Policy-based reasoning for smart web service interaction

Marco Alberti
Marco Gavanelli
Evelina Lamma
ENDIF, Università di Ferrara
{marco.alberti|marco.gavanelli|evelina.lamma}@unife.it

Federico Chesani
Paola Mello
Marco Montali
Paolo Torroni
DEIS, Università di Bologna
{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 interface behaviour 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 WAV^e.**

NOTE

This article is a modified version of [5].

I. 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, and by the success of its predecessor, Object Oriented programming paradigm. 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 then 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. It is the philosophy followed, e.g., by BPEL [11]: composing new applications through existing web services.

If we adopt the SOC programming paradigm, how to exploit the potential of a growing base of web services becomes one of our strategic issue. In a domain in which being more competitive means knowing more and using all available information at best, how shall we cope with the proliferation of new services? How shall we decide to use a web service rather than another one? when new ones become available, shall we go for them? are there new opportunities that were not there before? It is a necessary, never-ending, heavy and thus potentially very costly decision process, but it could also be very rewarding, if we had the proper tools.

A partial answer to these questions is given by service discovery. As new services become available, they are published, for instance by registration on some yellow-pages server; existing services can then become aware of the new ones and exploit them. This solves part of the problem: as through discovery we only know that there are some services, which possibly follow some standards, but understanding whether interacting with them will be profitable or detrimental, is far from being a trivial question. For one, it is not possible to think to try and invoke all newly discovered services and analyze the results. Beside being highly error-prone, such a method would require expensive rollbacks that are often unaffordable at run-time. Thus, alternative approaches have to be developed. This is what we intend to address in this article.

The focus of this article is the following problem: how to dynamically understand if two web services can inter-operate, without them having a-priori knowledge of each other's capabilities, but by reasoning about policies exchanged at run-time.

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, without resorting to trial and error. We envisage a two-phase discovery activity on the side of web services. First, web services collect information about other web services, and try and understand by reasoning which ones can lead to a fruitful interaction. This activity is carried out off-line, beforehand. Then they use the available information to interact with each other. It is the same philosophy of search engines: before, collect information through web spiders, then use it when requested by the user.

In this article we focus on the reasoning involved in the off-line phase, assuming that a new web service has been found, and we must decide about the possibility to interact with it. We assume that each web service publishes, alongside with its WSDL, its *interface behaviour specifications*. By reasoning on the information available about other web services' interface behaviour, each web service can verify which goals can be reached by interaction.

To achieve our vision, we propose a proof theoretic approach, based on computational logic – in fact, on abductive logic programming. In particular, we formalise policies for web services in a declarative language which is a modification of the $\mathcal{S}$CIFF language originally defined in the context of the EU IST-2001-32530 project, to specify and verify social-level agent interaction.

In this new language, policies can be defined by way

of *social integrity constraints* (*IC*s): a sort of reactive rules used to generate and reason about expectations about possible evolutions of a given interaction setting.

As claimed in [14], a rule-based approach to reactivity on the Web provides the following benefits over the conventional approach:

- Rules are easy to understand for humans. Requirements specifications often already comes in the for of rules expressed in a natural or formal language;
- Rule-based specifications are flexible and easy to adapt;
- Rules are well-suited for processing and analyzing by machines (verification, transformation);
- Rules can be managed in a single knowledge base or in several knowledge bases possibly distributed over the Web.

Moreover, we believe that, as advocated by Alferes et al. [10], an approach based on logic programming allows to express knowledge in form of rules, and to make inference with those rules. Like the authors, we follow Tim Berners-Lee et al. [12] in considering logic a natural conceptual and computational tool for the Semantic Web (“*Adding logic to the Web - the means to use rules to make inferences, choose courses of action and answer questions - is the task before the Semantic Web community at the moment*”)

Based on the $\mathcal{S}$CIFF framework we propose a new declarative semantics and a new proof-procedure that combines forward, reactive reasoning with backward, goal-oriented reasoning, and is tailored to the discovery activity’s off-line phase’s verification problem. We have called this new framework WAV^e(Web-service Abductive Verification).

In order to support the exchange of rules between web services in a standard format, we also propose a RuleML encoding for our language.

We start by showing the abstract architecture of WAV^e. In Sect. III we introduce a running on-line shopping scenario. In Sect. IV, we briefly introduce the language used in the framework, and in Sect. V we show how the scenario can be modeled in WAV^e in terms of *IC*s. Sect. VI presents the declarative and operational semantics of WAV^e, and Sect. VII proposes the application of WAV^e to the verification problem in the reference scenario. Sect. VIII discusses the encoding of WAV^e rules in RuleML. A brief discussion, also with respect to related work, follows.

II. The Architecture of WAV^e

Fig. 1 depicts our general reference architecture. Arrows indicate the flow of policies between web services. The layered architecture of a web service, e.g. ws, has WAV^e at the top of the stack, performing reasoning based on its own knowledge and on the policies obtained from other web services, e.g. ws'. The functionalities of the various elements of the knowledge will be explained in Sect. IV. For the moment, we say that policies are identified with the $\mathcal{IC}_{ws}$ component. The architecture is symmetric. We represented with thick borders the modules involved in the operations carried out by ws, and its output. In order for ws' to pass $\mathcal{IC}_{ws'}$ on to ws (and vice versa), a Rule Interchange Format (RIF) is adopted. One possibility for such a RIF could be RuleML [3]. Finally, as a result of the reasoning activity, ws produces an answer $\mathcal{C}$ to the question: “is it possible to inter-operate with ws' and achieve goal $\mathcal{G}_{ws}$?”

