Web Service Contracting: Specification and Reasoning with SCIFF

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

[1] ENDIF, University of Ferrara
Via Saragat 1, 44100 Ferrara, Italy
`Name.Surname@unife.it`
[2] DEIS, University of Bologna
V.le Risorgimento 2, 40136 Bologna, Italy
`Name.Surname@unibo.it`

Abstract. The semantic web vision will facilitate automation of many
tasks, including the location and dynamic reconfiguration of web services.
In this article, we are concerned with a specific stage of web service loca-
tion, called, by some authors, *contracting*. We address contracting both
at the operational level and at the semantic level. We present a frame-
work encompassing communication and reasoning, in which web services
exchange and evaluate goals and policies. Policies represent behavioural
interfaces. The reasoning procedure at the core of the framework is based
on the abductive logic programming SCIFF proof-procedure. We de-
scribe the framework, show by examples how to formalise policies in the
declarative language of SCIFF, and give the framework a model-theoretic
and a sound proof-theoretic semantics.

1 Introduction

The Service Oriented Computing (SOC) paradigm, and its practical implemen-
tation based on Web Services, are rapidly emerging as standard architectures
for distributed application development. Different service providers, heteroge-
nous in terms of hardware and software settings, can easily inter-operate by
means of communication standards and well-known network protocols. In such
a scenario, the use of off-the-shelf solutions and dynamic reconfiguration be-
comes attractive, both at design level, as well as at execution (run-time) level.
However, dynamic reconfiguration of services is possible only if supported by a
suitable technology. The Semantic Web vision, based on the idea that adding
machine-understandable semantic information to Web resources will facilitate
automation of many tasks [18,20], including the location of Web Services, is a
promising way to address this issue.

Drawing from [18], we consider a process of searching the right service to
match a request as consisting of two stages. During the first one, called *discov-
ery*, the requester states only the things that are desired, thus, using an ontology
or other KR formalisms, it seeks for all the services that can potentially satisfy

a request of such a kind. During the second stage, called *contracting*, 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's goal.

Many relevant papers are written about web service discovery; some of them use Logic Programming (LP) techniques. They mostly focus on the discovery stage. In this article, we are concerned with the contracting stage, which we address both at the operational level, and at the semantic level. We present a framework, called SCIFF Reasoning Engine (SRE) encompassing reasoning and communication in a networked architecture inspired to the model of discovery engines. We discuss the problem of communicating policies between web services, and of determining whether the policies of a provider and a requester are compatible with each other. We use a mixture of Abductive Logic Programming (ALP, [17]), and Constraint Logic Programming (CLP, [16]). ALP is used to construct sets of input and output messages, along with assumed data, while CLP is used to maintain a partial temporal order among them. We chose to adopt a hypothetical reasoning framework such as ALP because reasoning is made before execution. A component such as SRE which reasons on possible developments of the interaction among web services has to come up with hypotheses about which messages are to be expected in the future. In fact, a similar approach, also based on hypothetical reasoning, though not on LP, has been followed by others, notably [18].

In this work we assume that a previous discovery stage has already produced multiple candidate services. The intended user of SRE will typically be a web service, although for the sake of presentation we will name the users *alice* and *eShop*. The user query is given in terms of goals and policies. Policies describe the observable behaviour of users, i.e., their *behavioural interface*, in terms of relationships among externally observable events regarding the service. We formalise web services' policies in a declarative language derived from the $\mathcal{S}$CIFF language, born in the context of the EU SOCS project [1]. $\mathcal{S}$CIFF was conceived to specify and verify social-level agent interaction. In this work, a simplified version of the $\mathcal{S}$CIFF language is adopted for defining policies, by means of *social integrity constraints*: a sort of reactive rules used to generate and reason about possible evolutions of a given interaction instance. Distinguishing features of SRE are its declarative and operational approaches combined together into a working framework. Users specify their goals as well as their own policies (related to the goals) by means of rules; in their turn, service providers publish their service descriptions, together with their policies about the usage of the services, again by means of rules. The use of ALP reconciles forward and backward reasoning in a unified reasoning engine: two aspects that frequently, in the context of web services, are treated separately from each other. Moreover, while ALP and CLP have been used to address planning problems before, in this work we want to show how a mixture of known, but little-used techniques can be used to solve a real-world problem with a real implementation.
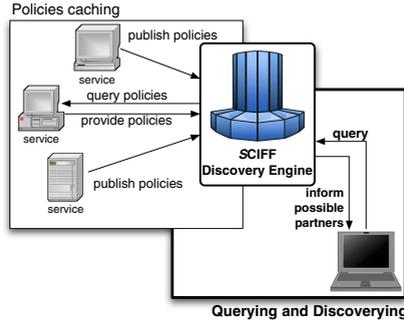
Fig. 1. The architecture of SRE

In the next section, we show the SRE architecture and introduce informally a walk through scenario. In Sect. 3 we explain the notation used to write policies in SRE and in Sect. 4 we present its underlying logic. Sect. 5 develops the scenario in more detail and shows the reasoning in SRE by example, and Sect. 6 concludes by discussing related approaches and future work.

