*:68–83.
3. M. Alberti et al. Verifiable agent interaction in abductive logic programming: the SCIFF framework. *ACM TOCL*, 9(4), Article 29, 2008.
4. A. C. Kakas et al. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.
5. M. Kifer et al. A logical framework for web service discovery. In *Proc. Semantic Web Services: Preparing to Meet the World of Business Applications, ISWC 2004 Worshop*.
6. S. U. Knowledge Systems. Knowledge interchange format. `http://ksl.stanford.edu/knowledge-sharing/kif/`.
7. R. A. Kowalski and F. Sadri. From logic programming towards multi-agent systems. *AMAI*, 25(3/4):391–419, 1999.
8. D. Martin et al. `http://www.daml.org/services/owl-s/`.
9. S. A. McIlraith et al. Semantic web services. *IEEE Int. Sys.*, 16(2):46–53, 2001.
10. M. Paolucci et al. Semantic matching of web services capabilities. In *Proc. ISWC 2002*:333–347.
11. Pellet reasoner. `http://pellet.owldl.com/`.
12. A. Ragone, T. D. Noia, E. D. Sciascio, F. M. Donini, S. Colucci, and F. Colasuonno. Fully automated web services discovery and composition through concept covering and concept abduction. *Int. J. Web Service Res.*, 4(3):85–112, 2007.
13. D. Roman et al. Web service modeling ontology. *Applied Ontology*, 1(1):77 – 106, 2005.
14. W3C. Semantic web rule language. `http://www.w3.org/Submission/SWRL/`, May 2004.