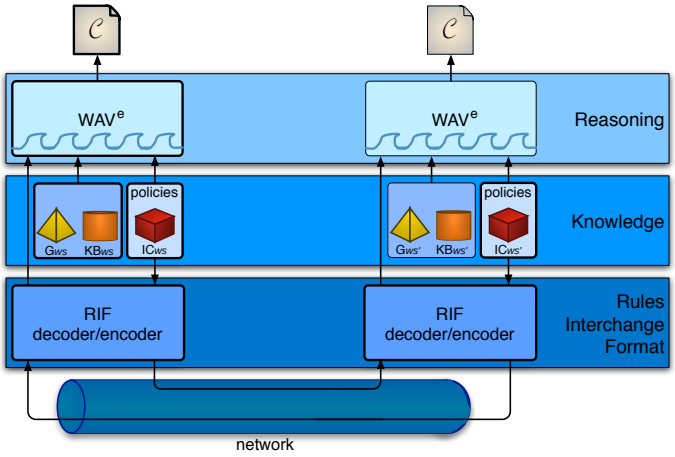

Fig. 1. The architecture of WAV^e

Fig. 1 does not show control elements, but only information flows. We assume that suitable interaction protocols are defined to control the flow of information (e.g. policies) between the web services. In particular, in a more comprehensive setting, ws and ws' could negotiate the exchange of policies in an incremental way, or could use the result $\mathcal{C}$ of this reasoning activity to perform the second, on-line phase of service interaction we mentioned in the introduction. All this is outside of this picture, and of this article’s scope.

III. The $alice$ & $eShop$ Scenario

This scenario is inspired to the one described by the Working Group on Rule Interchange Format [25]. A similar scenario is also in [14]. We consider two entities, which we call $alice$ and $eShop$.[1] $eShop$ is a web service which sells devices. $alice$ is another web service which instead needs to obtain a device, and which is considering buying it from $eShop$. $alice$ and $eShop$ describe their behaviour concerning sales/payment/... of items through policies, specified as rules, which they publish using some RIF.

Before $alice$ buys an item from $eShop$, $alice$ checks whether her policies and $eShop$’s policies are compatible, i.e., if they allow a successful transaction. During this process, it turns out that $eShop$ accepts credit card payments, besides other payment methods, and that $alice$ can only pay by credit card; in this case, in order to proceed with the payment, she requires evidence of the shop’s membership to some trusted “Better Business Bureau” ($\mathcal{BBB}$) association. We assume that the shop is able and ready to provide such a piece of evidence. We can thus define $eShop$’s and $alice$’s policies as follows:

[1]In this simplified scenario, we identify $alice$ and $eShop$ with their representative software counterparts which will carry out transactions on their behalf.

(shop1) if a customer wishes to buy an item, then (s)he should pay it either by credit card, or by cash, or by cheque;

(shop2) if a customer wishes to buy an item, and (s)he has paid it either by credit card, or by cash, or by cheque, then $eShop$ will deliver the item;

(shop3) if a customer wishes to receive a certificate about $eShop$'s membership to the $\mathcal{BBB}$, then the shop will send it;

(alice1) if a shop requires that $alice$ pays by credit card, $alice$ expects that the shop provides evidence of its membership to the $\mathcal{BBB}$;

(alice2) if a shop requires that $alice$ pays by credit card, and the shop has provided evidence of its membership to the $\mathcal{BBB}$, then $alice$ will pay by credit card;

In this example, we can identify two kinds of policy rules. shop1 and alice1 express requirements, i.e., what is needed in order to proceed with accomplishing some request. shop2, shop3 and alice2 represent the effect of requests, i.e., they tell what has to be expected if some conditions hold and some request is received.

Using this scenario, we want to demonstrate the possibility of reaching an agreement through rules exchange. Besides, we want to show how policies support backward and forward reasoning, in the following way. Backward, pro-active reasoning starts from goals to produce (expectations about) actions or events that should be generated in order to achieve the goals. Forward, reactive reasoning starts from events and is used to generate (expectations about) actions that represent reactions to such events.

In this scenario, the goal of $alice$ interacting with $eShop$ is to obtain an item from $eShop$. Actions are all the messages exchanged between the two web services.

The steps that we envisage are as follows:

1) $alice$ wants to obtain a device. She knows that she can have it if $eShop$ delivers it to her. Thus, she sends $eShop$ a request, by which she wants to know $eShop$'s policies regarding the delivery of that device;
2) $eShop$ considers $alice$'s request, and composes a set of rules related to $alice$'s request (its policies), possibly deriving/filtering them from a larger set. In this example, the set contains shop1, shop2, and shop3. Once such a set is put together, $eShop$ communicates it to $alice$;
3) $alice$ reasons on (1) her goal, (2) her own policies (alice1 and alice2), and (3) $eShop$'s policies. Two are the possible outcomes:
   - either $alice$ infers that she and $eShop$ can have a successful transaction that satisfies each other's policies and that achieves her goal,
   - or $alice$ infers that there is no such a possibility.
4) possibly, at a later point, $alice$ and $eShop$ may engage in a transaction which (hopefully) makes $alice$ achieve her goal.

Points (1) through (3) represent the off-line phase of service discovery/interaction, whereas point (4) represent the actual transaction occurring between $alice$ and $eShop$. The reasoning involved in (3) is the subject of this article.

IV. The WAV^e Framework

In WAV^e, the observable behaviour of the web services is represented by *events*. Since we focus on (explicit) interaction between web services, events always represent exchanged messages.