2   Architecture

The SRE architecture, shown in Figure 1, is a client-server architecture augmented with a "smart" discovery engine (which incorporates the SRE itself). We assume that SRE has information available about web services, either gathered in a previous discovery phase from the Internet (in the style of web spiders), or because explicitly published to it by web services. So we can safely assume that the data collected has already been filtered and, if providers refer to different ontologies, equivalences between concepts have already been established.

At the logical level, the retrieved information consists of triplets in the form $\langle s, ws, (\mathcal{KB}_{ws}, \mathcal{P}_{ws}) \rangle$, where s identifies a service, ws is the name of a web service that provides s, and $(\mathcal{KB}_{ws}, \mathcal{P}_{ws})$ are the knowledge base and policies that ws associates to s. In particular, for a given provider ws providing s, a set of policies (rules) describes ws's behaviour with respect to s, and a knowledge base, in the form of a logic program, contains information that ws wants to disclose to potential customers, together with its policies. A sample policy could state that the service delivers goods only to certain countries, or zones. The list of such zones could be made available through the knowledge base.

SRE reasons based on a client's query (also called *goal*, in the LP sense) which it matches to a service. Such a query will contain the name of the service that the client (c) needs, a (possibly empty) set of policies $\mathcal{P}_c$ and a (possibly empty) knowledge base $\mathcal{KB}_c$. The goal is an expression consisting of a conjunction of elements, which can represent, for example, events and constraints, like partial orders among events. The output of SRE is a number of triplets $\langle ws, \mathcal{E}, \Delta \rangle$, each one containing the name of a web application which provides the service, plus

some additional information: $\mathcal{E}$, which encodes a possible future interaction, i.e., a partially ordered sequence of events, occurring between ws and c and regarding s, and a set Δ containing a number of additional validity conditions for $\mathcal{E}$. For example, ws could be the name of a service that provides a *device*, $\mathcal{E}$ could be "first ws shows evidence of membership to $\mathcal{B}etter\ \mathcal{B}usiness\ \mathcal{B}ureau$ ($\mathcal{BBB}$), then c pays by credit card", and Δ could be "delivery in Europe". These concepts are better instantiated in the following scenario.

2.1 The *alice* and *eShop* Scenario

The scenario we use in this paper is inspired from [11,2]. *eShop* is a web service that sells devices, while *alice* is another web service, which wants to get a *device*. *alice* and *eShop* describe their behaviour concerning sales/payment/... of items through policies (rules), which they publish using some Rules Interchange Format. These two actors find each other via SRE: in particular, *alice* submits a query to the discovery engine, by specifying her policies and the service she is looking for (e.g., getting *device*). Once suitable services (e.g., *eShop*) have been found, SRE, by checking the satisfiability of *alice*'s goal and the compatibility of the rules describing *alice*'s and *eShop*'s behaviour, provides back to *alice* the list of web services that could satisfy her specific need. SRE also defines the conditions that must be fulfilled by each web service, in order to reach the goal.

Let *eShop*'s policies regarding *device* be as follows:

(shop1) if a customer wants to get an item, then, (i) if the customer can be accepted, *eShop* will request him/her to pay using an acceptable method, otherwise (ii) *eShop* will inform the customer of a failure;

(shop2) if an acceptable customer paid the item, using an acceptable method, then *eShop* will deliver the item;

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

eShop publishes a knowledge base $\mathcal{KB}_{eShop}$, which specifies that a customer is *accepted* if it is resident in some zone; it also specifies the accepted payment methods. SRE retrieves information about *eShop* in the triplet: $\langle sell(device),$ $eShop, (\mathcal{KB}_{eShop}, \mathcal{P}_{eShop})\rangle$, indicating that *eShop* offers service *sell(device)*, with a set $\mathcal{P}_{eShop}$ of policies defined as $\mathcal{P}_{eShop} \equiv \{(\mathsf{shop1}), (\mathsf{shop2}), (\mathsf{shop3})\}$ and a knowledge base $\mathcal{KB}_{eShop}$. We consider three different scenaria for *alice*:

Scenario 1. *alice*'s goal is to obtain *device*. Her policies are as follows:

(alice1) if a shop requires that *alice* pays by credit card, *alice* expects that the shop provides a certificate to guarantee that it is a $\mathcal{BBB}$ member;

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

(alice3) if a shop requires *alice* to pay with any other method than credit card, then *alice* will pay without any further request;

Besides, *alice* is based in Europe. However, for privacy reason, *alice* does not make this information public. $\mathcal{KB}_{alice}$ is an an empty knowledge base.

