A Rule-Based Approach for Reasoning About Collaboration Between Smart Web Services

Marco Alberti[1], Federico Chesani[2], Marco Gavanelli[1], Evelina Lamma[1], Paola Mello[2], Marco Montali[2], and Paolo Torroni[2]

[1] Dipartimento di Ingegneria – Università di Ferrara – Italy
{marco.gavanelli,marco.alberti,lme}@unife.it
[2] DEIS – Università di Bologna – Italy
{fchesani,pmello,mmontali,ptorroni}@deis.unibo.it

Abstract. We present a vision of smart, goal-oriented web services that reason about other services' policies and evaluate the possibility of future interactions. We assume web services whose behavioural interface is specified in terms of reactive rules. Such rules can be made public, in order for other web services to answer the following question: "is it possible to inter-operate with a given web service and achieve a given goal?". In this article we focus on the underlying reasoning process, and we propose a declarative and operational abductive logic programming-based framework, called $\mathcal{W}$AV^e. We show how this framework can be used for a-priori verification of web services interaction.

1 Introduction

Service Oriented Computing (SOC) is rapidly emerging as a new programming paradigm, propelled by the wide availability of network infrastructures, such as the Internet. Web service-based technologies are an implementation of SOC, aimed at overcoming the intrinsic difficulties of integrating different platforms, operating systems, languages, etc., into new applications. It is in the spirit of SOC to take off-the-shelf solutions, like web services, and compose them into new applications. Service composition is very attractive for its support to rapid prototyping and possibility to create complex applications from simple elements.

If we adopt the SOC paradigm, how to exploit the potential of a growing base of web services, in order to decide which service could be used for inter-operating, becomes a strategic issue. A partial answer is given by service discovery through yellow pages or other registries. This solves part of the problem: as through discovery we only know that there are some potentially useful services, but understanding whether interacting with them will be profitable or detrimental is far from being a trivial question. In this article we consider web serivces that need to understand, pairwise, and based on a run-time exchange of policies, if they can inter-operate or not. We present a vision of smart, goal-oriented web services that reason about other services' specifications, with the aim to separate out those that can lead to a fruitful interaction. We assume that each web service publishes, alongside with its WSDL, its *behavioural interface specification.*

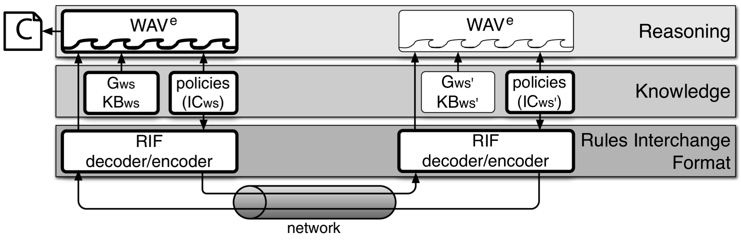

Fig. 1. The architecture of WAV^e

By reasoning on the information available about other web services' behavioural interface, each web service can verify which goals can be reached by interacting.

To achieve our vision, we propose a proof theoretic approach, based on computational logic – in fact, on abductive logic programming. We formalise service policies in a declarative language which is a modification of the $\mathcal{S}$CIFF language [7]. Policies are defined with *integrity constraints* ($\mathcal{IC}$s): a sort of reactive rules used to generate and reason about expectations on possible evolutions of a given interaction. We believe that, as advocated by Alferes et al. [9], an approach based on logic programming allows us to express knowledge in form of rules and to make inference with them. As claimed in [11], a rule-based approach to reactivity on the Web provides several benefits over conventional approaches. Rules are easy to understand for humans, and requirements often already come in the form of rules; they are well-suited for processing and analyzing by machines, and can be managed in a centralized knowledge base or distributed over the Web.

Based on $\mathcal{S}$CIFF, we propose a new declarative semantics and a proof-procedure that combines forward, reactive reasoning with backward, goal-oriented reasoning. The new framework, called WAV^e (Web-service Abductive Verification), features a language for logically defining the behavioural interface of web services (suitably encoded in RuleML), primitives for acquiring rules from the web and reasoning about them, and goal-directed discovery of web services with whom interaction could be successful.