WAV^e considers two types of events: those that one can control and those that one cannot. Typically, from the standpoint of a web service ws, an event such as a message generated by ws himself will fall into the first category, a message that ws is expecting from another fellow web service ws' will fall instead into the second one. We use two different functors to keep these two categories of messages distinct from each other. Atoms denoted by functor $\mathbf{H}$ will stand for events that a web service expects to be producing itself; atoms denoted by functor $\mathbf{E}$ will stand for events that a web service is expecting, and over which it does not have any control. Since WAV^e is about reasoning on possible future courses of events, both kinds of events represent *hypotheses* that a web service can make on possibly happening events. The notation is: $\mathbf{H}(ws, ws', M, T)$, for messages ($M$) that a web service ws is expecting to send to ws' at time T, and $\mathbf{E}(ws', ws, M, T)$ for messages (M) expected by ws from ws' for time T.

Web service specifications in WAV^e are relations among expected events, expressed by an Abductive Logic Program (ALP). In general, an ALP [21] is a triplet $\langle P, A, IC\rangle$, where P is a logic program, A is a set of predicates named *abducibles*, and IC is a set of integrity constraints. Roughly speaking, the role of P is to define predicates, the role of A is to fill-in the parts of P which are unknown, and the role of IC is to constrain the ways elements of A are hypothesised, or "abduced". Reasoning in abductive logic programming is usually goal-directed (being G a goal), and it accounts to finding a set of abduced hypotheses Δ built from predicates in A such that $P \cup \Delta \models G$ and $P \cup \Delta \models IC$. In the past, a number of proof-procedures have been proposed to compute Δ (see Kakas and Mancarella [22], Fung and Kowalski [17], Denecker and De Schreye [15], etc.).

Definition 4.1 (Web service interface behaviour specification): Given a web service ws, its *web service interface behaviour* specification $\mathcal{P}_{ws}$ is an ALP, represented by the triplet

$$\mathcal{P}_{ws} \equiv \langle \mathcal{KB}_{ws}, \mathcal{E}_{ws}, \mathcal{IC}_{ws}\rangle$$

where:

- $\mathcal{KB}_{ws}$ is ws's *Knowledge Base*,
- $\mathcal{E}_{ws}$ is ws's set of *abducible predicates*, and
- $\mathcal{IC}_{ws}$ is ws's set of *Integrity Constraints*.

$\mathcal{KB}_{ws}$ is a set of clauses which declaratively specifies pieces of knowledge of the web service. Note that the body of $\mathcal{KB}_{ws}$'s clauses may contain $\mathbf{E}$ expectations about the behaviour of the web services, as defined above. $\mathcal{KB}_{ws}$'s syntax is summarised in Eq. (1).

$$
\begin{array}{rcl}
\mathcal{KB}_{ws} & ::= & [\ Clause\ ]^\star \\
Clause & ::= & Atom \leftarrow Cond \\
Cond & ::= & ExtLiteral\ [\ \wedge ExtLiteral\ ]^\star \\
ExtLiteral & ::= & Atom \mid true \mid Expect \mid Constr \\
Expect & ::= & \mathbf{E}(Atom, Atom, Atom, Atom)
\end{array} \quad (1)
$$

$\mathcal{E}_{ws}$ includes $\mathbf{E}$ expectations, $\mathbf{H}$ events, and predicates not defined in $\mathcal{KB}_{ws}$.

$$
\begin{array}{rcl}
\mathcal{IC}_{ws} & ::= & [\ IC\ ]^\star \\
IC & ::= & Body \rightarrow Head \\
Body & ::= & (Event \mid Expect)\ [\wedge BodyLit]^\star \\
BodyLit & ::= & Event \mid Expect \mid Atom \mid Constr \\
Head & ::= & Disjunct\ [\ \vee Disjunct\ ]^\star \mid false \\
Disjunct & ::= & (Expect \mid Event \mid Constr) \\
 & & [\ \wedge (Expect \mid Event \mid Constr)]^\star \\
Expect & ::= & \mathbf{E}(Atom, Atom, Atom, Atom) \\
Event & ::= & \mathbf{H}(Atom, Atom, Atom, Atom)
\end{array} \quad (2)
$$

Integrity Constraints (*ICs*) are forward rules, of the form *Body* $\rightarrow$ *Head* (Eq. (2)). The *Body* of *ICs* is a conjunction of literals and expected events; the *Head* instead is a disjunction of conjunctions of expectations, events and literals, or *false*. The syntax of $\mathcal{IC}_{ws}$ is a modification of the integrity constraints in the $\mathcal{S}$CIFF language [6]. In particular, unlike $\mathcal{S}$CIFF, WAV^e treats $\mathbf{H}$ events as abducible predicates, and as such it allows them to occur in the *Head* of integrity constraints; however, this initial version of WAV^e does not yet accommodate negative expectations nor negation ($\neg$). We intend to consider these two features in future extensions of WAV^e.

Intuitively, the operational behaviour of integrity constraints is similar to forward rules: whenever the body becomes true, the head is also made true.