Scenario 2. Policies are the same as above. However, *alice* will not agree to pay cash, as she specifies in her query to SRE. Moreover, $\mathcal{KB}_{alice}$ is not empty, but instead it contains information about her place of residence and age;

Scenario 3. *alice* has no policies to express in relation to the query she submits to SRE. We can imagine here that *alice* is a human user, and she queries SRE, using a suitable interface, simply because she wishes to know what her options are regarding the purchase of *device*. Later, *alice* may evaluate SRE's answer and possibly re-submit a refined query.

3   Notation

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

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

Web service specifications in SRE are relations among expected events, expressed by an abductive logic program. This is in general a triplet $\langle \mathcal{KB}, \mathcal{A}, \mathcal{IC} \rangle$, where $\mathcal{KB}$ is a logic program, $\mathcal{A}$ (sometimes left implicit) is a set of literals named *abducibles*, and $\mathcal{IC}$ is a set of *integrity constraints*. Intuitively, in ALP the role of $\mathcal{KB}$ is to define predicates, the role of $\mathcal{A}$ is to fill-in the parts of $\mathcal{KB}$ which are unknown, and the role of $\mathcal{IC}$ is to control the ways elements of $\mathcal{A}$ are hypothesised, or "abduced." Reasoning in ALP is usually goal-directed. It starts from a "goal" $\mathcal{G}$, i.e., an expression which we want to obtain as a logical consequence of the abductive logic program, and it amounts to finding a set of abduced hypotheses Δ built from atoms in $\mathcal{A}$ such that $\mathcal{KB} \cup \Delta \models \mathcal{G}$ and $\mathcal{KB} \cup \Delta \models \mathcal{IC}$. Symbol $\models$ represents logical entailment, which can be associated with one among several semantics. In literature one finds different readings of abduction in LP. Δ can be considered as an "answer" to a query or goal $\mathcal{G}$. In other contexts, one can interpret $\mathcal{G}$ as an observation and Δ as its explanation. This is for example the reading of an abductive anwer in abductive reasoning-based diagnosis. In the

domain of web services, we will use ALP as a reasoning paradigm that combines backward, goal-oriented reasoning with forward, reactive reasoning [19]: two aspects that frequently, in the context of web services, are treated separately from each other.

Definition 1 (Web Service Behavioural Interface Specification). *Given a web service ws, its* web service behavioural interface specification $\mathcal{S}_{ws}$ *is an abductive logic program, represented by the triplet* $\mathcal{S}_{ws} \equiv \langle \mathcal{KB}_{ws}, \mathcal{A}, \mathcal{IC}_{ws} \rangle$, *where* $\mathcal{KB}_{ws}$ *is ws's Knowledge Base,* $\mathcal{A}$ *is the set of* abducible literals, *and* $\mathcal{IC}_{ws}$ *is ws's set of Integrity Constraints (ICs).*

$\mathcal{KB}_{ws}$, which corresponds to $\mathcal{KB}_{ws}$ of Sect. 2, 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/EN** expectations about the behaviour of the web services. $\mathcal{A}$ is the set of *abducible literals*. It includes all possible **E/EN** expectations, **H** events, and predicates left undefined by $\mathcal{KB}_{ws}$. It is the set of all possible unknowns. Note that $\mathcal{E}_{ws}$ and Δ of Sect. 2 are subsets of $\mathcal{A}$. In the following sometimes we leave $\mathcal{A}$ implicit, as we did in Sect. 2. $\mathcal{IC}_{ws}$, which corresponds to $\mathcal{P}_{ws}$ of Sect. 2, contains *ws*'s policies. In particular, each IC in $\mathcal{IC}_{ws}$ is a rule in the form *Body → Head*. Intuitively, the *Body* of an IC is a conjunction of events, literals and CLP constraints; the *Head* is either a disjunction of conjunctions of events, literals and CLP constraints, or *false*. The operational behaviour of ICs is similar to that of forward rules: whenever the body becomes true, the head is also made true. The syntax of $\mathcal{KB}_{ws}$ and $\mathcal{IC}_{ws}$ is given in Equations (1) and (2), respectively, where *Constr* indicates a CLP constraint [16].

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

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

Let us see how we can implement the walk through scenario in SRE. Note that, following the LP notation, variables (in *italics*) start with upper-case letters. $T_r, T_a, \ldots$ indicate the (expected) time of events.

