International Journal of Web Services Research

July-September 2011, Vol. 8, No. 3

Table of Contents

Editorial Preface

i **Services Modeling, Provisioning, and Access**
Liang-Jie Zhang, Kingdee International Software Group, China

Research Articles

1 **A Computational Logic Application Framework for Service Discovery and Contracting**
Marco Alberti, New University of Lisbon, Portugal
Massimiliano Cattafi, University of Ferrara, Italy
Federico Chesani, University of Bologna, Italy
Marco Gavanelli, University of Ferrara, Italy
Evelina Lamma, University of Ferrara, Italy
Paola Mello, University of Bologna, Italy
Marco Montali, Free University of Bozen-Bolzano, Italy
Paolo Torroni, University of Bologna, Italy

26 **Case Studies and Organisational Sustainability Modelling Presented by Cloud Computing Business Framework**
Victor Chang, University of Southampton and University of Greenwich, UK
David De Roure, University of Oxford, UK
Gary Wills, University of Southampton, UK
Robert John Walters, University of Southampton, UK

54 **Provisioning Virtual Resources Adaptively in Elastic Compute Cloud Platforms**
Fan Zhang, Tsinghua University, China
Junwei Cao, Tsinghua University, China
Hong Cai, IBM China Software Development Lab, China
James J. Mulcahy, Florida Atlantic University, USA
Cheng Wu, Tsinghua University, China

70 **Semantic-Based Access to Composite Mobile Services**
Xu Yang, Virginia Tech, USA
Athman Bouguettaya, RMIT University, Australia
Xumin Liu, Rochester Institute of Technology, USA

A Computational Logic Application Framework for Service Discovery and Contracting

Marco Alberti, New University of Lisbon, Portugal

Massimiliano Cattafi, University of Ferrara, Italy

Federico Chesani, University of Bologna, Italy

Marco Gavanelli, University of Ferrara, Italy

Evelina Lamma, University of Ferrara, Italy

Paola Mello, University of Bologna, Italy

Marco Montali, Free University of Bozen-Bolzano, Italy

Paolo Torroni, University of Bologna, Italy

ABSTRACT

In Semantic Web technologies, searching for a service means identifying components that can potentially satisfy user needs in terms of inputs and outputs (discovery) and devise a fruitful interaction with the customer (contracting). In this paper, the authors present an application framework that encompasses both the discovery and the contracting steps in a unified search process. In particular, the authors accommodate service discovery by ontology-based reasoning and contracting by reasoning about behavioural interfaces, published in a formal language. To this purpose, the authors consider a formal approach grounded on Computational Logic. They define, illustrate, and evaluate a framework, called SCIFF Reasoning Engine (SRE), which can establish if a Semantic Web Service and a requester can fruitfully inter-operate, by computing a possible interaction plan based on the behavioural interfaces of both. The same operational machinery used for contracting can be used for runtime verification.

Keywords: *Abductive Logic Programming, Analysis, Contracting, Interoperability, Proof Procedure, Rules, Semantic Discovery, Web Services*

DOI: 10.4018/jwsr.2011070101

INTRODUCTION

Service Oriented Architecture (SOA) and Web services are emerging as standard architectures for distributed application development. 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 services on the basis of the functionality they provide, rather than on some syntactical property, is still an open research issue. Some authors are seeking for a possible solution to this problem among the technologies for the Semantic Web (McIlraith et al., 2001; Kifer et al., 2004). The idea is to augment Web service descriptions by semantic information that can be used to search for *Semantic Web Services* (SWS).

WEB SERVICE DISCOVERY AND CONTRACTING WITH SRE

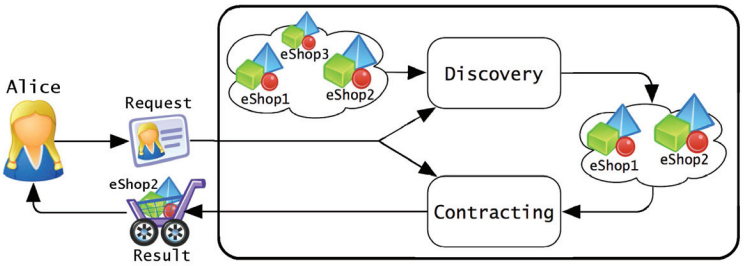
In our view, searching for a service means to identify components that i) can potentially satisfy the user needs, and ii) can be invoked by the customers and interact with them. Of course, an interaction is successful if it satisfies user/service goals and constraints; as an example, a user might not want to provide a credit card number to a non-certified service, or a service could disallow credit card payments for items out of stock or with more than 30% discount. Hence, a user request should contain not only a description (given in semantic terms) of the user desires, but also the user constraints about the *content* and the *order* of the exchanged messages, to be matched with the constraints that constitute a "behavioural interface" of the user/service. We consider the search of a SWS as the process of selecting, among a given set of services, those components that both *i)* satisfy the ontological requirements, by providing the requested functionality; and *ii)* satisfy the constraints on interaction, by supporting the requested behaviour.

In this article we present SRE (*SCIFF Reasoning Engine*), a framework for searching Semantic Web Services that takes into account requested functionalities as well as requested behaviours. Following Kifer et al. (2004), SRE adopts a two-step search process (Figure 1). For the first phase, called *discovery*, it extends a well-known algorithm from the literature (Paolucci et al., 2002). In particular, it considers a requester's desires, and, using ontology-based reasoning on knowledge expressed in OWL (Bechhofer et al., 2004), produces a shortlist of services that can potentially satisfy a request of such a kind. The second step, called *contracting*, matches the requester's behavioural interface with those of each shortlisted service. The purpose is to establish constructively whether an interaction can be effectively achieved, and if such interaction leads to achieve the user/service goals. Our choice has been to represent behavioural interfaces using a declarative, rule-based approach, and to exploit computational logic techniques to perform the reasoning task. Note that "contract" is a term also used in other contexts, such as in software engineering (Design By Contract, Brunel et al., 2004). This work does not focus on software engineering issues, and we use the terms "contract" and "contracting" in the sense it is used by others in the SWS literature.

We formalise the external behavior interfaces of users and web services in a declarative language which is a modification of the SCIFF abductive logic programming language (Alberti et al., 2008), originally developed for the specification of open societies. In this new language, behavioural interfaces are defined by *Integrity Constraints* (ICs): a sort of reactive rules used to generate and reason about expectations on possible evolutions of a given interaction. The SCIFF language is equipped with a proof procedure, which SRE exploits to automatically reason upon the behavioural interfaces. Such a reasoning task aims to establish if an interaction can be effectively achieved and, in case of a positive answer, to provide also a sort of a (partial) plan of a possible interaction.

In previous work (Alberti et al., 2007), we presented a prototype of the SRE framework,

Figure 1. Looking for the right service in SRE



focused on the contracting step alone. We introduced the idea of representing behavioural constraints as rules, showing how to reason from them using abductive inference, and Abductive Logic Programming, a powerful formalism for hypothetical reasoning with rules, in particular (Kakas et al., 1992). This article describes in greater detail an extended SRE framework, which supports the discovery step, and features ontological reasoning in both the discovery and the contracting phase. The abductive proof procedure can be used to verify, at runtime, the compliance of interacting parties to their contract.

INNOVATIVE SRE FEATURES

SRE advances state-of-the-art solutions on SWS search process in several directions. The first one is an increased flexibility of service descriptions, by choosing a declarative, rather than procedural, approach to specification, as advocated in recent literature (van der Aalst et al., 2005; Montali et al., 2010). An SRE WS description contains both functional properties and behavioural specifications. The latter are given using a declarative approach based on rules. This choice allows a greater flexibility when compared to procedural ways of describing service behavioural aspects, since it only focuses on constraining the desired behaviour, and it does not require to idly specify all the possible behaviours. This may be less significant in very simple behaviours, but it makes a difference when the behaviours to specify are complex and result from a set of requirements given in natural language. For example, a service might not provide enough information on how to interact for security reasons: in such a case, the plan of the possible interaction would be partial and/or incomplete. The ability to reason with partial information and still provide the user with an answer is a benefit mainly due to the declarative, as opposed to procedural, nature of SRE.

The underlying operational machinery, the SCIFF abductive proof procedure (Alberti et al., 2008), enriched with the possibility to access ontological knowledge, is used for (i) automated contracting (i.e., computing a partial plan of interaction that achieves the user's goals while satisfying the user's and the service's policies) and possibly (ii) runtime verification of compliance of the interacting parties to their contract, or possibly to an external interaction protocol or choreography. Whereas several approaches and formalisms have been proposed to deal with one of the aforementioned tasks, SRE provides a uniform specification language and a reasoner to perform all of them. While outside the scope of this article, in order to further demonstrate the flexibility of SCIFF-based approach, we would like to point out that in previous work we applied the same reasoner to a-priori verification of compliance of a service specification to a choreography (Alberti et al., 2006).

Moreover, the use of a computational logic language bears another important advantage, given by its built-in unification and constraint-handling engine: it enables reasoning about the content of the messages during the contracting phase. In other words, in SRE one can specify

the external behaviour of a SWS also in terms of the content of the messages. For instance, an eCommerce service could specify different behaviours based on the fact that the invoker of the service is a premium customer rather than a newcomer, or it could support special behaviours for customers that acquire an amount of goods which exceeds a certain threshold. Last but not least, we extended the SCIFF reasoning tool, which can now accommodate ontological inference also while dealing with rules. This solves an issue with the behavioural interfaces that are described by rules that use different terms.

ROADMAP

The article is structured as follows. First we position our technology with respect to the Semantic Web architecture, and we introduce a simple running scenario, which we will use throughout the paper to illustrate the framework. Then, we describe the discovery and the contracting phases. In the section "Behavioural Specification Language and Semantics" we introduce the language and tools used by SRE to represent and reason upon the behavioural interfaces of the services. In the section "Implementation Architecture", we describe the technical details about the implemented framework, followed by a discussion of related work and conclusions.