V. Modeling in WAV^e

In this section, we demonstrate web service policy modelling in WAV^e by showing the specification of $alice$ and $eShop$. The first three rules represent $eShop$'s policies.

$$
\begin{array}{l}
\mathbf{E}(eShop, alice, deliver(Item), T_s) \\
\rightarrow \mathbf{E}(alice, eShop, pay(Item, cc), T_{cc}) \wedge T_{cc} < T_s \\
\vee \mathbf{E}(alice, eShop, pay(Item, cash), T_{ca}) \wedge T_{ca} < T_s \\
\vee \mathbf{E}(alice, eShop, pay(Item, cheque), T_{ch}) \wedge T_{ch} < T_s
\end{array} \quad \text{(shop1)}
$$

IC shop1 says that, if $alice$ expects $eShop$ to deliver an Item, then $eShop$ expects $alice$ to pay by credit card, cash, or cheque, and that payment must be made before delivery.[2] In that case, the abducibles in the head are expectations, because they represent actions that should be performed by $alice$: from $eShop$'s viewpoint, they can only be expected.

$$
\begin{array}{l}
\mathbf{E}(eShop, alice, deliver(Item), T_s) \\
\wedge\ \mathbf{H}(alice, eShop, pay(Item, How), T_p) \wedge T_p < T_s \\
\wedge\ How{::}[cc, cash, cheque]) \\
\rightarrow \mathbf{H}(eShop, alice, deliver(Item), T_s).
\end{array} \quad \text{(shop2)}
$$

[2]The alternative in the head could alternatively be expressed via a variable with domain: $\mathbf{E}(alice, eShop, pay(Item, How), T) \wedge How{::}[cc, cash, cheque] \wedge T < T_s$, where "::" represents a domain constraint.

IC shop2 says that, if $alice$ expects $eShop$ to deliver the Item, and $alice$ has paid for it, then $eShop$ will actually deliver it to $alice$. In that case, the abducible in the head is an event, because it represents an action that $eShop$ should perform, and therefore it assumes that it will indeed happen (since it is its own responsibility).

$$
\begin{array}{l}
\mathbf{E}(eShop, alice, give\_guarantee, T_g) \\
\rightarrow \mathbf{H}(eShop, alice, give\_guarantee, T_g).
\end{array} \quad \text{(shop3)}
$$

IC shop3 says that if $alice$ expects to receive a guarantee, then $eShop$ will send it. The following two rules represent $alice$'s policies.

$$
\begin{array}{l}
\mathbf{E}(alice, eShop, pay(Item, cc), T_p) \\
\rightarrow \mathbf{E}(eShop, alice, give\_guarantee, T_g) \wedge T_g < T_p.
\end{array} \quad \text{(alice1)}
$$

IC alice1 says that, if $eShop$ expects $alice$ to pay for an Item by credit card, then $alice$ expects that $eShop$ will have provided a guarantee by the time she pays.

$$
\begin{array}{l}
\mathbf{E}(alice, eShop, pay(Item, cc), T_p) \\
\wedge\ \mathbf{H}(eShop, alice, give\_guarantee, T_g) \wedge T_g < T_p \\
\rightarrow \mathbf{H}(alice, eShop, pay(Item, cc), T_p).
\end{array} \quad \text{(alice2)}
$$

IC alice2 says that, if $eShop$ expects $alice$ to pay for an Item by credit card, and $eShop$ has provided $alice$ with a guarantee, then $alice$ will pay the Item by credit card. Finally, the following clause is part of $\mathcal{KB}_{alice}$

$$
\begin{array}{l}
have(alice, Item, T) \leftarrow \\
\mathbf{E}(eShop, alice, deliver(Item), T_d) \wedge T_d \leq T.
\end{array} \quad \text{(alice3)}
$$

Clause alice3 says that, in order for $alice$ to have an Item at time T, then $alice$ expects $eShop$ to deliver the Item by time T.

VI. Declarative and Operational Semantics

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

Another approach would be to obtain web services' interface behaviour through an appropriate request protocol, in which $\mathcal{IC}$s are (interactively) exchanged so that each web service may disclose *ad hoc*, customised information on demand.

In this work, we make the simplifying assumption that all information regarding the interface behaviour is provided at once. The web service will then try and prove that a fruitful interaction is possible based on what it receives.

The web service initiating the interaction has a goal $\mathcal{G}$, which is a given state of affairs. A typical goal could be to access a resource, to retrieve some information, or to obtain a service from another web service. $\mathcal{G}$ will often be an expectation (of obtaining a service, accessing a resource, or gathering information), but in general it can be any conjunction of expectations, CLP constraints, and any other literals, in the syntax of $\mathcal{IC}_{ws}$ *Head Disjuncts* (Eq. 2).

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

Note that we do not assume that ws knows $\mathcal{KB}_{ws'}$, as the $\mathcal{KB}$ is not part of the interface. However, in general integrity constraints can involve predicates defined in the knowledge base. For example, they can contain predicates defining parameters, deadlines, coefficients, etc., or other knowledge only available to ws'. If the interface behaviour provided by ws' involves predicates defined in $\mathcal{KB}_{ws'}$, unknown to ws, we have two alternatives:

- either ws' provides ws with the necessary information, e.g. with (part of) its $\mathcal{KB}_{ws'}$;
- or ws will have to make assumptions about such unknown predicates.

We take the second option, and consider unknowns that are neither $\mathbf{H}$ events nor $\mathbf{E}$ expectations as literals that can be abduced, and we keep them in a set Δ. We then have the following two equations that define the set of abductive answers representing possible interaction between ws and ws' achieving $\mathcal{G}$:

$$\mathcal{KB}_{ws} \cup \mathbf{HAP} \cup \mathbf{EXP} \cup \Delta \models \mathcal{G} \quad (3)$$

$$\mathcal{KB}_{ws} \cup \mathbf{HAP} \cup \mathbf{EXP} \cup \Delta \models \mathcal{IC}_{ws} \cup \mathcal{IC}_{ws'} \quad (4)$$

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

We can now proceed with defining what kind of interaction is possible/fruitful, given two web services and a goal.