The first IC in $\mathcal{IC}_{eShop}$, corresponding to (shop1), is the following:

$$
\begin{aligned}
\mathbf{H}&(Customer, eShop, request(Item), T_r) \\
\rightarrow &accepted\_customer(Customer) \\
&\wedge accepted\_payment(How) \\
&\wedge \mathbf{H}(eShop, Customer, ask(pay(Item, How)), T_a) \\
&\wedge \mathbf{E}(Customer, eShop, pay(Item, How), T_p) \\
&\wedge T_p > T_a \wedge T_a > T_r \\
\vee &rejected\_customer(Customer) \\
&\wedge \mathbf{H}(eShop, Customer, inform(fail), T_i) \wedge T_i > T_r.
\end{aligned}
\tag{shop1}
$$

All accepted payment modalities are listed in *eShop*'s knowledge base, $\mathcal{KB}_{eShop}$, shown in (kb) below. In our example, *Customer* may pay either by credit card or cash. The concepts of "accepted" and "rejected" customer are defined in the KB_{eShop} too: a *Customer* is accepted if the *Zone* she resides in is a valid destination for *eShop*; *Customer* is *rejected* otherwise. Both payment modalities and accepted destinations are listed as facts. In this example, *eShop* can only send items to Europe. The next element of *eShop*'s policies (shop2) states that if an accepted *Customer* pays for an *Item* using one of the supported payment modalities, then *eShop* will deliver the *Item* to *Customer*:

$$
\begin{aligned}
\mathbf{H}&(Customer, eShop, pay(Item, How), T_p) \\
&\wedge accepted\_customer(Customer) \\
&\wedge accepted\_payment(How) \\
\rightarrow &\mathbf{H}(eShop, Customer, deliver(Item), T_d) \\
&\wedge T_d > T_p.
\end{aligned}
\tag{shop2}
$$

Finally, (shop3) states that if a *Customer* asks it to provide a guarantee (i.e., a certificate about its membership to $\mathcal{BBB}$), *eShop* will send such a guarantee:

$$
\begin{aligned}
\mathbf{H}&(Customer, eShop, request\_guar(\mathcal{BBB}), T_{rg}) \\
\rightarrow &\mathbf{H}(eShop, Customer, give\_guar(\mathcal{BBB}), T_g) \\
&\wedge T_g > T_{rg}.
\end{aligned}
\tag{shop3}
$$

$$
\begin{aligned}
accepted\_customer(Customer) \leftarrow &resident\_in(Customer, Zone) \\
&\wedge accepted\_dest(Zone). \\
rejected\_customer(Customer) \leftarrow &resident\_in(Customer, Zone) \\
&\wedge not(accepted\_dest(Zone)). \\
accepted&\_payment(cc). \\
accepted&\_payment(cash). \\
accepted&\_dest(europe).
\end{aligned}
\tag{kb}
$$

When a (generic) *Shop* asks *alice* to pay an *Item* with credit card, then *alice* will request the *Shop* to provide a guarantee, and she will expect to receive it:

$$\mathbf{H}(Shop, alice, ask(pay(Item, cc)), T_a)$$
$$\rightarrow \mathbf{H}(alice, Shop, req\_guar(\mathcal{BBB}), T_{rg})$$
$$\wedge \mathbf{E}(Shop, alice, give\_guar(\mathcal{BBB}), T_g) \tag{alice1}$$
$$\wedge T_g > T_{rg} \wedge T_{rg} > T_a.$$

If *Shop* provides a guarantee, *alice* will pay for the requested *Item*:

$$\mathbf{H}(Shop, alice, ask(pay(Item, cc)), T_a)$$
$$\wedge \mathbf{H}(Shop, alice, give\_guar(\mathcal{BBB}), T_g)$$
$$\rightarrow \mathbf{H}(alice, Shop, pay(Item, cc), T_p) \tag{alice2}$$
$$\wedge T_p > T_A \wedge T_p > T_g.$$

When the *Shop* asks to use a payment modality other than credit card, *alice* satisfies *eShop*'s request:

$$\mathbf{H}(Shop, alice, ask(pay(Item, How)), T_a)$$
$$\wedge How \neq cc$$
$$\rightarrow \mathbf{H}(alice, Shop, pay(Item, How), T_p) \wedge T_p > T_A. \tag{alice3}$$

4 Declarative Semantics and Reasoning

In SRE, a client c specifies a goal $\mathcal{G}$, related to a requested service. $\mathcal{G}$ will often be an expectation, but in general it can be any goal, defined as a conjunction of expectations, CLP constraints, and any other literals. c also publishes a (possibly empty) knowledge base $\mathcal{KB}_c$, and a (possibly empty) set of policies $\mathcal{IC}_c$. The idea is to obtain, through abductive reasoning made by SRE, a set of expectations $\mathcal{E}$ and validity conditions Δ about a possible course of events that, together with $\mathcal{KB}_c$ and $\mathcal{KB}_{ws}$, satisfies $\mathcal{IC}_c \cup \mathcal{IC}_{ws}$ and $\mathcal{G}$. Note that we do not assume that all of ws's knowledge base is available to SRE, as it need not be entirely a part of ws's public specifications. $\mathcal{KB}_{ws}$ can even be the empty set. However, in general, ICs can involve predicates defined in the KB: such as "delivery in Europe." If the behavioural interface provided by ws involves predicates that have not been made public through $\mathcal{KB}_{ws}$, SRE makes assumptions about such unknown predicates, and considers unknowns that are neither $\mathbf{H}$ nor $\mathbf{E/EN}$ expectations as literals that can be abduced. These are kept then in the set Δ, of a returned triplet $\langle ws, \mathcal{E}, \Delta \rangle$ (see Sect. 2), and can be regarded as conditions which must be met to insure the validity of $\mathcal{E}$ as a possible set of expectations achieving a goal.