2 The WAV^e Framework

Fig. 1 depicts our general reference architecture. The layered design of a web service has WAV^e at the top of the stack, performing reasoning based on its own knowledge and on the specifications of other web services. Web services exchange their specifications/policies encoded in a Rule Interchange Format (RIF).

In WAV^e, the observable behaviour of web services is represented by *events*:

$$Event ::= \mathbf{H}(Sender, Receiver, Message, Time).$$

Since we focus on (explicit) interaction between web services, events represent exchanged messages. Events are hypothesised, when reasoning about the capabilities of a given service. Each web service tries to foresee the future course of events that will happen, assuming that its own policies, encoded in the published specifications, will be respected by the other peers. Expected events are atoms that represent a message that the web service *ws* is expecting will be exchanged:

$$Expectation ::= \mathbf{E}_{ws}(Sender, Receiver, Message, Time).$$

The subscript indicates the web service holding the expectation. If a corresponding event ($\mathbf{H}$) indeed happens, the expectation is *fulfilled*, otherwise it is *violated*.

Web service specifications in WAV^e are relations among happened and expected events, expressed by an Abductive Logic Program (ALP). In general, an ALP [15] is a triplet $\langle P, Ab, IC\rangle$, where P is a logic program, Ab is a set of predicates named *abducibles*, and IC is a set of integrity constraints. Intuitively, P contains definitions of predicates, Ab represents unknown predicates (not defined in P), and IC constrains the way elements of Ab are hypothesised, or "abduced". Reasoning in ALP is usually goal-directed: given a goal $\mathcal{G}$, the aim is to find a set of hypotheses $\Delta \subseteq Ab$ such that $P \cup \Delta \models \mathcal{G}$ and $P \cup \Delta \models IC$.

Definition 1 (Behavioural Interface Specification). *Given a web service ws, its* behavioural interface specification $\mathcal{P}_{ws}$ *is the ALP* $\langle \mathcal{KB}_{ws}, \mathcal{E}_{ws}, \mathcal{IC}_{ws}\rangle$*, where* $\mathcal{KB}_{ws}$ *is ws's Knowledge Base,* $\mathcal{E}_{ws}$ *is ws's set of abducible predicates, and* $\mathcal{IC}s_{ws}$ *is ws's set of Integrity Constraints.*

$\mathcal{E}_{ws}$ includes predicates not defined in $\mathcal{KB}_{ws}$, as well as expectations.

$\mathcal{KB}_{ws}$ is a set of clauses which declaratively specifies pieces of knowledge of the web service. In WAV^e, clauses can contain abducible literals (with signature in $\mathcal{E}_{ws}$), as well as constraints à la Constraint Logic Programming (CLP) [13].

$$\begin{aligned} \mathcal{IC} &::= Body \rightarrow Head \\ Body &::= (Event|Expectation)[\wedge(Event|Expectation|Atom|Constr)]^{\star} \\ Head &::= Disjunct\ [\ \vee Disjunct\ ]^{\star} \mid false \\ Disjunct &::= (Expectation \mid Constr)[\ \wedge (Expectation \mid Constr \mid Atom)]^{\star} \end{aligned} \tag{1}$$

Integrity Constraints ($\mathcal{IC}s$) are forward rules, that can involve the various types of literals in our language, namely expectations, happened events, literals of predicates defined in the $\mathcal{KB}$, other abducible predicates, and CLP constraints. The syntax of $\mathcal{IC}_{ws}$ (Eq. 1) is a modification of the integrity constraints in the $\mathcal{S}$CIFF language [7]. In particular, in WAV^e each expectation is labelled with the name of the web service that is expecting the event. Happened events ($\mathbf{H}$) are always acquired from the external in $\mathcal{S}$CIFF, while in WAV^e they are abducible during reasoning phase. Intuitively, the operational behaviour of $\mathcal{IC}$s is similar to forward rules: whenever the body is true, the head should also be proven true.

3 Modeling in WAV^e

Let us consider the following running example, showing how the involved services are modeled in WAV^e. *Evelyn* is a customer who wants to obtain an electronic book by tomorrow, encrypted with algorithm *best*; she can pay cash or by credit card (*cc*), and knows two shops potentially able to satisfy her requirements.