Definition 6.1 (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 set $\mathbf{HAP} \cup \mathbf{EXP} \cup \Delta$ such that Eq. 3 and 4 hold.

Among all possible interactions about $\mathcal{G}$, some of them are fruitful, and some are not. An interaction only based on expectations which will not be matched by corresponding events is not a fruitful one: for example, the goal of ws might not have a corresponding event, thus the goal 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 $\mathbf{H}$ and $\mathbf{E}$ are now considered as defined by atoms in $\mathbf{HAP}$ and $\mathbf{EXP}$, and they have to match. If among the possible interactions there exists one satisfying

$$\mathbf{HAP} \cup \mathbf{EXP} \models \mathbf{E}(X, Y, Action, T) \leftrightarrow \mathbf{H}(X, Y, Action, T) \quad (5)$$

then ws has found a sequence of actions that obtains the goal $\mathcal{G}$.

Definition 6.2 (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 Eq. 5.

Intuitively, the "$\rightarrow$" implication in Eq. 5 avoids situations in which a web service waits forever for an event that the other web service 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 scenarios it may lead to faulty, unpredictable behaviour of the parties involved).

A. Operational Semantics

The operational semantics is a modification of the $\mathcal{S}$CIFF proof-procedure [9]. $\mathcal{S}$CIFF is a transition system, whose state is given by the following tuple:

$$T \equiv \langle R, CS, PSIC, \Delta A, \mathbf{PEND}, \mathbf{HAP}, \mathbf{FULF}, \mathbf{VIOL} \rangle$$

The set of expectations $\mathbf{EXP}$ is partitioned into the fulfilled ($\mathbf{FULF}$), violating ($\mathbf{VIOL}$), and pending ($\mathbf{PEND}$) expectations. The other elements are: the resolvent (R), the abduced literals that are not expectations (ΔA), the constraint store (CS), a set of implications, inherited from the IFF [17], called *partially solved integrity constraints* ($PSIC$), and the history of happened events ($\mathbf{HAP}$).

A classical application of $\mathcal{S}$CIFF is on-line checking of compliance of agent interaction to protocols. In fact, $\mathcal{S}$CIFF was initially developed to specify and verify agent interaction protocols on-the-fly, under the assumption of open agent environments adopted by other noteworthy agent research work [27]. $\mathcal{S}$CIFF processes events drawing from $\mathbf{HAP}$ and generates (abduces) expectations; then it checks that all expectation are fulfilled by at least one happened event. The declarative semantics of $\mathcal{S}$CIFF contains in fact a requirement $\mathbf{E}(X) \rightarrow \mathbf{H}(X)$ – differently from WAV^e, which has a double implication (Eq. 5). In $\mathcal{S}$CIFF, as soon as new $\mathbf{H}$ events are processed, a transition *fulfilment* labels the relevant matching expectations as *fulfilled* and moves them to the set $\mathbf{FULF}$. At the end of the derivation, if some expectation remains in the set $\mathbf{PEND}$, a failure node is generated, and other alternative branches will be explored in backtracking, if there exist any.

WAV^e extends $\mathcal{S}$CIFF and abduces $\mathbf{H}$ 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 (which are perfectly acceptable in the multi-agent scenario addressed by $\mathcal{S}$CIFF) cannot be part of a *possible interaction achieving* the goal.

The two phases in the declarative semantics (generation of possible interactions and their test for conformance) are condensed into one single derivation process, thanks to a new transition adopted in WAV^e. The *expected* transition, symmetrical to *fulfilment*, labels each $\mathbf{H}$ events with an *expected* flag as soon as an expectation matching it is abduced. At the end of the derivation, $\mathbf{H}$ with *expected* status = false will cause failure.

Otherwise, if the WAV^e derivation in a program $\mathcal{P}$ for a goal $\mathcal{G}$ succeeds with set of expectation $\mathbf{EXP} \cup \mathbf{HAP} \cup \Delta$, we write $\mathcal{P} \vdash_{\mathbf{EXP} \cup \mathbf{HAP} \cup \Delta} \mathcal{G}$.

Soundness and completeness results. WAV^e is a conservative modification of the $\mathcal{S}$CIFF proof-procedure, which is sound and complete under reasonable assumptions [7]. Therefore, soundness and completeness results also hold for WAV^e. A detailed discussion of this issue can be found in [5].

We will next demonstrate the operational functioning of verification in WAV^e in the $alice$ & $eShop$ scenario.

VII. Verification in WAV^e

In the following, the sets $\mathbf{EXP}_a^N$ and $\mathbf{HAP}_a^N$ represent the evolution of $alice$'s expectations and events as WAV^e's derivation progresses; N is an incremental index. Let g be the following goal of $alice$'s:

$$g \leftarrow have(alice, device, 50). \quad \text{(goal)}$$

Then, by unfolding of clause alice3,

$$\begin{aligned}\mathbf{EXP}_a^0 =&\{\mathbf{E}(eShop, alice, deliver(device), T_s) \wedge T_s < 50\} \\ &(by\ \text{(alice3)})\end{aligned} \tag{6}$$

To this expectation, $eShop$ will react by expecting a payment:

$$\begin{aligned}\mathbf{EXP}_a^1 =&\{\mathbf{E}(eShop, alice, deliver(device), T_s) \wedge T_s < 50 \\ &\wedge(\mathbf{E}(alice, eShop, pay(device, cc), T_{cc}) \wedge T_{cc} < T_s \\ &\vee \mathbf{E}(alice, eShop, pay(device, cash), T_{ca}) \wedge T_{ca} < T_s \\ &\vee \mathbf{E}(alice, eShop, pay(device, cheque), T_{ch}) \wedge T_{ch} < T_s) \\ &(by\ \text{(shop1)})\end{aligned} \tag{7}$$