SRE AND THE SEMANTIC WEB

With respect to the semantic Web cake, our framework affects two layers: ontology and logical reasoning (Figure 2). They are internally represented with two different sets of information and stored in two different files. Ontological aspects are represented by means of an OWL-S 1.1 profile, and the reasoning upon such information is performed by Pellet (Parsia & Sirin, 2004). Behavioural properties are defined using a computational logic language, named SCIFF (Alberti et al., 2008). This allows us to keep the architecture open to other SWS description solutions, without giving up the powerful SCIFF formalism for representing the interaction issues.

Rule-based languages have been advocated to enhance the semantic information associated to Web content. As claimed by Bry and Eckert (2006), a rule-based approach to reactivity on the Web provides many benefits. To cite some:

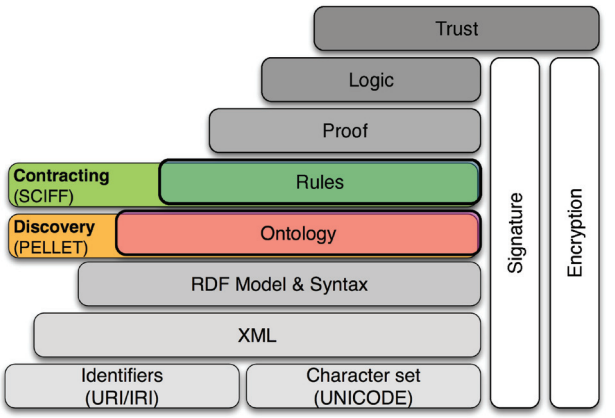
- Rules are easy to understand for humans. Requirement specifications often come in the form of rules expressed in a natural or formal language;
- Rule-based specifications are flexible and easy to adapt;
- Rules are well-suited to be processed and analysed by machines;
- Rules can be managed in a single knowledge base or in several knowledge bases possibly distributed over the Web.

Moreover, if the rules are defined in a logic-based formal language, computational logic technologies, i.e., languages, tools, and proof procedures, can be successfully used to perform reasoning tasks, such as a-priori verification of interoperability or runtime verification of compliance. For these reasons, the integration of ontological and rule-based knowledge in service description is a key advantage of the SRE approach, as we demonstrate in the remainder of this paper.

ESHOP SCENARIO

Let us consider an artificial example. User *alice* forgot to buy her brother a Christmas present and now she is desperately searching the Internet for an online shop that sells the last crime fiction novel featuring detective Montalbano. She is particularly worried because she needs to find a shop that can deliver the book to Italy within 3 days. She can pay cash or by credit card. She is also worried about frauds, so she will not provide her credit card number to any electronic shop, but only to those belonging to a Better Business Bureau (BBB).

Figure 2. Semantic Web cake and addressed levels



eShop1 is the biggest Internet book seller, and through its semantic Web services it provides all kinds of books. Its services are advertised with the generic term "book". Concerning delivery, fast delivery (one day) is allowed only if payment is performed by credit card; otherwise, standard delivery (one week) is the default option.

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 and "cash" payment. The shop delivers in two days but only delivers to customers in the European Union. It proves its membership to the BBB upon request.

eShop3 is a huge consumer electronics chain, which advertises its Internet service as "selling hardware." It accepts all payment methods, supports delivery in 1 day, provides its membership to BBB if credit card payment is chosen.

In our scenario, *alice* queries a search engine, which performs a discovery step; in this phase *eShop1* and *eShop2* are shortlisted as possible services (*eShop3* is discarded as it does not sell items related to the concept of "book"). However, this does not guarantee that an interaction is possible. Due to *alice*'s policy, the credit card number is provided only to BBB members, so only *eShop2* remains viable. Feasibility of delivery, based on geographical criteria, has to be checked too.

DISCOVERY AND CONTRACTING

As in Kifer et al. (2004), we distinguish between a discovery and a contracting phase. During discovery, the user request for a service is compared with each SWS description, and possible services are selected based on ontological matching criteria. The discovery process returns a shortlist of candidate services, which might fulfill the user requirements, because their descriptions match, at least partially, the client's requests. Such a shortlist is given in input to the next phase.

The contracting phase uses the behavioural interfaces, i.e., the set of rules that each partner has declared to represent their constraints about how the interaction should happen. Here we see exactly which services fulfill the requirements. In particular, the problem is to decide whether, given a set of service/user rules, an interaction between client and service can effectively happen and achieve the user/service goals.

Discovery

Our framework's discovery phase uses an extended version of a semantic matching algorithm defined by Paolucci et al. (2002). We are aware of other proposals for semantic matching (e.g., Oundhakar et al., 2005; Ragone et al., 2007) aimed at overcoming its limitations (one com-

mon criticism is that the algorithm considers only inputs and outputs of a service, whereas it ignores preconditions and effects, see Wang et al., 2006). However, our modular architecture supports other matching algorithms, and does not depend on the particular choice.

Given a client's request, discovery is conceived as the problem of selecting those services that might satisfy the client's needs. The client publishes her needs in terms of information she is willing to provide as input to the service, and in terms of outputs she expects from the service. Similarly, each service advertises its own capabilities as a list of pieces of information that it requires in input, paired with the information it will provide in output. Each piece of information represents a parameter, and the discovery problem can be intended as looking for those services whose input (output) parameters *match* the input (output) parameters of the client. The client's request is compared with every service description available, and a set of candidate services is returned to the client.

Paolucci et al. (2002) assume that each parameter is described via ontological propositions. For instance, in SRE, the parameters are defined in terms of OWL-S concepts. The parameters of each available service profile are checked against the parameters in the client's request. *Subsumption* is used to decide if two parameters *match*. There are four different matching levels, depending on the subsumption relation: *exact*, if it is possible to establish that two parameters defined with different terms both refer to the same concept; *plugin* and *subsume* if a parameter is subsumed/subsumes the other; and *fail* if no subsumption relation can be identified.

In this phase, the main problem is that the terms/concepts used in the request could differ from those used in the service description. In particular, two seemingly different terms could actually refer to the same concept. In this work, we do not deal with this problem, which we leave to further research and extensions of SRE. We simply assume that there exists a common ontology, and we let the ontological reasoner match terms and concepts that may be different (in that they do not match exactly, according to the aforementioned classification), but are in the same ontology.

To increase flexibility, our implementation generalizes the original algorithm defined by Paolucci et al. (2002), which requires the number of input/output parameters to be exactly the same in the client request and in the service profile, in order for a service to be discovered. In SRE, if a service provides more outputs or requires less inputs than those stated in the client's query, such a service may be selected anyway. As a result, the contracting phase may be provided with more choices, and it may indeed end up selecting a service which would be discarded according to Paolucci et al. (2002). The reason is that the contract could be satisfactory for both parties even if the client discards some of the outputs of the service, or if it provides an input disregarded by the service. Suppose, e.g., that *eShop2* has launched a marketing campaign to attract new customers: together with each book bought on its Web site, *eShop2* will provide also a free voucher of 10$ valid for the next order. The algorithm in its original form would disregard *eShop2* since it provides both a book and a voucher, while *alice* is looking for a book only. We instead include *eShop2* in the set of discovered services, following the intuition that *alice* can freely decide what to do with the bonus voucher.

Notable alternative approaches include semantic contexts and similiarity functions (Isaac et al., 2008). In our scenario, however, a client sending a request already expressed in term of a concept would not provide enough information for identifying a context, thus making such alternative less interesting from our viewpoint. Indeed, this would be a very interesting research direction once we extend and enrich the amount of information carried in every client's request.

Contracting

Based on the output of the discovery phase, SRE moves on to the next phase to decide whether an interaction can be achieved. To this end,

SRE 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 it does, SRE produces a plan (see the section on Operational Semantics) whose (partially ordered) actions are the messages that should be exchanged. Note that the reasoning process is user goal-driven: not all the possible interactions are of interest; only those that satisfy the user's needs are. The contracting phase, as well as the discovery phase, may involve ontological reasoning, which is delegated to an external ontological reasoner.

At the logical level, the SWS are represented in SRE with triplets:

$$\langle s, ws, P_{ws} \rangle$$

where s identifies a certain service, ws is the name of a Web service that provides s, and P_{ws} is ws's behavioural specification (see Definition 1).

A client c may submit a query to SRE by providing a description of a service it needs (its *goal* G), and possibly its behavioural specification P_c. SRE answers to c's query by providing a number of triplets:

$$\langle ws, \varepsilon, \Delta \rangle$$

each containing the name of a Web service that provides a certain service s satisfying the goal G, plus some additional information. Intuitively, ε encodes a possible sequence of inter-operations between ws and c regarding s, while Δ contains a number of additional validity conditions for ε. For example, in the *eShop* scenario, if G is "get book", ε may be "ws (*eShop2*) shows evidence of membership to the BBB, c pays by credit card", and Δ may be "delivery in Europe".

BEHAVIOURAL SPECIFICATION LANGUAGE AND SEMANTICS

In SRE, the behavioural specification describes a Web service's observable behaviour in terms of *events*, representing, for instance, exchanged messages. SRE considers two types of events: those that one actor receives or can directly control (e.g., if we consider Web service ws' behavioural interface, a message generated by ws itself) and those that one desires (not) to receive. Such information is represented in SRE by means of logic atoms (Lloyd, 1987) with functor **H** that denote "happened" events, and with functor **E** (resp. **EN)** which denote "desired" (resp. "undesired") events, also known as *positive* (resp. *negative*) *expectations*. Both happened events and expectations represent *hypotheses* about events that may happen in the future, which encode possible interactions between a client and a service. Arguments of event atoms can contain variables (conventionally with uppercase initial, e.g., M, T), that can be associated with domains and restrictions, as in Constraint Logic Programming. For example, a time variable may be associated with restrictions representing deadlines. Although our proof-procedure and implementation support both continuous and discrete time intervals, in this paper we will only consider integer domains.