The first shop accepts payments only with credit card and supports the encryption of goods. In our syntax, we can express that if a request arrives, then the shop will plan to reply asking for a payment, and expect the customer to pay for the good. Thus, the *eShop*1 raises an expectation about its own behaviour (I should ask for money), and one about the behaviour of the peer (you should pay)[1]:

$$\begin{aligned} &\mathbf{H}(X, eShop1, request(Item, enc(Alg)), T_s)\\ \rightarrow &\mathbf{E}_{eShop1}(eShop1, X, ask(pay(Item, cc)), T_a)\\ \wedge &\mathbf{E}_{eShop1}(X, eShop1, pay(Item, cc), T_{cc}). \end{aligned} \tag{eShop1.1}$$

If the shop received the money at least 48 hours earlier, it will deliver the item:

$$\begin{aligned} &\mathbf{H}(X, eShop1, request(Item, enc(Alg)), T_s)\\ &\wedge\ \mathbf{H}(X, eShop1, pay(Item, How), T_p)\\ \rightarrow &\mathbf{E}_{eShop1}(eShop1, X, deliver(Item, enc(Alg)), T_s), T_p + 48 < T_s \end{aligned} \tag{eShop1.2}$$

*eShop*2 accepts payments either by cash or credit card:

$$\begin{aligned} &\mathbf{H}(X, eShop2, request(Item, enc(Alg)), T_s)\\ \rightarrow &\mathbf{E}_{eShop2}(X, eShop2, pay(Item, How), T_p), How{::}[cc, cash]\\ \wedge &\mathbf{E}_{eShop2}(X, eShop2, pay(Item, How), T_{cc}). \end{aligned} \tag{eShop2.1}$$

Furthermore, it delivers goods in encrypted form only if the client has paid with credit card:

$$\begin{aligned} &\mathbf{H}(X, eShop2, pay(Item, cash), T_p)\\ \rightarrow &\mathbf{E}_{eShop2}(eShop2, X, deliver(Item, enc(none)), T_s) \end{aligned} \tag{eShop2.2}$$

$$\begin{aligned} &\mathbf{H}(X, eShop2, pay(Item, cc), T_p)\\ \rightarrow &\mathbf{E}_{eShop2}(eShop2, X, deliver(Item, enc(best)), T_s) \end{aligned} \tag{eShop2.3}$$

In this simple example, *Evelyn* knows the two shops and their URL. In a real world situation, the addresses could be collected from a yellow-pages service, or by advertisements broadcasted by the shops or sent directly to *Evelyn*. The known services, togheter with their corresponding URL, can be recorded in *Evelyn*'s $\mathcal{KB}$ by using a list of facts of the type $known\_service(Service, URL)$ (e.g. $known\_service(eShop1, "http://www.eShop1.com"))$.

Evelyn's goal is to find a web service that provides her the book within 24 hours

$$\mathcal{G}_{evelyn} = \mathbf{E}_{evelyn}(S, evelyn, deliver(book, enc(best)), T), T \leq 24. \tag{Goal}$$

Evelyn's $\mathcal{IC}$s say that upon request of payment, she will perform the payment, either by credit card or by cash:

$$\begin{aligned} &\mathbf{H}(X, evelyn, ask(pay(Item, How)), T_p)\\ \rightarrow &\mathbf{E}_{evelyn}(evelyn, X, pay(Item, How), T_p), How{::}[cc, cash] \end{aligned} \tag{evelyn1}$$

[1] The symbol "::" represents a domain constraint.

Moreover, *Evelyn* has a plan about how she could get an item; if she wants an item to be delivered her, she should *request* someone to deliver it:

$$\begin{aligned}&\mathbf{E}_{evelyn}(S, evelyn, deliver(Item, enc(How)), T_d)\\ \rightarrow&\mathbf{E}_{evelyn}(evelyn, S, request(Item, enc(How)), T_r), T_r < T_d,\\ &find\_conformant(S).\end{aligned} \qquad \text{(evelyn2)}$$

Predicate $find\_conformant$ is also defined in *Evelyn*'s $\mathcal{KB}$:

$$\begin{aligned}find\_conformant(Service) \leftarrow\ & known\_service(Service, URL),\\ & download(URL, ICS),\ impose\_ics(ICS).\end{aligned}$$

Primitive *download* retrieves information from the web (and can be implemented, e.g., with the PiLLoW library [12]). In our framework, web services expose their behavioural interface on the web, so in this case *Evelyn* downloads the $\mathcal{IC}$s of the peer she wants to interact with. Finally, *impose_ics* is a meta-predicate that adds a set of implications to the current set of $\mathcal{IC}$s, and is used by *Evelyn* to put its own policies togheter with those of the other peer.

