An Abductive Framework for A-Priori Verification of Web Services

Marco Alberti Marco Gavanelli
Evelina Lamma

ENDIF - Engeneering Dept. in Ferrara
University of Ferrara
Via Saragat, 1 - 44100, Ferrara
Italy
{marco.alberti,marco.gavanelli,evelina.lamma}@unife.it

Federico Chesani Paola Mello
Marco Montali

DEIS - Dip. di Elettronica, Informatica e Sistemistica
University of Bologna
viale Risorgimento, 2 - 40136, Bologna
Italy
{fchesani,pmello,mmontali}@deis.unibo.it

Abstract

Although stemming from very different research areas, Multi-Agent Systems (MAS) and Service Oriented Computing (SOC) share common topics, problems and settings. One of the common problems is the need to formally verify the conformance of individuals (Agents or Web Services) to common rules and specifications (resp. Protocols/Choreographies), in order to provide a coherent behaviour and to reach the goals of the user.

In previous publications, we developed a framework, $\mathcal{S}$CIFF, for the automatic verification of compliance of agents to protocols. The framework includes a language based on abductive logic programming and on constraint logic programming for formally defining the social rules; suitable proof-procedures to check on-the-fly and a-priori the compliance of agents to protocols have been defined.

Building on our experience in the MAS area, in this paper we make a first step towards the formal verification of web services conformance to choreographies. We adapt the $\mathcal{S}$CIFF framework for the new settings, and propose a heir of $\mathcal{S}$CIFF, the framework A^{l}LoWS (Abductive Logic Web-service Specification). A^{l}LoWS comes with a language for defining formally a choreography and a web service specification. As its ancestor, A^{l}LoWS has a declarative and an operational semantics. We show examples of how A^{l}LoWS deals correctly with interaction patterns previously identified. Moreover, thanks to its constraint-based semantics, A^{l}LoWS deals seamlessly with other cases involving constraints and deadlines.

Categories and Subject Descriptors F.4.1 [*MATHEMATICAL LOGIC AND FORMAL LANGUAGES*]: Mathematical Logic - Computational Logic; F.3.1 [*LOGICS AND MEANINGS OF PROGRAMS*]: Specifying and Verifying and Reasoning about Programs; D.2.12 [*SOFTWARE ENGINEERING*]: Interoperability; I.2.3 [*ARTIFICIAL INTELLIGENCE*]: Deduction and Theorem Proving - Logic Programming

General Terms Algorithms, Languages, Verification.

PPDP'06 July 10–12, 2006, Venice, Italy.

Keywords Abduction, Web Services, Choreographies, Formal Verification, Constraints

1. Introduction

The mating between high availability of network resources and the growing of software applications is breeding in recent years new technologies and software tools. Network ubiquity, joined together with the software engineering requirement to combine existing tools and divide the complexity of applications, is spawning new programming paradigms.

One of the most successful is Service Oriented Computing, one of the children of Object Oriented Computing. Web service technology is an important instance of Service Oriented Computing aiming at facilitating the integration of new applications, avoiding difficulties due to different platforms, languages, etc. In this context the way to build complex services from simpler ones is called *composition* and it is a very interesting and promising research area. Service composition promises to considerably reduce development time and costs by taking components off-the-shelf and joining them in a working application. Although very appealing, it leaves open questions, such as correctness of the composition, or ensuring interoperability of the web services. Choreographies propose an answer to such questions. The behaviour of the various web services is defined through a language (e.g., WS-CDL [23]), explaining the information flows amongst the components. However, the formal proofs of conformance of a web service to a choreography are not yet fully given.

Another technology descending from networks and artificial intelligence is the Multi Agent Systems (MAS). In the MAS area there exists a wide literature about checking the conformance of an agent to social rules (or protocols), both at run-time and at design-time. Baldoni et al. [8], amongst others, pointed out the similarities of requirements in the two areas of web service composition and multi agent systems. Both are devoted to define a collaboration between a collection of peers that share common goals. Both Choreographies and societies should capture interactions and dependencies between interactions (time constraints, deadlines, control-flow dependencies, etc.). Both describe the external behaviour of members avoiding the internal details of the implementation of the peers.

Stemming from previous experience in multi-agent systems, we follow the path of Baldoni et al. and propose to apply agent techniques to web service composition. Within the SOCS european project [27], we proposed a formal language to define multi agent protocols [5, 6]. We gave an abductive semantics to the devised language, and developed an abductive proof procedure, called $\mathcal{S}$CIFF

[7], to check on-line the compliance of agents to protocols. More recently, we applied a variant of $\mathcal{S}$CIFF, called g-$\mathcal{S}$CIFF [2], to the problem of proving properties of a protocol itself (such as security properties). In this work, we apply these technologies to web service composition, and propose a framework, called A^{l}LoWS (Abductive Logic Web-service Specification), that exploits the variants of $\mathcal{S}$CIFF for checking the interoperability of web services.

This work extends a previous proposal that was limited to deterministic choreographies, i.e., the web service does not have choices, but should always reply with predefined messages [11].

2. The A^{l}LoWS framework

We propose a framework, called A^{l}LoWS (Abductive Logic Web-service Specification), to verify the conformance of web services to choreographies. A^{l}LoWS is based on computational logic, and, in particular, it has an abductive semantics.