The syntax of event atoms is as follows:

- H(ws, ws_1, M, T), for messages (with content M) that a Web service ws_1 receives from another Web service ws, or (changing perspective) that ws intends to send to ws_1 at some time in the domain of T;
- E(ws_1, ws, M, T) for messages (with content M) expected by ws to be sent to him from ws_1 at some time inside the domain of T;
- EN(ws_1, ws, M, T) for messages (with content M) that are expected by ws not to be sent to him by ws_1 at *any* time in the domain of T (note that in negative expectations variables are implicitly universally quantified).

Message contents are logical terms. Intuitively, the functor represents the type of message (*ask*, *send*, etc.), although there is no predefined set of keywords.

Web service specifications in SRE are relations among expected events, expressed by an Abductive Logic Program (ALP). An ALP

(Kakas et al., 1993) is a triplet $\langle P, A, IC \rangle$, where P is a logic program, A is a set of predicate symbols named *abducibles*, and IC is a set of logical formulas called *integrity constraints*. P defines predicates, A the contains the predicate symbols with no definition in P, and the role of IC is to control the ways predicates built upon elements of A are hypothesised, or "abduced". Reasoning from an ALP is usually goal-directed (being G a goal), and it amounts to finding a set of abduced hypotheses Δ built from predicates in A such that $P \cup \Delta \models G$ and $P \cup \Delta \models IC$. Kowalski and Sadri (1999) have shown how an abductive logic programming proof-procedure such as the IFF by Fung and Kowalski (1997) can reconcile backward, goal-oriented reasoning with forward, reactive reasoning.

Definition 1. Web Service Behavioural Specification. Given a Web service *ws*, its *behavioural specification* P_{ws} is an ALP, represented by the triplet

$$P_{ws} \equiv \langle KB_{ws}, A, IC_{ws} \rangle$$

where:

- KB_{ws} is *ws*'s Knowledge Base,
- A is the set of *abducible predicates*, and
- IC_{ws} is *ws*'s set of Integrity Constraints.

KB_{ws} is a set of backward rules (clauses) which declaratively specifies pieces of knowledge of the Web service. Note that the body of KB_{ws} 's clauses may contain E/EN expectations about the behaviour of the Web services. A is the set of *abducible predicates*, which includes E/EN expectations, H events, and predicates not defined in KB_{ws}. *Integrity Constraints* (ICs) are forward rules[1], of the form *Body→Head*. The *Body* of IC_{ws} is a conjunction of events, literals and CLP[2] constraints (over integer or real numbers); *Head* is either a disjunction of conjunctions of events, literals and CLP constraints, or *false*. Operationally, whenever *Body* becomes true, the IC fires and forces *Head* to become true, possibly by assuming some abducibles to be true and by activating CLP propagation procedures. The syntax of KB_{ws} and IC_{ws} is given in Table 1 and Table 2, respectively.

Interactions are specified by means of *integrity constraints*, i.e., logical relations that link the messages (modeled by **H** atoms) sent or received by a Web service with those (modeled by **E**/**EN** atoms) it expects from other Web services or from the remote user. Constraints over variables can specify relations of various types, including temporal relations (e.g., deadlines), linear constraints and inequalities. Operationally, such definitions are used to make assumptions on the possible evolutions of the interaction. Sample ICs and clauses are given in Eq. (1) through Eq. (8).

THE ESHOP SCENARIO IN SRE

Let us now show an SRE implementation of the eShop scenario. The SRE language does not require any particular keyword in the argument of events/expectations. In this example, we will use the following symbols to identify different types of message content: *ask*, *pay*(*Item*,*PaymentMethod*), *request_guar* and *give_guar* respectively for requesting and giving a guarantee, and *deliver* to simulate the actual delivery.

User *alice*'s behavioural interface states that if a shop asks to pay cash at some time T_a, *alice* will proceed with the payment in a later time T_r:

$$H(s, alice, ask(pay(Item, cash)), T_a) \rightarrow H(alice, S, pay(Item, cash), T_r) \wedge T_a < T_r \quad (1)$$

If, instead, the payment is done by credit card, then *alice* will require evidence of the shop's affiliation to the *BBB* (2) and only afterwards proceed to pay (3).

Table 1. Grammar of the knowledge base (KB)

KB_{ws}	::=	*[Clause]**
Clause	::=	*Atom ← Cond*
Cond	::=	*ExtLiteral [∧ ExtLiteral]**
Extliteral	::=	*[¬]Atom | true | Expect | Constraint*
Expect	::=	**E***(Term, Term, Term, TimeTerm) |*
		EN*(Term, Term, Term, TimeTerm)*
TimeTerm	::=	*Variable | Integer*

Table 2. Grammar of the integrity constraints (IC)

IC_{ws}	::=	*[IC]**
IC	::=	*Body → Head*
Body	::=	*(Event | Expect) [∧ BodyLit]**
BodyLit	::=	*Event | Expect | Atom | Constraint*
Head	::=	*Disjunct [∨ Disjunct]* | false*
Disjunct	::=	*(Expect | Event | Constraint)*
		*[∧ (Expect | Event \ Constraint)]**
Expect	::=	**E***(Term, Term, Term, TimeTerm) |*
		EN*(Term, Term, Term, TimeTerm)*
Event	::=	**H***(Term, Term, Term, TimeTerm)*
TimeTerm	::=	*Variable | Integer*

$$H(S, alice, ask(pay(Item, cash)), T_a) \rightarrow H(alice, S, request\_guar(bbb), T_{rg}) \wedge T_{rg} > T_a \wedge E(S, alice, give\_guar(bbb), T_g) \wedge T_g > T_g > T_{rg} \quad (2)$$

$$\mathrm{H}(S, alice, ask(pay(Item, cc)), T_a) \wedge \mathrm{H}(S, alice, give\_guar(bbb), T_g) \rightarrow \mathrm{H}(alice, S, pay(Item, cc), T_p) \wedge T_p > T_a \wedge T_p > T_g. \quad (3)$$

The behavioural interface of *eShop2* is also represented by means of integrity constraints. "If an acceptable customer requests an item, then I expect the customer to pay for the item with an acceptable payment methods. If the customer is not acceptable, I will inform him/her of the failure (4). If an acceptable customer pays with an acceptable means of payment, I will deliver the item within two days (5). If a customer requests evidence of my affiliation to the BBB, I will provide it (6)."

$$\begin{aligned}&\mathrm{H}(C, eShop, request(Item), T_r)\\ &\rightarrow accepted\_customer(C) \wedge accepted\_pay(How)\\ &\wedge \mathrm{H}(eShop, C, ask(pay(Item, How)), T_a) \wedge T_a > T_r\\ &\wedge \mathrm{E}(C, eShop, pay(Item, How), T_p) \wedge T_p > T_a\\ &\vee rejected\_customer(C)\\ &\wedge \mathrm{H}(eShop, C, inform(fail), T_i) \wedge T_i > T_r.\end{aligned} \quad (4)$$

$$\begin{aligned}&H(eShop, C, ask(pay(Item, How)), T_a)\\ &\wedge H(C, eShop, pay\ (Item, How), T_p)\\ &\wedge accepted\_customer(C) \wedge accepted\_pay(How)\\ &\rightarrow H(eShop, C, deliver\ (Item), T_d) \wedge T_p < T_d < T_p + 2\end{aligned} \quad (5)$$

$$H(C, eShop, request\_guar(bbb), T_{rg}) \rightarrow H(eShop, C, give\_guar(bbb), T_g) \wedge T_g > T_{rg} \quad (6)$$

The notion of acceptability for customers and payment methods from *eShop*'s viewpoint, given with the *accepted_customer*/1 and *accepted_pay*/1 predicates, can be defined in *eShop*'s knowledge base, as we proposed in a previous work (Alberti et al., 2007). The fact that only EU residents are accepted customers, is defined by the following clauses:

$$\begin{aligned} &accepted\_customer(Customer) \leftarrow \\ &\quad resident\_in(Customer, Location), \\ &\quad accepted\_destination(Location). \\ &rejected\_customer(Customer) \leftarrow \\ &\quad resident\_in(Customer, Location), \\ &\quad not\ accepted\_destination(Location). \\ &accepted\_destination(european\_union). \\ &accepted\_pay(cc). \\ &accepted\_pay(cash). \end{aligned} \quad (7)$$

On her side, *alice* knows she is resident in the EU:

$$resident\_in(alice, european\_union). \quad (8)$$

Declarative Semantics

In SRE, a client *c* specifies a goal *G*, related to a requested service. *G* will often be an expectation, but in general it can be any goal, defined as a conjunction of expectations, CLP constraints, and any other literals. *c* also publishes a (possibly empty) knowledge base KB_c, and a (possibly empty) set of rules IC_c. The declarative semantics is meant to define a set of expectations ε and validity conditions Δ about a possible course of events that, together with KB_c and KB_{ws}, satisfies the conjunction of the integrity constraints $IC_c \cup IC_{ws}$ and the goal *G*. Note that we do not assume that all of *ws*'s knowledge base is available to SRE, as it need not be entirely a part of *ws*'s public specifications. KB_{ws} can even be the empty set. However, in general, ICs can involve predicates defined in the KB: such as "delivery in Europe". If the behavioural interface provided by *ws* involves predicates that have not been made public through KB_{ws}, SRE makes assumptions about such unknown predicates, and considers unknowns that are neither **H** nor **E**/**EN** expectations as literals that can be abduced. These are contained in the set Δ, of a returned triplet $\langle ws, \varepsilon, \Delta \rangle$ (see the section "Implementation Architecture"), and can be regarded as conditions which must be met to assure the validity of ε as a possible set of expectations achieving a goal.