4.1 Declarative Semantics

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

$$\mathcal{KB}_c \cup \mathcal{KB}_{ws} \cup \mathcal{E} \cup \Delta \models \mathcal{G} \tag{3}$$
$$\mathcal{KB}_c \cup \mathcal{KB}_{ws} \cup \mathcal{E} \cup \Delta \models \mathcal{IC}_c \cup \mathcal{IC}_{ws}. \tag{4}$$

where $\mathcal{E}$ is a conjunction of **H** and **E**, **EN** atoms, Δ is a conjunction of abducible literals, and the notion of entailment is grounded on the *possible models* semantics defined for abductive disjunctive logic programs [23]. In the possible models semantics, a disjunctive program generates several (non-disjunctive) *split programs*, obtained by separating the disjuncts in the head of rules. Given a disjunctive logic program P, a *split program* is defined as a (ground) logic program obtained from P by replacing every (ground) rule

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

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

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

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

   Note that in [23] the possible models semantics was also applied to provide a model theoretic semantics for Abductive Extended Disjunctive Logic Programs (AEDP), which is our case. Semantics is given to AEDP in terms of *possible belief sets*. Given an AEDP $\Pi = \langle P, \mathcal{A} \rangle$, where P is a disjunctive logic program and $\mathcal{A}$ is the set of abducible literals, a possible belief set S of Π is a possible model of the disjunctive program $P \cup E$, where P is extended with a set E of abducible literals ($E \subseteq \mathcal{A}$).

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

We rely upon possible belief set semantics, but we adopt a new notion for minimality with respect to abducible literals. In [23], a possible belief set S is $\mathcal{A}$-minimal if there is no possible belief set T such that $T \cap \mathcal{A} \subset S \cap \mathcal{A}$. We restate, then, the notion of $\mathcal{A}$-minimality as follows:

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

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

Definition 4 ($\mathcal{A}$-minimal answer to a goal). *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 $\mathcal{G}$.*

Definition 5 (Possible Interaction about $\mathcal{G}$). *A possible interaction about a goal $\mathcal{G}$ between a client c and a web service ws is an $\mathcal{A}$-minimal set $\mathcal{E} \cup \Delta$ such that Equations 3 and 4 hold.*

Among possible interactions, we identify those which are *coherent*:[1]

Definition 6 (Coherent Possible Interaction about $\mathcal{G}$). *A possible inter-action $\mathcal{E} \cup \Delta$ about a goal $\mathcal{G}$ is* coherent *iff:*

$$\mathcal{E} \models \mathbf{E}(X, Y, Action, T), \mathbf{EN}(X, Y, Action, T) \rightarrow false \tag{5}$$

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

SRE selects among coherent possible interactions only those where the course of events expected by c about ws's messages is *fulfilled* by ws's messages, and vice-versa, i.e., the course of events expected by ws about c's messages is *fulfilled* by c's messages:

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

$$\mathcal{E} \models \mathbf{E}(X, Y, Action, T) \rightarrow \mathbf{H}(X, Y, Action, T) \tag{6}$$

$$\mathcal{E} \models \mathbf{EN}(X, Y, Action, T), \mathbf{H}(X, Y, Action, T) \rightarrow false \tag{7}$$

In practice, Definition 7 requires that any positive expectation raised by c or ws on the behaviour of the other party is fulfilled by an event hypothetically performed by the other party (Equation 6), and that any negative expectation raised by c or ws on the behaviour of the other party does not match any event hypothetically performed by the other party (Equation 7).

