Università degli Studi di Bologna
DEIS

Policy-based reasoning for smart web service interaction

Marco Alberti Federico Chesani
Marco Gavanelli Evelina Lamma
Paola Mello Marco Montali Paolo Torroni

May 2006

DEIS Technical Report no. DEIS-LIA-06-002 *LIA Series no. 76*

Policy-based reasoning for smart web service interaction

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

[1] *Dipartimento di Ingegneria, Università di Ferrara*
Via Saragat, 1
44100 Ferrara, Italy
{marco.alberti, marco.gavanelli, evelina.lamma}@unife.it
[2] *DEIS, Università di Bologna*
V.le Risorgimento 2
40136 Bologna, Italy
{f chesani, m montali}@deis.unibo.it
{paola.mello, paolo.torroni}@unibo.it

May 2006

Abstract. We present a vision of smart, goal-oriented web services that reason about other services' policies and evaluate the possibility of future interactions. To achieve our vision, we propose a proof theoretic approach. 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.

Keywords: *Abductive Logic Programming, Proof-procedures, Agent Interaction Protocols, Declarative Semantics, Formal Properties, $\mathcal{S}$CIFF, SOCS (SOcieties of ComputeeS), SCIFF, IFF Proof-procedure, WAV^e*

Contents

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, 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 [5]: composing new applications through existing web services.

On the upside, the recent popularity of these new technologies developed into a growing presence of web services, made available through the Internet, and we can foresee a steady increase of such services also for the near future. On the downside, the lifetime of software developed with the classical methodologies of composition of existing services at design-time gets shorter and shorter. It quickly becomes a suboptimal choice, blind to the exploitation of new opportunities. In highly competitive markets, this can be a severe drawback.

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 becomes 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 some service, 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. In the first phase, a web service collects information about other web services, and tries to understand by reasoning which ones can lead to a fruitful interaction. This activity is carried out off-line, beforehand. In the second phase, the web service uses the information it gathered to interact with other web services. 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 one of them can verify whether (some of) its own goals can be reached by interacting with them.

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 UE 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*: a sort of reactive rules used to generate and reason about expectations about possible evolutions of a given interaction setting. 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, standing for Web-service Abductive Verification.

The paper is structured as follows. We start by showing the abstract architecture of WAV^e. In Sec. 3 we present a scenario of on-line shopping, used throughout the paper to demonstrate the proposed approach. In Sect. 4, we briefly introduce the language used in the framework, and in Sect. 5, we show how the scenario can be modeled in WAV^e in terms of *ICs*. Sect. 6 presents the declarative and operational semantics of WAV^e, and Sect. 7 proposes the application of WAV^e to the verification problem in the reference scenario. A brief discussion follows.

2 The Architecture of WAV^e

A general reference architecture is depicted in Fig. 1, where the 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

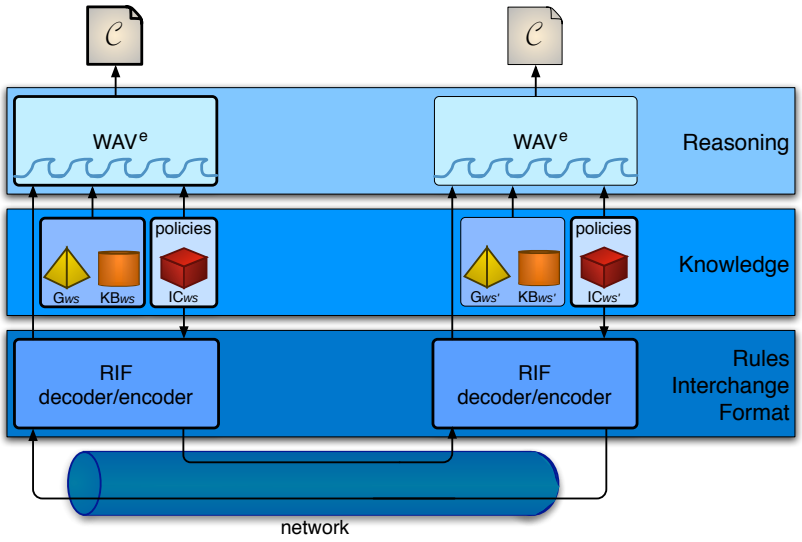

Figure 1. *The architecture of WAV^e*

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 Sect. 4. 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 [1]. 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 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.

3 The *alice* & *eShop* Scenario

This scenario is inspired to the one described by the Working Group on Rule Interchange Format [12]. A similar scenario is also in [6]. 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 that 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 any item from *eShop*, *alice* checks whether her policies and *eShop*'s policies are compatible, i.e., if they allow a successful transaction regarding the sales. 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.

To put it more formally, we can imagine *eShop*'s and *alice*'s policies to be defined as follows:

(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.

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

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. if *eShop* accepts, *alice* and *eShop* engage in a transaction that (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.

4 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 **H** will stand for events that a web service expects to be producing itself; atoms denoted by functor **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 [10] 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 control 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 [11], Fung and Kowalski [8], Denecker and De Schreye [7], etc.).