We define declaratively the set of abductive answers $\langle ws, \varepsilon, \Delta \rangle$ representing possible ways *c* and *ws* can interact to achieve *G* (we assume that KB_c and KB_{ws} are consistent) via the two following equations:

$$KB_c \cup KB_{ws} \cup \varepsilon \cup \Delta \models G \quad (9)$$

$$KB_c \cup KB_{ws} \cup \varepsilon \cup \Delta \models IC_c \cup IC_{ws} \quad (10)$$

where ε is a conjunction of **H** and **E**, **EN** atoms, Δ is a conjunction of abducible literals, and the notion of entailment is grounded on the *possible models* semantics defined by Sakama and Inoue (2000) for abductive disjunctive logic programs. In the possible models semantics, a disjunctive program generates several (non-disjunctive) *split programs*, obtained by separating the disjuncts in the head of rules. Given a disjunctive logic program *P*, a *split program* is defined as a (ground) logic program obtained from *P* by replacing every (ground) rule

$$r : L_1 \vee \cdots \vee L_l \leftarrow \Gamma$$

from *P* with the rules in a non-empty subset of $Split_r$, where

$$Split_r = \{ L_i \leftarrow \Gamma \mid i = 1, \ldots, l \}$$

By definition, P has in general multiple split programs. A *possible model* for a disjunctive logic program P is then defined as an answer set of a split program of P.

In Sakama and Inoue (2000), the possible models semantics was also applied to provide a model theoretic semantics for Abductive Extended Disjunctive Logic Programs (AEDP), which is our case. Semantics is given to AEDP in terms of *possible belief sets*. Given an AEDP $\Pi = \langle P, A \rangle$, where P is a disjunctive logic program and A is the set of abducible literals, a possible belief set S of Π is a possible model of the disjunctive program $P \cup E$, where P is extended with a set E of abducible literals ($E \subseteq A$).

Definition 2 (Answer to a goal G). An *answer E to a (ground) goal G* is a set E of abducible literals constituting the abductive portion of a possible belief set S (i.e., $E = S \cap A$) that entails G.

We rely upon possible belief set semantics, but we adopt a new notion for minimality with respect to abducible literals. In Sakama and Inoue (2000), a possible belief set S is A-minimal if there is no possible belief set T such that $T \cap A \subset S \cap A$. We restate, then, the notion of A-minimality as follows:

Definition 3 (A-minimality possible belief set). A possible belief set S is *A-minimal* iff there is no possible belief set T *for the same split program* such that $T \cap A \subset S \cap A$.

More intuitively, the notion of minimality with respect to hypotheses that we introduce is checked by considering the *same* split program, and by checking whether with a smaller set of abducible literals the same consequences can be made true, but in the same split program. Finally, we provide a model-theoretic notion of explanation to an observation, in terms of answer to a goal, as follows.

Definition 4 (A-minimal answer to a goal). E is an *A-minimal answer to a goal G* iff $E=S\cap A$ for some possible A-minimal belief set S that entails G.

Definition 5 (Possible Interaction about G). A *possible interaction about a goal G* between a client *c* and a Web service *ws* is an A-minimal set $\varepsilon \cup \Delta$ such that Eq. (9) and (10) hold.

Among possible interactions, we identify those that are *coherent*:[3]

Definition 6 (Coherent Possible Interaction about G). A possible interaction $\varepsilon \cup \Delta$ about a goal G is *coherent* iff:

$$\varepsilon \models E(X, Y, Action, T),$$
$$EN(X, Y, Action, T) \rightarrow false \quad (11)$$

Possible interactions about a goal G generally contain (minimal) sets of events and expectations about messages raised either by *c* and *ws*. Moreover, further abducible literals in Δ represent assumptions about unknown predicates (for *c* and *ws*).

Among coherent possible interactions only those where the course of events expected by *c* about *ws*'s messages is *fulfilled* by *ws*'s messages (i.e., happened events match positive and negative expectations), and the course of events expected by *ws* about *c*'s messages is *fulfilled* by *c*'s messages.

Definition 7 (Possible Interaction Achieving G). Given a client *c*, a Web service *ws*, and a goal G, a *possible interaction achieving G* is a coherent possible interaction $\varepsilon \cup \Delta$ satisfying the following equations:

$$\varepsilon \models \mathrm{E}(X, Y, Action, T) \rightarrow \mathrm{H}(X, Y, Action, T) \quad (12)$$

$$\varepsilon \models EN(X, Y, Action, T),$$
$$H(X, Y, Action, T) \rightarrow false \quad (13)$$

By Definition 7, every positive expectation raised by *c* or *ws* on the behaviour of the other party must be fulfilled by an event hypothetically performed by the other party (Eq. (12)), and every negative expectation raised by *c* or *ws* on the behaviour of the other party must not match any event hypothetically performed by the other party (Eq. (13)).

Operational Semantics

The operational semantics is an extension of the SCIFF proof-procedure (Alberti et al., 2008). SCIFF was initially developed to specify and verify agent interaction protocols in open environments. It processes events drawing from a given narrative of events and abduces expectations, checking that each one of them is fulfilled by the occurring events.

SRE extends SCIFF and abduces **H** events as well as expectations. As opposed to SCIFF, the event narrative is not an input. It is instead an output, as all possible interactions are hypothesized. Moreover, in SRE events not matched by an expectation (acceptable in an open scenario) cannot be part of a *possible interaction achieving* the goal. For this reason, in SRE a new transition labels each **H** event with an *expected* flag as soon as a matching expectation is abduced. At the end of the derivation, unflagged **H** will cause failure. The reasoning module of SRE, like the SCIFF proof procedure, has been implemented in Prolog, and can run on top of SICStus or SWI Prolog (Fung & Kowalski, 1997).

The soundness and completeness results, proven for the SCIFF proof-procedure (Alberti et al., 2008), also hold for SRE.

Operationally, the SCIFF reasoning engine (SRE) operates on the union of the user's and service's disclosed integrity constraints (IC_c, IC_{ws}) and knowledge bases (KB_c, KB_{ws}) in order to find a (partial) plan $\varepsilon \cup \Delta$ able to satisfy equations (9) and (10). An example of such a computation follows.

Example of SRE Computation

In the *eShop* scenario, it starts with *alice*'s goal, which is to obtain a *book* by interacting with a shop: *alice* will start an interaction by *request*ing the book, and she will expect the shop to *deliver* it:

$$\mathrm{H}(alice, eShop, request(book), 0) \wedge$$
$$\mathrm{E}_{alice}(eShop, alice, deliver(book), T_d) \wedge 0 \leq T_d \leq 3.$$

SRE reasons about events and expectations, and tries to match them in order to find a successful arrangement. To this end, SRE tags each expectation with its holder: in this example, *alice* is the entity that holds the expectation of *eShop* delivering the book.

Now, the happened *request* event triggers new integrity constraints. In particular, it activates (4), which in turn can be satisfied in two alternative ways: either the transaction succeeds, or it fails because *alice* is rejected as a customer. The SCIFF proof-procedure generates a proof tree, which is explored depth-first, as customary. In the first branch, SCIFF verifies that *alice* is an acceptable customer. This can be proven by reasoning from *alice*'s and *eShop*'s knowledge bases together. SCIFF then abduces that *eShop* will ask for payment with one of the accepted payments. Let us consider the case in which payment by *cc* is assumed. In that case, *eShop* will ask for the payment and will expect *alice* to be responsible for it:

$$\mathrm{H}(eShop, alice, ask(pay(book, cc)), T_a) \wedge$$
$$\mathrm{E}_{eShop}(alice, eShop, pay(book, cc), T_p).$$

The new abduced event fires (2), because it verifies its body. SRE assumes that *alice* will follow her own behavioural interface, by requesting the guarantee and expecting a reply. The request event will fire (6), which forces *eShop* to provide the guarantee.

In the end, a set of events and a set of expectations are generated by abduction. If the expectations are matched by corresponding events, the abductive process succeeds, otherwise the exploration of the current branch will fail, and an alternative branch will be selected (if there exists one). In this way, the SCIFF proof-procedure determines if there exists at

least a set of events that satisfies a given ALP, and computes both the abduced events and the expectations. A successful computation yields a sequence of actions that satisfies all the parties' constraints and goals.

Note that representing the acceptability of costumers as a mere logic program would not work in case the customer declared *resident_in(alice,italy)*, as the term *italy* does not syntactically unify with *european_union*. Other problems may occur if acceptability is defined by a transitive, symmetric relation. For example, a service could accept requests from a set of trusted peers, and also from customers that are trusted by them, in a transitive fashion. If the abductive proof procedure adopts a depth-first search, symmetric and/or transitive relations can lead to loops that could prevent the proof-procedure to terminate in finite time. Our solution to this problem is described in the section "Ontological Reasoning in SRE".

IMPLEMENTATION ARCHITECTURE

We developed a prototype implementation of SRE. The framework is organized as a set of Web services, which provides two facilities: registering and querying. The first one is used by service providers, which register by providing a service description in terms of the pair (OWL-S profile, behavioural profile). The second facility accepts requests from the users, and returns a list of SWS's that fulfill the requirements.

The Web services composing the system are shown in Figure 3. A Web Service can register at our application, by providing its own specification. After a syntactic validation performed by the *Syntax Validation* module, a service description is sent to the *Service Register* component which manages the storing procedures. It stores OWL-S profiles by means of a RDF store, while behavioural 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; this simplifies the matching algorithm described in the section "Discovery," by identifying and handling some specific cases. If the storing procedure terminates successfully, an acknowledgement is returned to the service asking for registration.