4.2   Operational Semantics

The operational semantics of SRE is a modification of the $\mathcal{S}$CIFF proof-procedure [8], that combines forward, reactive reasoning with backward, goal-oriented reasoning, and was originally developed to check compliance of the agent behaviour to interaction protocols in multi-agent systems. Like the IFF proof procedure [13], which inspired it, $\mathcal{S}$CIFF is a rewriting system, defined in terms of *transitions* which turn a state of the computation into another. Since some of the transitions open choice points, a computation can be represented as a tree, whose root node is the initial state and whose leaves can be either the special node *fail*, or a termination node (i.e., a node to which no transition is applicable), that is considered as a *success* node (and, in the original $\mathcal{S}$CIFF setting, represents a response of compliance of the agent behaviour to the interaction protocols)

$\mathcal{S}$CIFF is sound and complete, under reasonable assumptions [8]; it has been implemented in SICStus Prolog and Constraint Handling Rules [12] and integrated in the SOCS-SI software component, in order to interface it to several

---

[1] This notion is introduced because of **EN** expectations in the SRE framework, and therefore the necessity of stating explicitly the incompatibility between **E** and **EN**.

multi-agent systems [7], and with web services via a RuleML encoding of ICs. The $\mathcal{S}$CIFF version that acts as the core reasoning engine in SRE is designed to reason, off-line, about the web service behaviour: a successful leaf node represents an interaction which achieves the desired goal while respecting the specified policies. SRE is a conservative modification of the SCIFF proof-procedure, in which the happened events are abducibles. The proofs of soundness and completeness can be trivially extended to such a case.

5   The *alice* and *eShop* Scenario Revisited

We provide here a sketched demonstration of the operational behaviour of the SRE engine, by showing how the answers to *alice*'s query are generated. Let us suppose that *alice* sends a query to SRE containing policies (alice1), (alice2) and (alice3), an empty knowledge base and the following goal $\mathcal{G}$:

$$\mathcal{G} \equiv \mathbf{H}(alice, Shop, request(device), T_r)$$
$$\wedge \mathbf{E}(Shop, alice, deliver(device), T_d) \wedge T_d > T_r. \tag{goal1}$$

which states that *alice* will send a request to some *Shop* in order to obtain *device* and she expects that *Shop* will deliver it. SRE starts from *alice*'s goal:

$$\mathcal{E}_0 = \{\mathbf{H}(alice, eShop, request(device), T_r),$$
$$\mathbf{E}(eShop, alice, deliver(device), T_d), T_d > T_r \}$$
$$\Delta_0 = \emptyset$$

According to (shop1), *eShop* may react to this expectation in different ways, depending on whether *alice* is an accepted customer or not. SRE tries initially to resolve predicate $accepted\_customer(alice)$. By unfolding it, SRE finds atom $resident\_in(alice, Zone)$, which is not known to SRE and, therefore, is abduced. Afterwards, based on $\mathcal{KB}_{eShop}$, the *eShop* public knowledge base, SRE grounds $Zone$ to $europe$: the only destination accepted by *eShop*. As a consequence, hypothesising that *alice* is resident in $europe$, *eShop* would ask *alice* to pay with one of the accepted modalities, and it would expect to receive the payment in response. Moreover, *eShop* specifies in $\mathcal{KB}_{eShop}$ that credit card is an accepted payment modality.

$$\mathcal{E}_1 = \{\mathbf{H}(alice, eShop, request(device), T_r),$$
$$\mathbf{H}(eShop, alice, ask(pay(device, cc)), T_a),$$
$$\mathbf{E}(alice, eShop, pay(device, cc), T_p),$$
$$\mathbf{E}(eShop, alice, deliver(device), T_d),$$
$$T_p > T_a, T_a > T_r, T_d > T_r \}$$
$$\Delta_1 = \{resident\_in(alice, europe)\}$$

alice has specified that, in order to perform credit card payments, she requests a guarantee from the shop (alice2); *eShop* volunteers to provide such a document,

by (shop3), and *alice*'s expectation about the guarantee is then satisfied (SRE hypothesises that the document is indeed sent):

$$\mathcal{E}_2 = \{\mathbf{H}(alice, eShop, request(device), T_r),$$
$$\mathbf{H}(eShop, alice, ask(pay(device, cc)), T_a),$$
$$\mathbf{H}(alice, eShop, req\_guar(\mathcal{BBB}), T_{rg}),$$
$$\mathbf{H}(eShop, alice, give\_guar(\mathcal{BBB}), T_g),$$
$$\mathbf{E}(alice, eShop, pay(device, cc), T_p),$$
$$\mathbf{E}(eShop, alice, deliver(device), T_d),$$
$$T_g > T_{rg}, T_{rg} > T_a, T_p > T_a, T_a > T_r, T_d > T_r \}$$
$$\Delta_2 = \{resident\_in(alice, europe)\}$$

Upon receipt of the guarantee, *alice* would proceed with the payment (alice2), and *eShop* would deliver the *device* (shop3). Therefore, the following, (possibly) fruitful, interaction is found by SRE:

$$\mathcal{E}_F = \{\mathbf{H}(alice, eShop, request(device), T_r),$$
$$\mathbf{H}(eShop, alice, ask(pay(device, cc)), T_a),$$
$$\mathbf{H}(alice, eShop, req\_guar(\mathcal{BBB}), T_{rg}),$$
$$\mathbf{H}(eShop, alice, give\_guar(\mathcal{BBB}), T_g),$$
$$\mathbf{H}(alice, eShop, pay(device, cc), T_p),$$
$$\mathbf{H}(eShop, alice, deliver(device), T_d),$$
$$T_d > T_p, T_p > T_g, T_g > T_{rg}, T_{rg} > T_a, T_a > T_r \}$$
$$\Delta_F = \{resident\_in(alice, europe)\}$$

SRE provides in output also a simpler possible interaction, where instead of selecting "credit card" as payment method, "cash" is now preferred. Policy (alice3) tells us that, in such a case, *alice* would proceed straightforward with the payment, and SRE is indeed able to propose a second fruitful interaction as answer to *alice*'s initial query.