4 Declarative and Operational Semantics

We assume that all web services have their own behavioural interface specified in the language of $\mathcal{IC}$s. This behavioural interface could be thought of as an extension of WSDL, that can be used by other web services to reason about the specifications, or to check if inter-operability is possible.

The web service initiating the interaction has a goal $\mathcal{G}$, which is a given state of affairs. Typical goals are to access resources, retrieve information, or obtain services from another web service. $\mathcal{G}$ can be any conjunction of expectations, CLP constraints, and any other literals, in the syntax of $\mathcal{IC}_{ws}$ *Disjuncts* (Eq. 1).

A web service ws reasons about the possibility to achieve a goal $\mathcal{G}$ by interacting with a peer ws' using $\mathcal{KB}_{ws}$, $\mathcal{IC}_{ws}$, $\mathcal{G}$, and the information obtained about ws''s policies, $\mathcal{IC}_{ws'}$ (Fig. 1). The idea is to obtain, through abductive reasoning, a possible course of events that together with $\mathcal{KB}_{ws}$ entails $\mathcal{IC}_{ws} \cup \mathcal{IC}_{ws'}$ and $\mathcal{G}$.

Definition 2 (Possible interaction about $\mathcal{G}$). *A possible interaction about a goal $\mathcal{G}$ between two web services ws and ws' is an $\mathcal{A}$-minimal [6] set* $\mathbf{HAP} \cup \mathbf{EXP} \cup \Delta A$ *such that Eq. 2, 3 and 4 hold:*

$$\mathcal{KB}_{ws} \cup \mathbf{HAP} \cup \mathbf{EXP} \cup \Delta A \models \mathcal{G} \qquad (2)$$

$$\mathcal{KB}_{ws} \cup \mathbf{HAP} \cup \mathbf{EXP} \cup \Delta A \models \mathcal{IC}_{ws} \cup \mathcal{IC}_{ws'} \qquad (3)$$

$$\mathcal{KB}_{ws} \cup \mathbf{HAP} \cup \mathbf{EXP} \cup \Delta A \models \mathbf{E}_X(X, Receiver, Action, Time) \rightarrow \mathbf{H}(X, Receiver, Action, Time). \qquad (4)$$

where $\mathbf{HAP}$ *is a conjunction of* $\mathbf{H}$ *atoms,* $\mathbf{EXP}$ *a conjunction of* $\mathbf{E}$ *atoms, and* ΔA *a conjunction of abducible atoms.*

We ground the notion of entailment on a model theoretic semantics defined for Abductive Disjunctive Logic Programs [6], a slight modification of the semantics presented in [17]. Rule 4 means that we assume all the web services will behave rationally, i.e., they will perform all actions that fulfil their own expectations.

Note that currently in our framework web services do not expose their knowledge base, but only the integrity constraints. However, in general integrity constraints can involve predicates defined in the $\mathcal{KB}$. In this case, the web service ws that reasons upon the specifications of ws' will make hypotheses on the possible truth value of the predicates defined in the (unknown) $\mathcal{KB}_{ws'}$; such hypotheses are abduced and recorded in the set ΔA. [2]

Among all possible interactions about $\mathcal{G}$, some of them are fruitful, and some are not. An interaction only based on expectations which are not matched by corresponding events is not fruitful: for example, the goal of ws might not have a corresponding event, thus $\mathcal{G}$ is not actually reached, but only *expected.* Or, one of the web services could be waiting for a message from the other fellow, which will never arrive, thus undermining the inter-operability.

We select, among the possible interactions, those whose history satisfies all the expectations of both the web services. After the abductive phase, we have a verification phase in which there are no abducibles, and in which the previously abduced predicates **H** and **E** are now considered as defined by atoms in **HAP** and **EXP**, and they have to match. If there is a possible interaction satisfying all expectations, then ws has found a sequence of actions that obtains the goal.