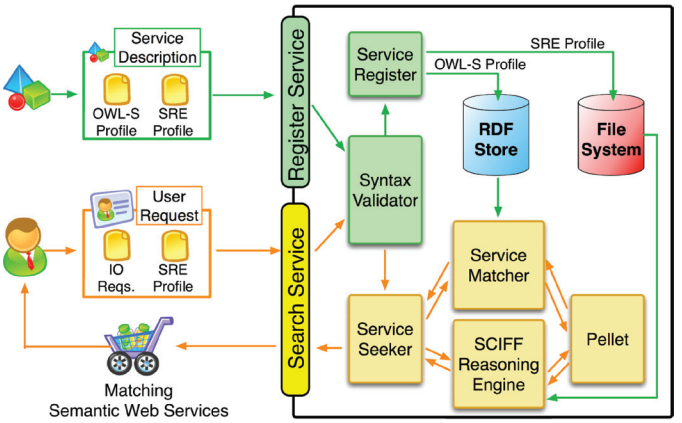
A user starts the process with a request, composed of a description of the functionality she is looking for, together with her own behavioural interfaces. The description of the desired service is given in terms of inputs and outputs: however we assume such lists as a sort of "indication'" of the needs of the user, and some certain flexibility is adopted, as explained in the section "Discovery". After an initial syntactic validation step, the request is passed to the *Service Seeker* component, which manages and coordinates the search process orchestrating the other components. The input/output list is passed to the *Service Matcher* component that selects, among the registered services, only those that satisfy the user request. To this end, the *Service Matcher* implements the algorithm explained in the section "Discovery". The ontology subsumption relation is evaluated by a simple wrapper for the Pellet reasoner (Parsia & Sirin, 2004).

The list of services selected by the *Service Matcher* is returned to the *Service Seeker*, which gives it in turn to the *Contracting Reasoner* module. Such a module reasons about the possibility and the existence of an interaction that could effectively satisfy the user needs, as explained in the section "Contracting". Its outcomes consist of a shortlist of services, which for each selected service contains a possible interaction plan that justifies why that service has been selected. This also shows how the user can successfully interact with the service. Pellet is again used, in integration with SCIFF, to bridge rules with ontologically expressed knowledge that is useful for contracting. Finally, the list of selected services is returned to the user by the *Service Seeker* module.

Ontological Reasoning in SRE

SRE represents ontologies in OWL (Web Ontology Language), the W3C recommendation for ontology representation on the Web

Figure 3. The SRE architecture



(Bechofer et al., 2004), based on Description Logics (Lutz, 2008) and on XML and RDF syntax. OWL guarantees expressivity (with such features as stating subclassing relations, constructing classes on property restrictions or by set operators, defining transitive properties and so on) and decidability (if using OWL Lite or OWL DL) in a straight-forward and domain modeling-oriented notation. Moreover, since OWL is tailored for the Web, it provides support for expressing knowledge in distributed contexts (identified by URIs) and its recognized standard status is a warranty on interoperability and reusability issues. Community-driven development of Semantic Web tools already provides good support for OWL ontology management tasks such as editing (Noy et al., 2001), which has become a feasible task even for the non-KR-savvy user.

Figure 4 illustrates a possible ontological representation of *eShop*'s constraints concerning acceptable customers and means of payments, merged with *alice*'s own knowledge. For example, we may want to express that acceptedCustomer is a subclass of the potentialCustomer class, and that it is disjoint from the rejectedCustomer class. This would correspond to the following OWL fragment:

```
<owl:Class
rdf:about="#acceptedCustomer">
 <rdfs:subClassOf rdf:resource="#poten
tialCustomer" />
 <owl:disjointWith rdf:resource="#reje
ctedCustomer" />
</owl:Class>
```

The following assertion states that cash is an instance of the acceptedPayment class:

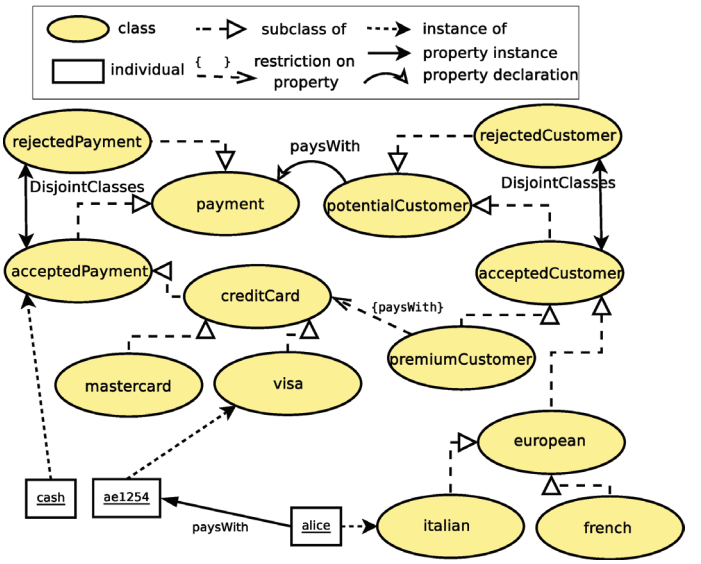
```
<owl:Thing rdf:about="#cash">
 <rdf:type
rdf:resource="#acceptedPayment" />
</owl:Thing>
```

The following is the declaration of the paysWith property:

```
<owl:ObjectProperty rdf:ID="paysWith">
 <rdfs:domain rdf:resource="#potential
Customer" />
 <rdfs:range rdf:resource="#payment"
/>
</owl:ObjectProperty>
```

The following assertion states that alice is an instance of italian, with value ae1254 for the paysWith property:

Figure 4. A graphical representation of the ontology




```
<owl:Thing rdf:about="#alice">
 <rdf:type rdf:resource="#italian" />
 <paysWith rdf:resource="#ae1254" />
</owl:Thing>
```

Now *alice* no longer needs to explicitly express that she is a EU resident. If *alice* simply declares that she is an Italian resident, then ontological reasoning can infer that *alice* is indeed European by considering, e.g., some official ontology of the EU, enlisting all the member states.

Another interesting feature of Description Logic (and thus OWL) ontologies is the definition of classes using restrictions on properties. For instance we could define a class, premiumCustomer, representing the accepted customers who pay by credit card. This notion could then be used to add refinements to user constraints (for instance for the purpose of providing such customers with a faster delivery service, or with a lower price) and since alice is an accepted customer and pays with her credit card, the ontological reasoning would automatically recognize her as a premiumCustomer.

Interfacing SCIFF and Ontological Reasoners

During contracting, we can access and use the knowledge represented in OWL as illustrated above, thanks to an existing interface between SCIFF and the external ontological reasoner Pellet (Parsia & Sirin, 2004). This solution involves a Prolog meta-predicate which invokes ontological reasoning on desired goals, an intercommunication interface from SCIFF to the external component (which incorporates a query and results translation schema) and the actual reasoning module. Both modules can access both local and networked knowledge. This approach has proved better, both for performance and expressiveness, than encoding ontological knowledge as SCIFF rules (Alberti et al., 2009).

As suggested by Hustadt et al. (2004) and Vrandecić et al. (2006), goals given to the meta-predicate are handled by considering single arity predicates as "belongs to class (with same name of predicate)" queries and double arity ones

as "are related by property (with same name of predicate)" queries. We also implemented a caching mechanism that reduces the overhead caused by external communication. In this way, the meta-predicate first checks if a similar query (i.e., involving the same predicate) has been issued before and, only if not, it invokes the external reasoner and stores its answers as Prolog facts. The OWL reasoning module uses the Pellet (Parsia & Sirin, 2004) API, while the communication interface uses the Jasper Prolog-Java library (SICStus). This solution provides full OWL(-DL) expressivity, including features such as equivalence of classes and properties, transitive properties, declaration of classes on property restriction and property-based individual classification.

Runtime Verification

The result of the contracting step is a (partial) sequence of messages that achieve the goal while respecting both the user's and the service's specification; therefore, if both the user and the service involved in the interaction stick to their specifications, then the interaction should be successful. However, the implementation of a Web service may not respect its own specifications, or network problems may arise, unexpected events may occur, deadlines may be missed because of overloaded servers or malicious attacks. On-the-fly verification aims at finding possible violations of the agreements. As pointed out in Maximilien and Singh (2005), automatic monitoring and run-time conformance test can be very useful in modeling trust and providing rating for empirical selection of services. SCIFF features on-the-fly verification (Alberti et al., 2008), and has been applied successfully to multi-agent interaction protocols and service choreographies. For example, the following narrative:

$$\mathrm{H}(alice, eShop, request(book), 0).$$
$$\mathrm{H}(eShop, alice, ask(pay(cash)), 1).$$
$$\mathrm{H}(alice, eShop, pay(cash), 2).$$
$$\mathrm{H}(eShop, alice, deliver(book), 3).$$

can be checked as events occur, to show that it satisfies *alice*'s specification and goal. The following:

$$\mathrm{H}(alice, eShop, request(book), 0).$$
$$\mathrm{H}(eShop, alice, ask(pay(cash)), 1).$$
$$\mathrm{H}(alice, eShop, pay(cc), 2).$$

breaks the agreement between *alice* and *eShop*, because *alice* uses a different means of payment from the one requested by *eShop*, so the latter does not react to the last message, and does not deliver the *book*. Using SCIFF, users can determine a broken agreement, and possibly label other services as unreliable. This mechanism would be a useful feature for service discovery engines as it can provide verifiable user feedback, which is already built inside SCIFF. Of course, interacting parties can be verified at runtime not only against their agreement, but also against external interaction protocols or choreographies, if so desired.

Preliminary Performance Evaluation

We tested SRE using up to 10,000-class ontologies and a pool of 4 simple use cases. We observed that SRE provides a response within seconds or tens of seconds, and that it spends most of the time in the communication between SCIFF and Pellet, and only a smaller fraction of time in the reasoning tasks.