Since the expectation containing the payment by cc is the only one which generates an expectation matching a rule of $alice$ ((alice1)), the first expectation among the three payment alternatives is selected (the other branches eventually fail by Eq. 5, because no matching $\mathbf{H}$ is abduced). This choice triggers (alice1):

$$\begin{aligned}\mathbf{EXP}_a^2 =&\{\mathbf{E}(eShop, alice, deliver(device), T_s) \wedge T_s < 50 \\ &\wedge \mathbf{E}(alice, eShop, pay(device, cc), T_{cc}) \wedge T_{cc} < T_s \\ &\wedge \mathbf{E}(eShop, alice, give\_guarantee, T_g) \wedge T_g < T_{cc}\} \\ &(by\ \text{(alice1)})\end{aligned} \tag{8}$$

Then (shop3) fires, and abduces the happening of $give\_guarantee$ event. We then have:

$$\begin{aligned}\mathbf{EXP}_a^3 =&\{\mathbf{E}(eShop, alice, deliver(device), T_s) \wedge T_s < 50 \\ &\wedge \mathbf{E}(alice, eShop, pay(device, cc), T_{cc}) \wedge T_{cc} < T_s \\ &\wedge \mathbf{E}(eShop, alice, give\_guarantee, T_g) \wedge T_g < T_{cc}\} \\ &(by\ \text{(alice1)}) \\ \mathbf{HAP}_a^3 =&\{\mathbf{H}(eShop, alice, give\_guarantee, T_g) \wedge T_g < T_{cc}\} \\ &(by\ \text{(shop3)})\end{aligned} \tag{9}$$

Given the guarantee, $alice$ will pay by credit card (rule (alice2) fires):

$$\begin{aligned}\mathbf{EXP}_a^4 =&\{\mathbf{E}(eShop, alice, deliver(device), T_s) \wedge T_s < 50 \\ &\wedge \mathbf{E}(alice, eShop, pay(device, cc), T_{cc}) \wedge T_{cc} < T_s \\ &\wedge \mathbf{E}(eShop, alice, give\_guarantee, T_g) \wedge T_g < T_{cc} \\ \mathbf{HAP}_a^4 =&\{\mathbf{H}(eShop, alice, give\_guarantee, T_g) \wedge T_g < T_{cc} \\ &\wedge \mathbf{H}(alice, eShop, pay(device, cc), T_{cc}) \wedge T_{cc} < T_s\} \\ &(by\ \text{(alice2)})\end{aligned} \tag{10}$$

Having received the payment, $eShop$'s policy would be to deliver the device:

$$\begin{aligned}\mathbf{EXP}_a^5 =&\{\mathbf{E}(eShop, alice, deliver(device), T_s) \wedge T_s < 50 \\ &\wedge \mathbf{E}(alice, eShop, pay(device, cc), T_{cc}) \wedge T_{cc} < T_s \\ &\wedge \mathbf{E}(eShop, alice, give\_guarantee, T_g) \wedge T_g < T_{cc}\} \\ \mathbf{HAP}_a^5 =&\{\mathbf{H}(eShop, alice, give\_guarantee, T_g) \wedge T_g < T_{cc} \\ &\wedge \mathbf{H}(alice, eShop, pay(device, cc), T_{cc}) \wedge T_{cc} < T_s \\ &\wedge \mathbf{H}(eShop, alice, deliver(device), T_s) \wedge T_s < 50\} \\ &(by\ \text{(shop2)})\end{aligned} \tag{11}$$

Summarising, $alice$ devised the following set of events, which should let her achieve her goal (have the desired device) while respecting both of $alice$'s and $eShop$'s policies.

$$\begin{aligned}\mathcal{C}_a =&\{\mathbf{H}(eShop, alice, give\_guarantee, T_g) \wedge T_g < T_p \\ &\wedge \mathbf{H}(alice, eShop, pay(device, cc), T_p) \wedge T_p < T_s \\ &\wedge \mathbf{H}(eShop, alice, deliver(device), T_s) \wedge T_s < 50)\}\end{aligned} \tag{12}$$

VIII. Rule Mark-Up

In WAV^e, the $\mathcal{IC}$s can be exchanged between web services, as well as advertised together with their WSDL. For this reason, a mark-up language is necessary for the rules in $\mathcal{IC}$s. RuleML [3] is the perfect mark-up language for exchanging rules on the web, so our choice has been easy. RuleML 0.9 contains mark-ups for expressing important concepts of the $\mathcal{S}$CIFF proof-procedure. In particular, $\mathcal{S}$CIFF is a rule engine able to distinguish and use both backward and forward rules. Backward rules are used to plan, reason upon events, perform proactive reasoning. Forward rules are used for reactive reasoning, to quickly perform actions in response to occurred events. Both are seamlessly integrated in $\mathcal{S}$CIFF. RuleML 0.9 contains a $direction$ attribute that can be attached to rules. Being based on abduction, $\mathcal{S}$CIFF 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.

$\mathcal{S}$CIFF was implemented in SICStus Prolog: SICStus contains an implementation of the PiLLoW library [19], which makes it easy to perform http requests, as well as implementing services on the web. Finally, SICStus contains an XML

parser, which allowed us to easily implement the RuleML parser. The RuleML parser is freely available on the $\mathcal{S}$CIFF web site [26].