Definition 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 the 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 **E** expectations about the behaviour of the web services, as defined above. $\mathcal{KB}_{ws}$'s syntax is summarised in Eq. (1).

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

$\mathcal{E}_{ws}$ is the set of *abducible predicates*, which includes **E** expectations, **H** events,

and predicates not defined in $\mathcal{KB}_{ws}$.

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

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 that defined in [2]. In particular, unlike $\mathcal{S}$CIFF, WAV^e treats **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.

5 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.

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

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

[2] The alternative in the head could also be expressed by means of 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.

represent actions that should be performed by *alice*: from *eShop*'s viewpoint, they can only be expected.

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

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).

$$\begin{aligned}&\mathbf{E}(eShop, alice, give\_guarantee, T_g)\\ \rightarrow&\mathbf{H}(eShop, alice, give\_guarantee, T_g).\end{aligned} \tag{shop3}$$

IC shop3 says that if *alice* expects to receive a guarantee, then *eShop* will send it.

The following two rules represent *alice*'s policies.

$$\begin{aligned}&\mathbf{E}(alice, eShop, pay(Item, cc), T_p)\\ \rightarrow&\mathbf{E}(eShop, alice, give\_guarantee, T_g) \wedge T_g < T_p.\end{aligned} \tag{alice1}$$

IC alice1 says that, if *eShop* expects *alice* to pay for an Item by credit card, then *alice* expects *eShop* to provide a guarantee in advance.

$$\begin{aligned}&\mathbf{E}(alice, eShop, pay(Item, cc), T_p)\\ &\wedge \mathbf{H}(eShop, alice, give\_guarantee, T_g) \wedge T_g < T_p\\ \rightarrow&\mathbf{H}(alice, eShop, pay(Item, cc), T_p).\end{aligned} \tag{alice2}$$

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 $\mathcal{KB}_{alice}$

$$\begin{aligned}&have(alice, Item, T) \leftarrow\\ &\mathbf{E}(eShop, alice, deliver(Item), T_d) \wedge T_d \leq T.\end{aligned} \tag{alice3}$$

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.

6 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 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. We are currently studying an XML-like extension of RuleML [1] to represent $\mathcal{IC}$s.

Another possible approach is to obtain other 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, in general, to obtain a service from another web service. $\mathcal{G}$ will often be an expectation (of obtaining a service, of accessing a resource, of gathering information), but in general it can be any goal, defined as a conjunction of expectations, CLP constraints, and any other literals, in the syntax of $\mathcal{IC}_{ws}$ *Head*' *Disjunct*s given in 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. by disclosing (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 **H** events nor **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 **HAP** is a conjunction of **H** atoms, **EXP** is a conjunction of **E** atoms, and Δ a conjunction of abducible atoms.

We ground the notion of entailment on a model theoretic semantics defined for Abductive Disjunctive Logic Programs in [13].

Different semantics have been proposed for logic programs with disjunctions. Among them, answer set semantics [9] adopts an exclusive interpretation of disjunction, whereas possible model semantics [13] adopts an inclusive one (and recovers the former by additional constraints imposing mutual exclusion among the literals in the disjunctive head of a clause). In the possible model semantics, the disjunctive program gives raise to several (non-disjunctive) split programs, obtained in practice by separating the disjuncts in the head of rules. A possible model for a disjunctive logic program P is then an answer set of a split program of P.

The inclusive interpretation of disjunctions better fits with our case, since more than one disjunct in the head of an integrity constraint can be true at the same time, as shown by the following example.[3]

Example 1. *Let us consider the program:*

$$\begin{array}{rcl} \mathbf{E}(p) \vee \mathbf{H}(p) & \leftarrow & \mathit{true}. \\ \mathbf{H}(p) & \leftarrow & \mathbf{E}(p). \\ \mathit{goal} & \leftarrow & \mathbf{E}(p). \end{array}$$

We would like to have an explanation for goal, where $\mathbf{E}(p)$ *is assumed, and* $\mathbf{H}(p)$ *is also true because of clause* $\mathbf{H}(p) \leftarrow \mathbf{E}(p)$*. This is, instead, avoided by following an answer set approach.*

Furthermore, in [13] possible model semantics was also applied to provide a model theoretic semantics for Abductive Extended Disjunctive Logic Programs (AEDP), which is our case. In particular, semantics is given to AEDP in terms of possible belief sets. Given an AEDP $\Pi = \langle P, \mathcal{A} \rangle$, a possible belief set S of Π is a possible model of the disjunctive program P extended with a set of abducible literals.

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

In our approach, we rely upon possible belief set semantics, but we adopt a new notion for minimality with respect to abducible literals:

[3] For the sake of simplicity, in this example and in the following one, we specify a single argument for expectations and events.

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

More intuitively, minimality with respect to hypotheses made is checked by considering the same split program, and by checking whether with a smaller set of abducibles the same consequences can be made true.