In order to compute these two possibly fruitful interactions, $resident\_in(alice, europe)$ has been abduced. This means that if such interactions are really possible or not, it depends on whether *alice* resides in *europe*, and in fact it may well turn out that such interaction is not possible at all. SRE looks also for other solutions where this hypothesis is not assumed, but all other interactions do not satisfy *alice*'s goal, and hence they are discarded.

5.1 Refined Query

In the second scenario, *alice* submits a different goal, and the $\mathcal{KB}$ below:

$$\mathcal{G} \equiv \mathbf{H}(alice, Shop, request(device), T_r)$$
$$\wedge \mathbf{E}(Shop, alice, deliver(device), T_d) \wedge T_d > T_r \qquad \text{(goal2)}$$
$$\wedge \mathbf{EN}(alice, Shop, pay(device, cash), T_p).$$

$$resident\_in(alice, europe). \quad age(alice, 24). \qquad \text{(kb2)}$$

This time, *alice* explicitly prohibits to pay cash (this is expressed using the **EN** notation). Thanks to the piece of knowledge (kb2) that *alice* provides through her

$\mathcal{KB}$, SRE knows that *alice* does indeed resides in the EU, hence this information does not need to be abduced anymore, but it is simply verified. SRE finds a solution which is similar to the one above (Scenario 1). However, since this time the set Δ is empty, this interaction will surely lead to success, provided that both *alice* and *eShop* behave coherently with respect to their own policies.

5.2   Unconstrained Query

As we have pointed out, *alice* is able to query SRE without specifying any policy. In this case, *alice* only wishes to obtain a list of services that are able to accommodate her goal. In such a situation, *alice* only sends the following general policy:

$$E(alice, Shop, DoSomething, T)$$
$$\rightarrow H(alice, Shop, DoSomething, T) \tag{r1}$$

which specifies that *alice* will perform every action that she is expected to do. If *alice* queries SRE by using (r1) and (goal1), the response of SRE will be:

$$\mathcal{E}_F = \{\mathbf{H}(alice, eShop, request(device), T_r),$$
$$\mathbf{H}(eShop, alice, ask(pay(device, How)), T_a),$$
$$\mathbf{H}(alice, eShop, pay(device, How), T_p),$$
$$\mathbf{H}(eShop, alice, deliver(device), T_d),$$
$$T_d > T_p, T_p > T_a, T_a > T_r, How :: [cc, cash] \}$$
$$\Delta_F = \{resident\_in(alice, europe)\}$$

6   Discussion

We described a reasoning engine, SRE, which considers the policies of two web services and a goal of one of them. SRE tries to match such policies and find possible ways the two web services could interact, and eventually achieve the goal. The output of SRE is a sequence of events, which could be messages to be exchanged between the web services and lead to a state in which the goal is achieved. This can be regarded as a possible plan of action. SRE is based on a mixture of ALP and CLP. ALP is used to construct sets of input and output messages, along with assumed data, while CLP is used to maintain a partial temporal order among the plans. In this work we did not address the issue of efficiency of the reasoning process of SRE, but we are aware that this may be a drawback, as it is the case with many expressive logics proposed for the Semantic Web. We intend to evaluate SRE, both as it concerns its complexity and its efficiency, through an empirical analysis based on case studies.

Another aspect we did not look into in detail is the problem of reasoning about equivalences of concepts or ontologies, as much related work instead does. Also our notions of *action*, such as it could be the delivery of goods, are pretty much left at the abstract level. Our proposal could be regarded as a functionality complementary to many proposals, which could further improve the discovery

process. To cite some, [3,22] propose ontology languages to define web services. In [4], besides proposing a general language for representing semantic web service specification 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.

Our work makes explicit reference to [18], in which the authors present a framework for automated web service discovery which uses the Web Service Modeling Ontology (WSMO) as the conceptual model for describing web services, requester goals and related aspects. Whereas [18] tackles both (mainly) discovery and contracting stage, in our work we are only concerned with the contracting stage. In [18] 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. Compare to work by Kifer et al. [18,4], in which only the client's goal is considered, in our framework the client can specify its policies. In this way, the client could be considered a web service as well. Therefore, we hope to be able to smoothly extend SRE to dealing with the problem of inter-operability. A proposal in this direction is presented in [6].