In order to assess the scalability of such interface, we tested the performances of the *Contracting Reasoner* in simple contracting scenarios.

We experimented with randomly generated ontologies. Each ontology, composed of N classes, was built starting from its root node, and recursively trying, for each node, five attempts of child generation, each with probability 1/3. In Table 3 the we report the time spent for loading the ontology into the reasoner and for the actual query (PC with Intel Celeron 2.4 GHz CPU, times in seconds, average over 50 runs). The approach appears to scale reasonably.

RELATED WORK

Authors have proposed various ways to describe a Semantic Web Service, and a vast literature is available on the topic.

The two major proposals, the Web Service Modeling Ontology (WSMO) (Roman et al., 2005) and the Semantic Markup for Web Services (OWL-S) (Martin et al., 2004) address both the ontological aspects and the behavioral issues, when describing a SWS.

WSMO offers a complete suite of tools for editing, developing and testing SWS descriptions. However, the behavioural aspects are mainly defined on abstract state machines semantics, which make it difficult to perform the reasoning tasks we address in our work. Building on the WSMO conceptual model, Kifer et al. (2004) propose a comprehensive solution, for the discovery and contracting problem based on Flora-2, in which F-logic is used to express ontologies and achieve matching and Transaction Logic is used to model declaratively how services behave. However, SRE provides the user with the peculiar ability to express not only her goal, but also the behaviour which constrains how the goal can be achieved. This also means that the client could also be a service itself.

OWL-S can be extended by the user: behavioural aspects are supported by allowing their definition using at least two languages, Knowledge Interchange Format (KIF) (Genesereth & Fikes, 1992) and Semantic Web Rule Language (SWRL) (Horrocks et al., 2004), plus the possibility of adding any required language. OWL-S comes as a general ontology, not associated with specific dedicated tools. Our system supports service descriptions by means of OWL-S profiles.

However, let us emphasize that, in spite of the many proposals (another logic-based language for description of web services, for instance, is described in the Semantic Web Services Language (SWSL) W3C submission) (Battle et al., 2005), none to date has reached consensus, thus a proper standard for defining the semantics of a Web Service is still a matter of research. This is, in our opinion, one of the obstacles to the adoption of SWS standards.

SAWSDL (W3C, 2002) is a W3C recommendation aimed at extending WSDL documents with semantic annotations. These annotations link the different elements of a WSDL document with corresponding concepts in a semantic model (e.g., an ontology), therefore providing the foundation for semantic-based service discovery and composition. In this respect, it could be considered complementary to our approach: while SAWSDL is an ontology-agnostic way to annotate the "atomic" description of a service, SRE encompasses both a concrete language for the declarative description of the service behavioral interface, where the involved elements refer to one ontology, and a proof procedure to concretely carry out the discovery task. Therefore, SRE could rely on SAWSDL as a standardized mean to describe the messages exchanged by the services as well as their references to the corresponding ontology.

SRE shares motivations and approach with Baldoni et al. (2007). We also advocate the application of reasoning techniques on declarative service interaction specifications to enable flexibility. However, we believe that greater emphasis should be devoted to issues such as practical viability. In particular, our experimental results show that SRE and SCIFF are applicable to realistic scenario, and the implementation we discussed integrations SRE and SCIFF with existing Web service solutions and standards.

Our work is related to the automatic composition of web services. Given a set of services that are published on the web, and given a goal, the purpose of automated composition is to generate a composition of the available services that satisfies the goal.

There is a large amount of literature addressing the problem of automated composition of web services. However, most of the approaches address composition at the functional level, and only a few consider the composition at a finer degree of detail (i.e., at process-level) by considering web services as stateful, non-

Table 3. Evaluating the impact of extending the SCIFF with ontological reasoning (pellet)

Interfacing SCIFF with Pellet			
N	Load	Query	Total
100	~ 0	~ 0	~ 0
500	1.0	~ 0	1.0
1000	1.0	~ 0	1.0
5000	2.0	1.2	3.2
10000	4.0	2.8	6.8

deterministic processes. In this context, Pistore et al. (2004) propose a framework where web services are modeled at process level and planning techniques based on symbolic model checking are used for composing them. Due to the use of model checking, however, these techniques work under the rather unrealistic assumption that web services can exchange a very limited and statically determined number of data values. The authors show that reasonable performance can be obtained for web services whose variables can assume only two values. To solve this problem, a novel approach presented in Pistore et al. (2005) is an automated composition based on planning at the "knowledge level" where a solution plan encodes the desired composition. "Knowledge-level techniques" allow us not to fix finite ranges of value for variables and are very general.

Another related work about composition is Friesen and Lemcke (2007), where the authors, inspired by Pistore et al. (2005), describe a composition algorithm that generates correct Web services composition respecting user-defined goals.

Our work can be compared to Pistore et al. (2005) and Friesen and Lemcke (2007) since it shares their advantages even if, at the technical level, our work differs from them in the kind of information that we represent and store in the knowledge level (IC logic-based constraints instead of transition rules or finite state machine) as well as in the automatic technique we use (SCIFF abductive proof procedure instead of planning or an ad hoc combination algorithm). Behavioural interfaces described as IC allow, in fact, a more detailed description of web-services behaviour with respect to the functional one and the use of them allows a very rich description of them without any limitation to the range of variables used. Moreover, differently from these proposals, our techniques are easily integrated with reasoning techniques for discovery and selection of web-services. However, while in Pistore et al. (2005) the proper knowledge-level model can be obtained automatically from the published descriptions of the web services in standard process modeling and execution languages like BPEL4ws, this is for us a matter of future work. Another future work is to compare our approach in terms of execution time to the performances of related approaches.

Ragone et al. (2007) use the idea of Concept Covering and Concept Abduction to overcome some of the limitations 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.

Another notable work in the trust setting is represented by the PROTUNE (De Coi et al., 2008) framework, a rule-based system for trust negotiation. Trust negotiation has been introduced in the literature in order to address access control requirements and privacy preferences in open distributed environments. PROTUNE agents exchange rules and evidences to interoperate and make decisions related to security and privacy. The rule-based language adopted in PROTUNE is function-free and limited to stratified logic programs. This class of programs

ensures the existence of a single (2-valued) model which is an essential recommendation for security and trust. Declaratively, PROTUNE is grounded on Answer Set Semantics, and provided in two implementations (tuProlog and XSB). The SRE language is more flexible than the one of PROTUNE. In particular, it accommodates functions and domain variables, and for this reason it can model a large number of realistic scenarios more accurately.

Concerning the problem of making ontological knowledge available to a logic programming framework such as SCIFF, a first approach is to exploit the common root of logic programming and description logics in first order logic, by finding their intersection and translating ontologies to LP clauses. These problems have been addressed by Grosof et al. (2003), who named this intersection DLP (Description Logics Program) and by Hustadt et al. (2004) who proposed a method for translation. On these basis, the dlpconvert (Motik et al., 2005) tool was developed; it converts (the DLP fragment of) OWL ontologies to datalog clauses. We used dlpconvert to translate domain knowledge described in OWL to SCIFF clauses. Reasoning is then performed by SCIFF in the usual way. However, this solution limits ontological expressivity. First of all, since the DLP fragment is a proper subset of DL, some OWL axioms are not included. For instance, out of the features mentioned as available by Vrandečić et al. (2006). DisjointClasses and the important DL (and OWL) feature of class definition by restriction on properties are missing. Moreover, some axioms' translation is not actually suitable for reasoning with goal-driven operational semantics, such as resolution or unfolding, employed in SCIFF, because it leads to loops. For all these reasons, we are investigating an alternative approach, which involves the interface with Pellet, instead, and incurs no expressivity limitations.

In Motik (2006) and Lukacsy et al. (2008) the authors propose techniques for reasoning on Description Logic (respectively SHOIN and SHIQ) which are not based on the usual tableau algorithms but are instead related to, respectively, bottom-up Datalog and Deductive Database inference, and top-down Prolog resolution. In both cases the motivation comes from the attempt to offer better results in ABox reasoning with large data sets of individuals. Since Deductive Database deal natively with rules, extending the obtained reduction of the DL KB with a rule level appears straightforward. Motik (2006) shows that it is sufficient to append rules to the obtained KB. Our work is focused on very expressive rules to describe behavioural interfaces, so it goes far beyond the restrictions (motivated by computational reasons) of Motik (2006) and Lukacsy et al. (2008).

An extensive study of how rules and ontologies can be integrated, with a specific focus on Semantic Web, can be found in de Bruijn (2008) where a language, WSML, is proposed to be used as a Web Service Modeling Language in the WSMO framework (while our work aims to retain full compatibility with OWL).

In Behrends et al. (2008), the authors introduce the MARS framework, focused on the rule layer of the Semantic Web cake. In particular, MARS is equipped with a rule-based language which combines ECA rules with event and action algebras for respectively covering the specification of events and actions. Like SRE, MARS follows a declarative style of modeling; however, the MARS language is introduced as a general language aimed at specifying rules for the semantic web, and it is not equipped with specific reasoning techniques able to deal with the discovery and the contracting for (semantic) web services.

Various approaches have been proposed to deal with incompatibilities that may arise when services interact in unforeseen ways (Dumas et al., 2008). Typical incompatibilities can be classified into *signature incompatibilities* (when a service requires an operation which is not provided by the other, or when the message format is different) and *protocol incompatibilities* (when the two services expect different orderings of messages). Service adapters can be synthesized (automatically or manually, depending on the approach) to solve such incompatibilities. Currently, our approach assumes that incompatibili-

ties regarding terminology or message format have already been solved. It does not attempt to deal automatically with operations not provided or protocol incompatibilities (which result in failure). An approach to service adaptation could be to synthesize not only messages, but also integrity constraints. This, however requires an extension of the underlying SCIFF framework and is subject of future work.