IX. Discussion and related work

WAV^e is a framework intended for describing declaratively the behavioural interface of web services, and for testing operationally the possibility of fruitful interaction between them. WAV^e answers the question "does there exist a viable interaction, between two given web services, which achieves a given goal $\mathcal{G}$?" In case of success, WAV^e produces a set of expectations about events. WAV^e is particularly suitable for highly dynamic environments, in which interoperability is an unknown that has to be checked.

WAV^e uses and extends a technology initially developed for online compliance verification of agent interaction to protocols [6]. $\mathcal{S}$CIFF and the protocol specification language based on social integrity constraints were motivated and inspired by conspicuous work done in the context of agent interaction in open societies, notably work by Singh [27] and colleagues. The extension of such a work to the context of web services, centering around the concept of policies, as proposed in this work, seems to be very promising. The idea of policies for web services and policy-based reasoning is one that many other authors also adopt. We will cite work by Finin and colleagues [20], and by Bradshaw and colleagues [28], the first one with an emphasis on representation of actions, the latter on the deontic semantic aspects of web service interaction. We acknowledge the importance of action modelling and we point that the idea of expected behaviour of web services can have a deontic reading. In fact, previous work on $\mathcal{S}$CIFF has been devoted to investigating and clarifying the interesting links between deontic operators and expectation-based reasoning [8]. The distinguishing features of WAV^e, compare to most work of literature, are its logical underpinning and its sound and complete operational characterisation. It is in our agenda to carry out an extensive empiric evaluation of WAV^e based on interesting cases and scenarios such as those proposed in related work, and on the existing implementation of the $\mathcal{S}$CIFF framework.[3]

Another direction of current work relates to the actual use of the answers of WAV^e by web services after they manage a successful derivation. In principle, the sequence of events produced by WAV^e could be instantiated into a concrete sequence of messages, which will guarantee the achievement of $\mathcal{G}$, under ideal external conditions. But this is true only if the policies disclosed by both web services are a faithful representation of their actual behaviour. This may not be the case, as for example policies may depend on sensible data, and web services may be not allowed to disclose full information to the outside. In that case nothing warrants that the course of action produced by WAV^e will be satisfactory for either web service. We might then have to resort to further steps. For example both web services could "formally" agree that a certain course of events in an acceptable option, possibly after another mutual verification phase. This is subject of ongoing work. Finally, we are currently investigating the exchange of policies between web services, for which a suitable interaction protocol needs to be devised. We are thinking of specifying such a protocol for exchanging the policies in the same language WAV^e uses to specify policies.

In this work, we do not address the problem of reasoning about ontologies: rather, we focus on the reasoning process, given the policies of both the peers involved in the interaction. Many other approaches instead focus on the former issue: hence our proposal could be seen as a complementary functionality, that could further improve the discovery process. For instance, in [24] an ontology language to define web services is proposed. In [1], [2], besides proposing a general language for representing semantic web service specifications using logic, a discovery scenario is depicted and an architectural solution is proposed (we draw inspiration for our scenario from the Discovery Engine example). A notion of mediator is introduced to overcome differences between different ontologies, and then a reasoning process is performed over the user inputs and the hypothetical effects caused by the service execution. To some extent, our work can be related to [23], where the authors present a framework for automated web service discovery that uses the Web Service Modeling Ontology (WSMO) as the conceptual model for describing web services, requester goals and related aspects. This conceptual framework distinguishes two main stages. During the discovery stage, the requester states only the things that are desired, thus seeks for all the services that can potentially satisfy a request of such a kind. During the contracting stage, instead, the requester provides in input specific information for an already requested service. The purpose of this second stage is to verify that the input provided will lead to a desired state that satisfies the requester goal. In our work, we are concerned mainly with the contracting stage. While in [23] the authors use F-logic and transaction logic as the underlying formalisms, we rely on extended logic programming. In both the approaches, however, hypothetical reasoning is used for service contracting.

In [23], [1], [2], only the clients goal is considered, while in our framework the client can specify its policies (its intended behavioural interface); in this way, the client could be considered a web service as well. Therefore, our framerwork can be exploited without major changes to deal with the problem of interoperability between web services behavioural interfaces [4].

Foster et al. [16] propose a model-based approach to verify a given service composition can successfully execute a choreography, in particular with respect to the obligations imposed on the services by a choreography. The web service specifications and the choreography are translated to FSP algebra and checked by model checking techniques. The main difference with respect to our work is that Foster et al. check a web service composition against a choreography for conformance, while we check a set of web services for their capability to achieve a goal. Another notable difference is in

[3] See `http://lia.deis.unibo.it/research/sciff`.

the adopted formal approaches (abduction in our case, model checking in theirs).

The outcome of the reasoning process performed by WAV^e could be intended as a sort of "contract agreement" between the client and the discovered service, provided that each peer is tightly bounded to the policies/knowledge bases he/she has previously published. For example, the dynamical agreement about contracts (e-contracting) is addressed in [13], where Situated Courteous Logic is adopted for reasoning about rules that defines business provisions policies. The used formalism supports contradicting rules (by imposing a priorization and mutual exclusion between rules), different ontologies, and effectors as procedures with side effects. However, their work is more focussed on establishing the characteristics of a business deal, while our approach address the problem of evaluating the feasibility of an interaction. To this end, we perform hypothetical reasoning on the possible actions and consequences; moreover, we hypothesised also which condition must hold, in order to interoperate. Other works use rules to reason about established contracts: in [18], for example, Defeasible Deontic Logic of Violation is used to monitor the execution of a previously agreed contract. We have addressed such issue in a companion paper [8], where integrity constraints have been exploited and conciliated with the deontic concepts.