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

Example 2.

$$\begin{array}{rcl} \mathbf{E}(p) \ \vee \ \mathbf{H}(p) & \leftarrow & \mathbf{E}(p). \\ \mathit{goal} & \leftarrow & \mathbf{E}(p). \end{array}$$

This program splits into three different split programs (by choosing the first, the second or both the disjuncts when producing the split rules). The minimal explanations for goal, for each split program respectively, correspond to the possible belief sets: $\{\mathit{goal}, \mathbf{E}(p)\}$, $\{\mathit{goal}, \mathbf{E}(p), \mathbf{H}(p)\}$, *and, again,* $\{\mathit{goal}, \mathbf{E}(p), \mathbf{H}(p)\}$.

Nonetheless, only the first explanation is minimal under set inclusion, but we cannot rely upon such definition for minimality since we would discard explanations which are, instead, correct.

Among all $\mathcal{A}$-minimal possible belief sets, and all answers for a given goal, we are interested in those containing *possible interactions*, as indicated by the following definition. **Definition 5 (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.

However, a possible interaction might not be fruitful; 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 of the two web services.

We select, among the set of 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

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

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

Definition 6 (possible interaction achieving $\mathcal{G}$) Given two web services, ws and ws', and a goal $\mathcal{G}$, we define a *possible interaction achieving $\mathcal{G}$* a possible interaction satisfying Eq. 5.

Intuitively, the "→" implication in Eq. 5 is there to avoid situations in which a web service waits forever for an event that the other web service will never produce. The "←" implication is there to avoid that one web service sends unexpected messages, which in the best case may not be understood.

6.1 Operational Semantics

The operational semantics is a modification of the $\mathcal{S}$CIFF proof-procedure [4].

One feature of $\mathcal{S}$CIFF is on-line checking of compliance of agent interaction to protocols. $\mathcal{S}$CIFF processes events drawing from **HAP** and generates expectations (a special type of abducibles). As soon as new **H** events are processed, the relevant matching expectations are labelled as *fulfilled*. At the end of the derivation, when the $\mathcal{S}$CIFF proof-procedure reaches quiescence, if some expectation ha remained unfulfilled, a failure node is generated, and other alternative branches will be explored in backtracking, if there exist any.

As opposed to $\mathcal{S}$CIFF, WAV^e abduces **H** events as well as expectations. The history is not taken as input, but all possible interactions are hypothesised. In WAV^e, we apply the same labelling technique adopted in $\mathcal{S}$CIFF for *fulfilment*, to accommodate the "**E** → **H**" part of Eq. 5. As for the "**H** → **E**" implication of Eq. 5, symmetrically to *fulfilment*, WAV^e labels each **H** events with an *expected* flag as soon as an expectation matching it is abduced. At quiescence, **H** with *expected* status = false will cause failure.

6.2 Results

WAV^e is a conservative modification of the $\mathcal{S}$CIFF proof-procedure, which is sound and complete under reasonable assumptions [3]. Although we have no formal results for WAV^e yet, we believe that the results proven for $\mathcal{S}$CIFF can be easily exported to the declarative and operational semantics of WAV^e. In particular, the only "important" addition of WAV^e to $\mathcal{S}$CIFF, beside syntactic aspects, is the "→" implication of Eq. 5. It is subject for current work to verify the consequences of such an implication in terms of soundness and completeness results.

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

7 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:

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

Then, by unfolding of clause alice3,

$$\mathbf{EXP}_a^0 = \{ \quad \mathbf{E}(eShop, alice, deliver(device), T_s) \wedge T_s < 50 \ \} \qquad (by\ (\text{alice3}))$$

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

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

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 **H** is abduced). This choice triggers (alice1):

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

Then (shop3) fires, and abduces the happening of *give_guarantee* event. We then have:

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

$$\mathbf{HAP}_a^3 = \{ \quad \mathbf{H}(eShop, alice, give\_guarantee, T_g) \wedge T_g < T_{cc} \ \} \qquad (by\ (\text{shop3}))$$

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

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

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

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

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

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

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.

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

8 Discussion

WAV^e is a framework intended to be used 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 inter-operability is an unknown that has to be checked.

One first question: after a successful check, how can a web service use the answers of WAV^e? 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.

Another issue that deserves further investigation is 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.

Acknowledgements

This work has been partially supported by the MIUR PRIN 2005 projects n. 2005-011293, *Specification and verification of agent interaction protocols* and n.2005-015491, *Constraints and preferences: a unified formalism for computer systems analysis and real problem solution.*

References

[1] A. Adi, S. Stoutenburg, and S. 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.

[2] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. The SOCS computational logic approach for the specification and verification of agent societies. In C. Priami and P. 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.

[3] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable agent interaction in abductive logic programming: the $\mathcal{S}$CIFF proof-procedure. Technical Report DEIS-LIA-06-001, University of Bologna (Italy), Mar. 2006. LIA Series no. 75.

[4] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. 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.

[5] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for web services version 1.1, May 2003. http://www.ibm.com/developerworks/library/ws-bpel/.

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

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

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

[9] M. Gelfond and V. Lifschitz. Classical negation in logic programming and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

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

[11] A. C. Kakas and P. 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.

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

[13] C. Sakama and K. Inoue. Abductive logic programming and disjunctive logic programming: their relationship and transferability. *Journal of Logic Programming*, 44(1-3):75–100, 2000.