The outcome of the reasoning process performed by SRE, which we called a possible plan, could in fact be regarded as a sort of "contract agreement" between the client and the discovered service, provided that each party is tightly bounded to its previously published policies/knowledge bases. For example, the dynamic agreement about contracts (e-contracting) is addressed in SweetDeal [15,10], where Situated Courteous Logic (SCL) is adopted for reasoning about rules that define business provisions policies. The formalism used supports contradicting rules (by imposing a prioritisation and mutual exclusion between rules), different ontologies, and effectors as procedures with side effects. However, SweetDeal is more focussed on establishing the characteristics of a business deal, while our aim is to 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 hypothesise which condition must hold, in order to inter-operate. This technique in literature is also known as "constructive" abduction.

Other authors propose to rules to reason about established contracts: in [14], for example, Defeasible Deontic Logic of Violation is used to monitor the execution of a previously agreed contract. We have addressed this issue in a companion paper [9], where integrity constraints have been exploited and conciliated with the deontic concepts. Among other work in the area of policy specifications and matching we find PeerTrust [21,5]. Similarly to our work and to SCL, PeerTrust builds upon an LP foundation to represent policy rules and iterative trust declaratively. In PeerTrust, trust is established gradually by disclosing credentials and requests for credentials by using a process of trust negation. An important difference is in the language used in PeerTrust for specifying policies, which can be considered as orthogonal to the one described in this paper. While PeerTrust

represents policies and credentials as guarded distributed logic programs, and the trust negotiation process consists of evaluating an initial LP query over a physically distributed logic program, in this work we use ALP, integrity constraints and CLP constraints for expressing policies, perform a local proof and we use abductive reasoning to formulate hypotheses about unknown external behaviour. Moreover, while in our current approach reasoning is done in a single step using $\mathcal{S}$CIFF, an iterative version could be introduced in order to support trust negotiation.

Acknowledgments

References

1. http://lia.deis.unibo.it/research/socs/.
2. http://www.w3.org/TR/rif-ucr/.
3. http://www.daml.org/services/owl-s/.
4. http://www.w3.org/Submission/SWSF-SWSL/.
5. http://www.l3s.de/peertrust/.
6. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and M. Montali. An abductive framework for a-priori verification of web services. In *Proc. PPDP*, pp. 39–50. ACM Press, 2006.
7. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Compliance verification of agent interaction: a logic-based tool. *Applied Artificial Intelligence*, **20**(2-4):133–157, 2006.
8. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable agent interaction in Abductive Logic Programming: the SCIFF framework. *ACM Transactions on Computational Logic*, **8**, 2007.
9. M. Alberti, M. Gavanelli, E. Lamma, P. Mello, G. Sartor, and P. Torroni. Mapping deontic operators to abductive expectations. *Computational and Mathematical Organization Theory*, **12**(2-3):205–225, 2006.
10. S. Bhansali and N. Grosof. Extending the sweetdeal approach for e-procurement using sweetrules and RuleML. In *Proc. RuleML, LNAI* **3791**:113–129, 2005.
11. F. Bry and M. Eckert. Twelve theses on reactive rules for the web. In *Proc. Workshop on Reactivity on the Web*, Munich, Germany, March 2006.
12. T. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, **37**(1-3):95–138, 1998.
13. T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, **33**(2):151–165, 1997.
14. G. Governatori and D. P. Hoang. A semantic web based architecture for e-contracts in defeasible logic. In *Proc. RuleML, LNAI* **3791**:145–159, 2005.
15. B. N. Grosof and T. C. Poon. SweetDeal: representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *Proc. 12th WWW*, pp. 340–349. ACM Press, 2003.

16. J. Jaffar and M. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, **19-20**:503–582, 1994.
17. A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, **2**(6):719–770, 1993.
18. M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, and D. Fensel. A logical framework for web service discovery. In *Semantic Web Services: Preparing to Meet the World of Business Applications. CEUR Workshop Proc.* **119**, 2004.
19. R. A. Kowalski and F. Sadri. From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, **25**(3/4):391–419, 1999.
20. S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, **16**(2):46–53, 2001.
21. W. Nejdl, D. Olmedilla, and M. Winslett. PeerTrust: Automated trust negotiation for peers on the semantic web. In *Proc. Secure Data Management (SMD 2004)*, *LNAI* **3178**:118–132. Springer-Verlag, 2004.
22. D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web service modeling ontology. *Applied Ontology*, **1**(1):77 – 106, 2005.
23. C. Sakama and K. Inoue. Abductive logic programming and disjunctive logic programming: their relationship and transferability. *JLP*, **44**(1-3):75–100, 2000.