ACKNOWLEDGEMENTS

This work has been partially supported by the MIUR PRIN 2005 projects *Specification and verification of agent interaction protocols* 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] http://www.w3.org/Submission/2005/SUBM-SWSF-Applications-20050909/.

[3] Asaf Adi, Suzette Stoutenburg, and Said Tabet, editors. *Rules and Rule Markup Languages for the Semantic Web, First International Conference, RuleML 2005, Galway, Ireland, November 10-12, 2005, Proceedings*, volume 3791 of *Lecture Notes in Computer Science*. Springer Verlag, 2005.

[4] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Marco Montali. An abductive framework for a-priori verification of web services. In Michael Maher, editor, *Proceedings of the Eighth Symposium on Principles and Practice of Declarative Programming, July 10-12, 2006, Venice, Italy*, pages 39–50, New York, USA, July 2006. Association for Computing Machinery (ACM), Special Interest Group on Programming Languages (SIGPLAN), ACM Press.

[5] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, Marco Montali, and Paolo Torroni. Policy-based reasoning for smart web service interaction. In *Proceedings of the 1st International Workshop on Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS 2006)*, volume 196 of *CEUR Workshop Proceedings*, pages 87–102, Seattle, WA, USA, August 2006.

[6] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. The SOCS computational logic approach for the specification and verification of agent societies. In Corrado Priami and Paola Quaglia, editors, *Global Computing: IST/FET International Workshop, GC 2004 Rovereto, Italy, March 9-12, 2004 Revised Selected Papers*, volume 3267 of *Lecture Notes in Artificial Intelligence*, pages 324–339. Springer-Verlag, 2005.

[7] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Verifiable agent interaction in abductive logic programming: the $\mathcal{S}$CIFF proof-procedure. Technical Report DEIS-LIA-06-001, University of Bologna (Italy), March 2006. LIA Series no. 75.

[8] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, Giovanni Sartor, and Paolo Torroni. Mapping deontic operators to abductive expectations. *Computational and Mathematical Organization Theory*, 12(2–3):205 – 225, October 2006.

[9] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. The $\mathcal{S}$CIFF abductive proof-procedure. In *Proceedings of the 9th National Congress on Artificial Intelligence, AI*IA 2005*, volume 3673 of *Lecture Notes in Artificial Intelligence*, pages 135–147. Springer-Verlag, 2005.

[10] José Júlio Alferes, Carlos Viegas Damásio, and Luís Moniz Pereira. Semantic web logic programming tools. In François Bry, Nicola Henze, and Jan Maluszynski, editors, *PPSWR*, volume 2901 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2003.

[11] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business process execution language for web services version 1.1, May 2003. http://www.ibm.com/developerworks/library/ws-bpel/.

[12] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.

[13] S. Bhansali and N. Grosof. Extending the sweetdeal approach for e-procurement using sweetrules and ruleml. In Adi et al. [3].

[14] Franois Bry and Michael Eckert. Twelve theses on reactive rules for the web. In *Proceedings of the Workshop on Reactivity on the Web*, Munich, Germany, March 2006.

[15] M. Denecker and D. De Schreye. SLDNFA: an abductive procedure for abductive logic programs. *Journal of Logic Programming*, 34(2):111–167, 1998.

[16] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Model-based analysis of obligations in web service choreography. In *Proceedings of the International Conference on Internet and Web Applications and Services (ICIW 2006)*, Guadeloupe French Caribbean, 2006. IEEE Computer Society Press.

[17] T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 33(2):151–165, November 1997.

[18] G. Governatori and D. P. Hoang. A semantic web based architecture for e-contracts in defeasible logic. In Adi et al. [3].

[19] Daniel Cabeza Gras and Manuel V. Hermenegildo. Distributed WWW programming using (Ciao-)Prolog and the PiLLoW library. *Theory and Practice of Logic Progr.*, 1(3):251–282, 2001.

[20] Lalana Kagal, Timothy W. Finin, and Anupam Joshi. A policy based approach to security for the semantic web. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 402–418. Springer, 2003.

[21] A. C. Kakas, R. A. Kowalski, and Francesca Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.

[22] A. C. Kakas and Paolo Mancarella. On the relation between Truth Maintenance and Abduction. In T. Fukumura, editor, *Proceedings of the 1st Pacific Rim International Conference on Artificial Intelligence, PRICAI-90, Nagoya, Japan*, pages 438–443. Ohmsha Ltd., 1990.

[23] Michael Kifer, Ruben Lara, Axel Polleres, Chang Zhao, Uwe Keller, Holger Lausen, and Dieter Fensel. A logical framework for web service discovery. In *Semantic Web Services: Preparing to Meet the World of Business Applications, ISWC Worshop, Hiroshima, Japan*, November 2004.

[24] D. et al. Martin. http://www.daml.org/services/owl-s/1.0/.

[25] Working Group on Rule Interchange Format. Use cases and requirements. `http://www.w3.org/2005/rules/wg/ucr/draft-20060323.html`, March 2006.

[26] The *SCIFF* abductive proof procedure, 2005. `http://lia.deis.unibo.it/research/sciff/`.

[27] M. Singh. Agent communication language: rethinking the principles. *IEEE Computer*, pages 40–47, December 1998.

[28] Andrzej Uszok, Jeffrey M. Bradshaw, Renia Jeffers, Austin Tate, and Jeff Dalton. Applying kaos services to ensure policy compliance

for semantic web services workflow composition and enactment. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 425–440. Springer, 2004.