Definition 3 (Possible interaction achieving $\mathcal{G}$). *Given two web services, ws and ws', and a goal $\mathcal{G}$, a* possible interaction achieving $\mathcal{G}$ *is a possible interaction about $\mathcal{G}$ satisfying (for all $X \in \{ws, ws'\}$)*

$$\mathbf{HAP} \cup \mathbf{EXP} \models \mathbf{E}_X(S, R, Action, T) \leftrightarrow \mathbf{H}(S, R, Action, T) \quad (5)$$

Intuitively, the "$\rightarrow$" implication in Eq. 5 avoids situations in which a web service waits for an event that the peer will never produce. The "$\leftarrow$" implication avoids that one web service sends unexpected messages, which in the best case may not be understood (and in the worst cases may lead to faulty behaviour).

4.1 Operational Semantics

The operational semantics is a modification of the $\mathcal{S}$CIFF proof-procedure [7]. $\mathcal{S}$CIFF was initially developed to specify and verify agent interaction protocols in open environments. It processes events drawing from **HAP** and abduces expectations, checking that all of them are fulfilled by a happened event.

WAV^e extends $\mathcal{S}$CIFF and abduces **H** events as well as expectations. The events history is not taken as input, but all possible interactions are hypothesised. Moreover, in WAV^e events not matched by an expectation (accepatble in an open scenario) cannot be part of a *possible interaction achieving* the goal. For this reason, in WAV^e a new transition labels each **H** events with an *expected* flag as soon as a matching expectation is abduced. At the end of the derivation,

[2] Possibly, the result of the abductive phase can be sent to the peer ws', that can accept or refuse such a proposal. In other words, a contracting phase could be initiated.

unflagged **H** will cause failure. Also, in WAV^e new $\mathcal{IC}$s can be dinamically added. A transition accounts for this need: if the selected literal is $impose\_ics(X)$ it adds the set X to the $\mathcal{IC}$s.

Finally, note that soundness and completeness results, proven for the $\mathcal{S}$CIFF proof-procedure under reasonable assumptions, also hold for WAV^e. In particular, adding dynamically new $\mathcal{IC}$s can be performed in $\mathcal{S}$CIFF, because the success nodes do not change if $\mathcal{IC}$s are dynamically added with respect to the case in which they are stated from the beginning of the derivation. [3]

5 Verification

WAV^e supports different types of verification, using the same description of web services in terms of $\mathcal{IC}s$. For space reasons, we will consider only the a-priori verification, in which web services check whether there exists a possible interaction for obtaining the desired goal. After finding the possible interactions achieving its goal, the service can submit them to the other party, to establish an agreement, which could be considered as a contract, where the allowed interactions are (implicitly) listed. At this step both web services know which are the approved communications, so if they stick to what has been agreed the interaction should be successful. However, at execution time violations could always happen: on-the-fly verification aims at finding such possible violations. We have addressed this issue in [5], where the same web service specification is used to verify if the interacting parties actually behave in a conformant manner.

5.1 A-Priori Verification

Starting from her goal (Eq. Goal), *Evelyn* abduces that she wants the electronic *book* delivered to her within one day (24 hours):

$$\mathbf{E}_{evelyn}(S, evelyn, deliver(book, enc(best)), T), T \leq 24. \tag{6}$$

This expectation triggers the Rule evelyn2, and another expectation is abduced. By *rationality* (Eq. 4), such expectation becomes a happened event

$$\mathbf{H}(evelyn, S, request(book, enc(best)), T_r), T_r < 24. \tag{7}$$

Now, *Evelyn* invokes $find\_conformant$, that will choose one of the shops, download its interface, and test if it is conformant. Let us suppose to start with *eShop*1: rule eShop1.1 will trigger, as its antecedent is true because of event (7).