Lamparter's Ph.D. thesis (2007) presents a framework for the formal expression and matching of Web Service behavioural interfaces in order to achieve improved and automatic interoperability with the goal of a 'market' of Web Services. The author argues that a very important requirement to meet for this purpose is that beahviours are specified using unarguably shared concepts. For this reason the thesis' main contribution is allegedly the proposal for an ontology which unambiguously defines the vocabulary for communication in the aforementioned market. Besides the 'Core Policy Ontology,' Lamparter also provides ontologies for Bids and Contracts.

Such ontologies are 'core ontologies' in the sense that they occupy a middle ground between top (very general) ontologies and domain specific ones. In particular they are built upon the DOLCE framework (which encomprises the DOLCE vocabulary, Ontologies of Descriptions and Situations, of Information Objects and of Plans) which the author perceives as the most successful and sound attempt to formally model general concepts, in particular with respect to time variations.

The language chosen for the ontologies is the W3C standard OWL-DL for the part that lies within Description Logic expressivity and SWRL, with limitation to the 'safe fragment' (conditions restrict variables in the head to maintain decidability), for the 'rule' part. Matchmaking and other queries on the ontologies are performed by means of SPARQL.

Our work and the author's one differ in goals and approach, because we propose a framework to specify Web Service semantic information without focusing on providing an actual formalization for the concepts used for Web Service descriptions and behavioural interfaces.

Our work has strong links with the automatic composition of services. However, some important differences distinguish this contribution. The SRE framework, in its current implementation, does not address the level of the single operations that must be invoked to use a service. SRE is focused on a higher level of abstraction, where the functionalities offered by the services are the object of the reasoning. In this sense, SRE solutions are orthogonal with respect to composition issues. Moreover, SRE focuses on finding a match between a request and a service providing a solution to such a request. Although the algorithms adopted in SRE could treat also many-to-many cases, the current implementation permits to reason upon a user request and if an interaction can happen with a single service provider: situations like a user request matched against a set of many providers are left for future extensions. Finally, SRE computes a possible plan of how the user can interact with a service in order to satisfy a goal. Such plan is partial, since it primarily depends on how much information the user/service is willing to disclose.

Finally, "Contract" is a term also used in multi-agent literature (see for instance the CONTRACT project Web site, http://www.ist-contract.org/), but it has little relation with Web service contracting as we intend it here.

CONCLUSIONS AND FUTURE WORKS

We presented a framework for Web service discovery and contracting. The discovery phase has been implemented following a well known approach, while the contracting phase makes use of a powerful, yet simple, declarative and rule-based behavioural interfaces description language and an abductive logic programming proof-procedure. Ontological reasoning is also used, both during the discovery phase, and during the contracting phase. SRE helps automate many key processes, such as dis-

covery, contracting, runtime verification and user feedback provision. A SRE prototype has been used to conduct some experimental analysis, which demonstrated the viability of the approach. Based on the implemented scenarios, we conclude that the use of SRE provides good expressivity when defining the behaviours, with the typical advantages of declarative approaches and of a solid underlying computational counterpart.

Other issues are left to future work. Let us briefly review them. A problem with ontologies is that different terms may refer to similar concepts in different ontologies, and, vice-versa, different concepts may have similar (same) names in different ontologies. This opens up the wide research problem of ontology alignment, which SRE does not currently address.

Moreover, the evaluation of the SRE framework and the experimental results we discuss here refer to a rather simple case study. Testing the system on a large scale and with more complex scenarios is also one of our intended future activities.

We also plan to extend our application by providing support to other service description languages, such as WSMO, SWSL, and SAWSDL. We plan also to encode SRE rules using emerging standards such as the Rule Interchange Format (RIF) and its Framework for Logic Dialects (RIF-FLD). From the architectural viewpoint, SRE will be extended to consider also the WSDL description of the single services, so as to provide a more comprehensive solution to the discovery issue; alternative matching approaches (possibly letting the user choose among options) will be considered.

ACKNOWLEGMENT

This work has been partially supported by the FIRB project TOCAI.it (RBNE05BFRK) and by the Italian MIUR PRIN 2007 project No. 20077WWCR8. We thank the anonymous referees for their valuable feedback on a previous version of this article.

REFERENCES

Alberti, M., Cattafi, M., Gavanelli, M., Lamma, E., Chesani, F., Montali, M., et al. (2009) Integrating abductive logic programming and description logics in a dynamic contracting architecture. In *Proceedings of the IEEE International Conference on Web Services* (pp. 254-261). Washington, DC: IEEE Computer Society.

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., & Montali, M. (2006, July 10-12). An abductive framework for a-priori verification of web services. In *Proceedings of the Eighth Symposium on Principles and Practice of Declarative Programming*, Venice, Italy (pp. 39-50). New York, NY: ACM Press.

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M., & Torroni, P. (2007). Web service contracting: specification and reasoning with SCIFF. In E. Franconi, M. Kifer, & W. May (Eds.), *Proceedings of the 4th European Semantic Web Conference: Research and Applications* (LNCS 4519, pp. 68-83).

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., & Torroni, P. (2008). Verifiable agent interaction in abductive logic programming: the SCIFF framework. *ACM Transactions on Computational Logic*, *9*(4). doi:10.1145/1380572.1380578

Alferes, J., Damásio, C., & Pereira, L. (2003). Semantic web logic programming tools. In F. Bry, N. Henze, & J. Maluszynski (Eds.), *Proceedings of the International Workshop on Principles and Practice of Semantic Web Reasoning* (LNCS 2901, pp. 16-32).

Apt, K. R., & Turini, F. (Eds.). (1995). *Meta-logics and logic programming*. Cambridge, MA: MIT Press.

Baldoni, M., Baroglio, C., Martelli, A., & Patti, V. (2007). Reasoning about interaction protocols for customizing web service selection and composition. *Journal of Logic and Algebraic Programming*, *70*(1), 53–73. doi:10.1016/j.jlap.2006.05.005

Barklund, J. (1995). Metaprogramming in logic . In Kent, A., & Williams, J. G. (Eds.), *Encyclopedia of computer science and technology*. New York, NY: Marcell Dekker.

Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., et al. (2005). *Semantic web services language.* Retrieved from http://www.w3.org/Submission/SWSF-SWSL

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P., & Stein, L. (2004). *OWL web ontology language reference.* Retrieved from http://www.w3.org/TR/owl-ref/

Behrends, E., Fritzen, O., May, W., & Schenk, F. (2008). Embedding event algebras and process for ECA rules for the semantic web. *Fundamenta Informaticae*, *82*(3), 237–263.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). *The semantic web*. New York, NY: Scientific American.

Brunel, J., Di Natale, M., Ferrari, A., Giusto, P., & Lavagno, L. (2004). SoftContract: an assertion-based software development process that enables design-by-contract. In *Proceedings of the Conference on Design, Automation and Test in Europe* (pp. 358-363). Washington, DC: IEEE Computer Society.

Bry, F., & Eckert, M. (2006). Twelve theses on reactive rules for the web. In *Proceedings of the Workshop on Reactivity on the Web*, Munich, Germany.

de Bruijn, J. (2008). *Semantic web language layering with ontologies, rules, and meta-modeling*. Unpublished doctoral dissertation, Computer Science and Physics of the University of Innsbruck, Innsbruck, Austria.

De Coi, J. L., Olmedilla, D., Bonatti, P. A., & Sauro, L. (2008). *Protune: A framework for semantic web policies*. Poster presented at the International Semantic Web Conference.

Dumas, M., Benatallah, B., & Motahari Nezhad, H. R. (2008). Web service protocols: Compatibility and adaptation. *Data Engineering Bulletin*, *31*(3), 40–44.

Friesen, A., & Lemcke, J. (2007). Composing web-service-like abstract state machines (ASM). In *Proceedings of the IEEE Congress on Web Services* (pp. 262-269). Washington, DC: IEEE Computer Society.

Fung, T. H., & Kowalski, R. A. (1997). The IFF proof procedure for abductive logic programming. *The Journal of Logic Programming*, *33*(2), 151–165. doi:10.1016/S0743-1066(97)00026-5

Genesereth, M., & Fikes, R. (1992). *Knowledge interchange format version 3.0 reference manual*. Stanford, CA: Stanford Logic Group.

Grosof, B. N., Horrocks, I., Volz, R., & Decker, S. (2003). Description logic programs: combining logic programs with description logic. In *Proceedings of the ACM Conference on World Wide Web* (pp. 48-57). New York, NY: ACM Press.

Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2004). *SWRL: A semantic web rule language combining OWL and RuleML*. Retrieved from http://www.w3.org/Submission/SWRL/

Hustadt, U., Motik, B., & Sattler, U. (2004). Reducing SHIQ- description logic to disjunctive datalog programs. In *Proceedings of the AAAI Symposium* (pp. 152-162).

Isaac, A., Matthezing, H., van der Meij, L., Schlobach, S., Wang, S., & Zinn, C. (2008, June 1-5). Putting ontology alignment in context: usage scenarios, deployment and evaluation in a library case. In S. Bechhofer, M. Hauswirth, J. Hoffmann, & M. Koubarakis (Eds.), *Proceedings of the 5th European Semantic Web Conference on the Semantic Web: Research and Applications*, Tenerife, Canary Islands, Spain (LNCS 5021, pp. 402-417).

Jaffar, J., & Maher, M. (1994). Constraint logic programming: a survey. *The Journal of Logic Programming*, (19-20): 503–582. doi:10.1016/0743-1066(94)90033-7

Kakas, A. C., Kowalski, R. A., & Toni, F. (1993). Abductive logic programming. *Journal of Logic and Computation*, *2*(6), 719–770. doi:10.1093/logcom/2.6.719

Kifer, M., Lara, R., Polleres, A., Zhao, C., Keller, U., Lausen, H., & Fensel, D. (2004). A logical framework for web service discovery. In *Proceedings of the IEEE International Conference on Web Services* (p. 119). Washington, DC: IEEE Computer Society.