*eShop*1 is thus supposed to generate two expectations: it will ask *Evelyn* to pay by *cc*, and will expect *Evelyn* to do it. Again, by rationality, the expectation

[3] One way to see this property is using a lemma of the soundness theorem [7]. To prove that an $\mathcal{IC}$ can be added dynamically, it is enough to insert in the body a fictitious event and add such event dynamically. Propagation of this $\mathcal{IC}$ is thus delayed until such event occurs. The effect is the same as adding the $\mathcal{IC}$ dynamically.

of *eShop*1 about its own behaviour becomes a happened event, and *Evelyn* will react to it by performing the payment:

$$\mathbf{H}(eShop1, evelyn, ask(pay(book, cc)), T_a) \wedge How :: [cc, cash]$$
$$\mathbf{H}(evelyn, eShop1, pay(book, cc), T_p).$$

*eShop*1 can now trigger its Rule eShop1.2, generating an expectation about its own behaviour, that will be translated by *rationality* into the event:

$$\mathbf{H}(eShop1, evelyn, deliver(book, enc(best)), T_s) \wedge T_p + 48 < T_s.$$

Now the proof-procedure tries to match such event with *Evelyn*'s expectation (6). The propagation of CLP constraints infers $T_p < -24$, reminding *Evelyn* that she should have made her request one day earlier. The proof-procedure signals a deadline violation: there is no way to obtain the book on time from *eShop*1.

Evelyn can now download the behavioural interface of *eShop*2. Since the behaviour of *eShop*2 depends on the chosen payment method, we have two possible interactions. In the first one she pays by cash, obtaining the following history:

$$\mathbf{H}(evelyn, eShop2, request(book, enc(best)), T_r), T_r < 24$$
$$\mathbf{H}(eShop2, evelyn, ask(pay(book, cash)), T_a)$$
$$\mathbf{H}(evelyn, eShop2, pay(book, cash), T_p)$$
$$\mathbf{H}(eShop2, evelyn, deliver(book, enc(none)), T_s).$$

This time there are no missed deadlines, but the book is sent unencrypted: *Evelyn*'s expectation (6) is not matched by any event. Luckily, *Evelyn* has another branch to explore, i.e. the one in which she actually pays by *cc*. In this case, *eShop*2 will use the *best* algorithm (rule eShop2.3): the generated history satisfies all expectations of both peers, thus *eShop*2 is considered conformant.

6 Rule Mark-Up

In WAV^e, the $\mathcal{IC}$s can be exchanged between web services, as well as advertised together with their WSDL. As the exchanged information is made of rules, the natural choice for the web-friendly interchange format is RuleML [3].

WAV^e embeds two types of rules: $\mathcal{IC}s$ and clauses. $\mathcal{IC}s$ are forward rules, used to react to events and generate new expectations. Clauses are backward rules, used to plan, reason upon events and perform proactive reasoning. RuleML 0.9 contains a *direction* attribute to represent both kinds of rules. Being based on abduction, WAV^e can deal both with negation as failure and negation by default, that have an appropriate tagging in RuleML. In this work, we only used standard RuleML syntax; in future work we might be interested in distinguishing between defined and abducible predicates, or between expectations and events.

WAV^e was implemented in SICStus Prolog, which contains an implementation of PiLLoW [12], making it easy to access information on the web, and an XML parser, useful to easily implement a bidirectional RuleML parser.

7 Discussion and Related Work

WAV^e is a framework for defining declaratively the behavioural interface of web services, and for testing the possibility of fruitful interaction between them. It uses and extends a technology initially developed for online compliance verification of agent interaction to protocols [7]. The extension of $\mathcal{S}$CIFF to the context of web services, centering around the concept of policies seems very promising.

In a companion paper [6] we propose the use of $\mathcal{S}$CIFF in the context of discovery engines. We present a fundamentally different architecture, in which a third party (i.e. a discovery engine) reasons on behalf of the requesting web service. Specifically, we focus on the "contracting" stage of service discovery, in which, following ontological matchmaking, the third party needs to understand if there exists a concrete interaction between the "requestor" and a "provider" web service that achieves a given requestor's objective. D ifferently from what we show here, such interaction is not defined based on a "total expectation" concept (see Section 4, Eq. 5), but it may include "unexpected" events, which leads to a different semantics. This is due to the different architecture, in which the third party has to reason under the assumption of incomplete knowledge – thus even sequences of events that are not totally expected by the third party may lead to achieving the requestor's objective. We are working on the combination of the two proposed approaches into a unified architecture.

The idea of policies for web services and policy-based reasoning is also adopted by many other authors, among which Finin et al. [14], and Bradshaw et al. [18]. The first has an emphasis on representation of actions, the latter on the deontic semantic aspects of web service interaction. Previous work on $\mathcal{S}$CIFF addressed the links between deontic operators and expectation-based reasoning [8].

The outcome of the WAV^e reasoning process could be intended as a sort of "contract agreement", provided that each peer is tightly bounded to the policies it has previously published. The dynamic agreement on contracts (e-contracting) is addressed in [10], where Situated Courteous Logic is adopted for reasoning about rules that define business provisions policies.

In this work we mainly focus on the reasoning process upon the policies of both the peers, without considering ontologies. Many other approaches focus on the latter issue (as for example OWL-S [2]), hence our proposal could be seen as a complementary functionality. In [1] it is proposed a language for semantic web service specification (using logic), and a notion of *mediator* is introduced to overcome differences between ontologies. In [16], the authors present a framework for automated web service discovery that uses the Web Service Modeling Ontology (WSMO) as the conceptual model, and distinguishes between a discovery phase and a contracting phase. Both the approaches perform hypothetical reasoning; however, in [16,1], only the client's goal is considered, while in WAV^e also behavioural interfaces are taken into account. Therefore, our framework can be exploited to verify interoperability between behavioural interfaces [4].

Acknowledgements. This work has been partially supported by the MIUR PRIN 2005 projects *Specification and verification of agent interaction proto-*

cols and *Vincoli e preferenze come formalismo unificante per l'analisi di sistemi informatici e la soluzione di problemi reali*, and by the MIUR FIRB project *Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet*.

References

1. http://www.w3.org/Submission/SWSF-SWSL
2. OWL-S. http://www.daml.org/services/owl-s
3. Adi, A., Stoutenburg, S., Tabet, S. (eds.): RuleML 2005. LNCS, vol. 3791. Springer, Heidelberg (2005)
4. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M.: An abductive framework for a-priori verification of web services. In: PPDP (2006)
5. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M., Storari, S., Torroni, P.: In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, Springer, Heidelberg (2006)
6. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M., Torroni, P.: Contracting for dynamic location of web services: specification and reasoning with SCIFF. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC. LNCS, Springer, Heidelberg (2007)
7. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: the SCIFF framework. ACM Transactions on Computational Logics. Accepted for publication.
8. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Sartor, G., Torroni, P.: Mapping deontic operators to abductive expectations. Computational and Mathematical Organization Theory 12(2–3), 205–225 (2006)
9. Alferes, J., Damásio, C., Pereira, L.: Semantic web logic programming tools. In: Bry, F., Henze, N., Małuszyński, J. (eds.) PPSWR 2003. LNCS, vol. 2901, pp. 16–32. Springer, Heidelberg (2003)
10. Bhansali, S., Grosof, N.: Extending the sweetdeal approach for e-procurement using sweetrules and ruleml. In: Adi. et al. [3]
11. Bry, F., Eckert, M.: Twelve theses on reactive rules for the web. In: Proc. of the Workshop on Reactivity on the Web, Munich, Germany (March 2006)
12. Gras, D., Hermenegildo, M.: Distributed WWW programming using CiaoProlog and the PiLLoW library. TPLP 1(3), 251–282 (2001)
13. Jaffar, J., Maher, M.: Constraint logic programming: a survey. Journal of Logic Programming 19–20, 503–582 (1994)
14. Kagal, L., Finin, T.W., Joshi, A.: A policy based approach to security for the semantic web. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, Springer, Heidelberg (2003)
15. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive Logic Programming. Journal of Logic and Computation 2(6), 719–770 (1993)
16. Kifer, M., Lara, R., Polleres, A., Zhao, C., Keller, U., Lausen, H., Fensel, D.: A logical framework for web service discovery. In: ISWC Worshop, Hiroshima (2004)
17. Sakama, C., Inoue, K.: Abductive logic programming and disjunctive logic programming: their relationship and transferability. Journal of Logic Programming 44(1-3), 75–100 (2000)
18. Uszok, A., Bradshaw, J., Jeffers, R., Tate, A., Dalton, J.: Applying KAoS services to ensure policy compliance for semantic web services workflow composition and enactment. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 425–440. Springer, Heidelberg (2004)