Kowalski, R. A., & Sadri, F. (1999). From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, *25*(3-4), 391–419. doi:10.1023/A:1018934223383

Lamparter, S. (2007). Po*licy-based contracting in semantic web service markets*. Unpublished doctoral dissertation, Universität Karlsruhe (TH), Karlsruhe, Germany.

Lloyd, J. W. (1987). *Foundations of logic programming* (2nd extended ed.). Berlin, Germany: Springer-Verlag.

Lukacsy, G., Szeredi, P., & Kadar, B. (2008). Prolog based description logic reasoning. In M. G. de la Banda & E. Pontelli (Eds.), *Proceedings of the 24th International Conference on Logic Programming* (LNCS 5366, pp. 455-469).

Lutz, C. (2008). *Description logic resources*. Retrieved from http://dl.kr.org/

Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., & McGuinness, D. (2004). Bringing semantics to web services: the OWL-S approach. *World Wide Web (Bussum)*, *10*(3).

Maximilien, E. M., & Singh, M. P. (2005). Multiagent system for dynamic web services selection. In *Proceedings of the AAMAS Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE),* Utrecht, The Netherlands.

McIlraith, S. A., Son, T. C., & Zeng, H. (2001). Semantic web services. *IEEE Intelligent Systems*, *16*(2), 46–53. doi:10.1109/5254.920599

Montali, M., Pesic, M., van der Aalst, W. M. P., Chesani, F., Mello, P., & Storari, S. (2010). Declarative specification and verification of service choreographies. *ACM Transactions on the Web, 4*(1).

Motik, B. (2006). *Reasoning in description logics using resolution and deductive databases.* Unpublished doctoral dissertation, Universität Karlsruhe (TH), Karlsruhe, Germany.

Motik, B., Vrandecić, D., Hitzler, P., Sure, Y., & Studer, R. (2005). *Dlpconvert - Converting OWL DLP statements to logic programs.* System demonstrated at the 2nd ESWC Workshop on Inductive Reasoning and Machine Learning for the Semantic Web.

Noy, N. F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R. W., & Musen, M. A. (2001). Creating semantic web contents with Protégé-2000. *IEEE Intelligent Systems*, *16*(2), 60–71. doi:10.1109/5254.920601

Oundhakar, S., Verma, K., Sivashanugam, K., Sheth, A., & Miller, J. (2005). Discovery of web services in a multi-ontology and federated registry environment. *International Journal of Web Services Research*, *2*(3), 1–32. doi:10.4018/jwsr.2005070101

Paolucci, M., Kawamura, T., Payne, T. R., & Sycara, K. P. (2002). Semantic matching of web services capabilities. In *Proceedings of the International Semantic Web Conference*.

Parsia, B., & Sirin, E. (2004). Pellet: An OWL DL reasoner. In *Proceedings of the 3rd International Semantic Web Conference*.

Pistore, M., Barbon, F., Bertoli, P., Shaparau, D., & Traverso, P. (2004). Planning and monitoring web service composition. In C. Bussler & D. Fensel (Eds.), *Proceedings of the 11th International Conference on Artificial Intelligence* (LNCS 3192, pp. 106-115).

Pistore, M., Marconi, A., Bertoli, P., & Traverso, P. (2005). Automated composition of web services by planning at the knowledge level. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence.*

Ragone, A., Di Noia, T., Di Sciascio, E., Donini, F., Colucci, S., & Colasuonno, F. (2007). Fully automated web services discovery and composition through concept covering and concept abduction. *International Journal of Web Services Research*, *4*(3), 85–112. doi:10.4018/jwsr.2007070105

Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M. et al. (2005). Web service modeling ontology. *Applied Ontology, 1*(1).

Sakama, C., & Inoue, K. (2000). Abductive logic programming and disjunctive logic programming: their relationship and transferability. *The Journal of Logic Programming*, *44*(1-3), 75–100. doi:10.1016/S0743-1066(99)00073-4

SICStus. (n. d.). *SICStus Prolog.* Retrieved from http://www.sics.se/sicstus

van der Aalst, W. M. P., Dumas, M., ter Hofstede, A. H. M., Russell, N., Verbeek, H. M. W., & Wohed, P. (2005). Life after BPEL? *Russell: The Journal of the Bertrand Russell Archives*, *3670*(3670), 35–50.

Vrandecić, D., Haase, P., Hitzler, P., Sure, Y., & Studer, R. (2006). *DLP-an introduction*. Karlsruhe, Germany: University of Karlsruhe.

W3C. (2002). *Semantic annotations for WSDL working group*. Retrieved from http://www.w3.org/2002/ws/sawsdl/

Wang, H., Li, Z., & Fan, L. (2006, December 18-22). An unabridged method concerning capability matchmaking of web services. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence* (pp. 662-665). Washington, DC: IEEE Computer Society.

ENDNOTES

1 With "forward rules" here we mean rules that are used by the SCIFF proof procedure in a forward reasoning style, as opposed to the rules in the Knowledge base, that are used as in Prolog in a backward manner.

2 Constraint Logic Programming (Jaffar & Maher, 1994).

3 This notion is introduced to encode in the logic of SRE the intuitively understandable incompatibility between **E** and **EN** (the same event should not be expected to and not to occur at the same time).

Marco Alberti is a researcher at the Center for Artificial Intelligence (CENTRIA) of the Universidade Nova de Lisboa. His research interests are in the areas of Artificial Vision, Abductive Logic Programming, Constraint Logic Programming, Multi-Agent Systems, Normative Systems, Legal Reasoning, Service-Oriented Architectures. He has participated in the program committee of international conferences and workshops in the areas of Computational Logics and Multi-Agent Systems and in international and national research projects. He is member of Italian Association for Artificial Intelligence, and the Italian group of logic programming (GULP).

Massimiliano Cattafi is a PhD student in artificial intelligence and computational logic at the Department of Engineering of the University of Ferrara, currently focusing on rule based programming and constraint satisfaction problems.

Federico Chesani is a post-doc at the DEIS department of the University of Bologna. He got his PhD in 2007, with a thesis entitled "Specification, Execution and Verification of Interaction Protocols". His research topics concern computational logic and abduction, argumentation, and applications of logic-based tools to Semantic Web Services, Service Discovery, Composition and Contracting, commitment-based multi-agent systems, integration between rule-based languages and ontologies. He has been author and co-author of more than 50 refereed papers, and has been also involved as a reviewer for international conferences and workshops such as CLIMA and IJCAI. He is also member of the Logic Programming National Interest Group (GULP).

*Marco Gavanelli is assistant professor at the Department of Engineering of the University of Ferrara since 2004. He has published more than 50 papers on various aspects of logic programming and artificial intelligence, ranging from constraint logic programming to visual recognition, multi-criteria optimization, abductive logic programming, programming with sets. He is member of the Italian Association for Artificial Intelligence (AI*IA) and is coordinator of its interest group in knowledge representation and automatic reasoning (RCRA). He is member of the executive board of the Logic Programming National Interest Group (GULP). He has organized national and international events, and was guest editor of special issues of top-class national and international journals.*

Evelina Lamma is Full Professor in Artificial Intelligence at the Faculty of Engineering of the University of Ferrara. Her research activity focuses around logic computational logic, logic languages and theirs extensions, artificial intelligence, knowledge representation, intelligent agents and multi-agent systems, machine learning and data mining, Web Service composition and verification. She is author of several papers on these issues. She is member of the executive board of the Association for Logic Programming (ALP), and member of the Italian Association for Artificial Intelligence (AIIA), and the Logic Programming National Interest Group (GULP). She took part to several national and international research projects, and she was leader of the University of Ferrara group within the UE IST-32530 project named SOCS.

Paola Mello is Full Professor in Artificial Intelligence at the University of Bologna since 1994. She has published over 180 papers on implementation, application and theoretical aspects of programming languages, especially logic languages and their extensions towards modular and object-oriented programming, artificial intelligence, knowledge representation, expert systems and multi-agent systems, and the application of computational logic to workflow patterns and Web Service composition. She is founding member and current President of the Italian Association for Artificial Intelligence and was executive member of the Logic Programming National Interest Group (GULP) and of several national and international (UE) research projects, including COMPUNET, Esprit ALPES, CRAFT, IST-FET SOCS.

Marco Montali is currently an assistant professor with a fixed term contract in the KRDB Research Centre at the Free University of Bozen-Bolzano. He received his PhD in 2009 from the University of Bologna. His dissertation received the 'Marco Cadoli' prize, awarded by the Italian Association on Logic Programming (GULP) for the best two theses focused on Computational Logic and discussed between 2007 and 2009. He is co-author of more than 50 papers on computational logics and extensions, formal verification and monitoring, (declarative) business process modeling and business rules, service choreographies, clinical guidelines and care-flow protocols, process mining, commitment-based multiagent systems. His current research interests are in the area of specification and verification of artifact-centric business processes and services, and in the combined management of data and processes.

Paolo Torroni is an assistant professor in Computing at the University of Bologna. He obtained his PhD from the University of Bologna's Department of Electronics, Computer Science, and Systems Engineering in 2002, with a dissertation on logic-based reasoning and interaction in multi-agent systems. He is active in the area of artificial intelligence, where he authored more than 80 refereed publications, received 2 best paper awards, and edited 7 books for Springer. He has been invited for keynote talks and tutorials at several international conferences. His scientific publication record includes state-of-the-art surveys, technical and vision papers. His current recent interests are: contract-regulated multi-agent interaction, service engineering, social Web, hypothetical reasoning, diagnosis, and argumentation.