Abduction is the inference rule that lets an intelligent system raise *hypotheses*, and reason about them, in order to find explanations for some evidence. Its typical application is the diagnosis, or, in general all the applications in which reasoning from effects to causes is necessary. It has often been used for *planning* (for example, with the abductive event calculus [15]): a goal state is considered as the final effect, and the sequence of actions (hypotheses) that would cause such a state make up a plan for obtaining the goal.

In this paper, we adopt abductive reasoning to make hypotheses on the sequence of events that might occur in an interaction between web services. We are able to generate such interaction through abductive proof-procedures, and to test conformance of web services to choreographies.

Formally, an Abductive Logic Program [22] is a triple $\mathcal{P} \equiv \langle KB, \mathcal{E}, \mathcal{IC} \rangle$ where KB is a logic program, $\mathcal{E}$ is a distinguished set of predicates, called *abducibles*, that have no definition in KB, and $\mathcal{IC}$ is a set of logical formulas, called *Integrity Constraints*, that involve both abducibles and defined predicates, and that must always hold true. The aim is to find a set $\Delta \subseteq \mathcal{E}$ such that it explains a goal G

$$KB \cup \Delta \models G$$

and such that all the integrity constraints are satisfied

$$KB \cup \Delta \models \mathcal{IC}.$$

A^{l}LoWS builds upon the tools and technologies developed in the SOCS project in the Multi Agent Systems area. In A^{l}LoWS, the behaviour of the web services is represented by means of *events*. Since we focus on the interactions between web services, events always represent exchanged messages. We adopt the same syntax used in [8]: a message is described by the term $m_x(Sender, Receiver, Content)$, where m_x is the type of message, and the arguments retain their intuitive meaning. We sometimes simplify the notation, and omit some of the parameters when the meaning is clear from the context.

In A^{l}LoWS we have two types of events. Happened events are represented as $\mathbf{H}(Message, Time)$, where $Message$ has the syntax previously defined, and $Time$ is an integer, representing the time point in which the event happened. As we will see in the following, the $\mathbf{H}$ predicate can be abduced, when making hypotheses on the possible interactions. In other phases, happened events are considered as given a priori, thus considered as a defined predicate.

The second type of events are *expectations*. From both web services and choreography's viewpoint, given a past history of happened events, more events are expected to happen, in a conformant evolution. We represent expectations with the predicate $\mathbf{E}_X(Message, Time)$ expressing the fact that the corresponding event is expected to happen, in order to fulfil the coherent evolution, from the viewpoint of X (where X might be either the choreography or a web service). An expectation is called *fulfilled* if there exists a matching $\mathbf{H}$ event. For example, the expectation $\mathbf{E}(p(X), T)$ can be fulfilled by the event $\mathbf{H}(p(a), 1)$.

2.1 Specification of a Choreography

A choreography describes, from a global viewpoint, what are the patterns of communication, or interactions, allowed in a system that adopts such choreography [9]. The choreography specification defines the messages that are allowed: it is not possible to exchange other messages except the ones explicitly specified. The choreography also enlists the participants, the roles the participants can play, and other knowledge about the web service interaction.

We specify a choreography by means of an abductive logic program [22]. A choreography specification $\mathcal{P}_{chor}$ is defined by the triple:

$$\mathcal{P}_{chor} \equiv \langle KB_{chor}, \mathcal{E}_{chor}, \mathcal{IC}_{chor} \rangle$$

where:

- KB_{chor} is the *Knowledge Base*,
- $\mathcal{E}_{chor}$ is the set of *abducible predicates*, and
- $\mathcal{IC}_{chor}$ is the set of *Choreography Integrity Constraints*.

The *Knowledge Base* (KB_{chor}) specifies declaratively pieces of knowledge of the choreography, such as roles descriptions, list of participants, etc. KB_{chor} is expressed in the form of clauses (a logic program); the clauses may contain in their body expectations about the behaviour of participants, defined literals, and constraints, while their heads are atoms. The syntax is reported in Grammar (1), where $Atom$ and $Term$ have the usual meaning in Logic Programming [25] and $Constraint$ is interpreted as in Constraint Logic Programming [20].

$$\begin{array}{rcl} SOKB & ::= & [Clause]^\star \\ Clause & ::= & Atom \leftarrow Cond \\ Cond & ::= & ExtLiteral\ [\ \wedge ExtLiteral\ ]^\star \\ ExtLiteral & ::= & Literal | Expectation | Constraint \\ Expectation & ::= & \mathbf{E}_{chor}(Term\ [, T]) \\ Literal & ::= & Atom \mid \neg Atom \mid true \end{array} \quad (1)$$

The *abducible predicates* are those that can be hypothesized (abduced) in our framework, namely happened events (denoted by the functor $\mathbf{H}$) and expectations (denoted by the functor $\mathbf{E}_{chor}$).

Choreography Integrity Constraints $\mathcal{IC}_{chor}$ are forward rules, of the form *Body* $\rightarrow$ *Head*, whose *Body* can contain literals and (happened and expected) events, and whose *Head* can contain (disjunctions of) conjunctions of expectations. In Grammar (2) we report the formal definition.

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

The syntax of $\mathcal{IC}_{chor}$ is a simplified version of that defined for the SOCS Integrity Constraints [4]. In particular in A^{l}LoWS we do not need negative expectations and explicit negation.

In Fig. 1 a multi-party interaction is shown, expressed by the set of Integrity Constraints in Specification 2.1: the depicted scenario is about a User that wants to buy a flight ticket from a Flight Service, and pay by sending a payment order to a Bank. In this particular

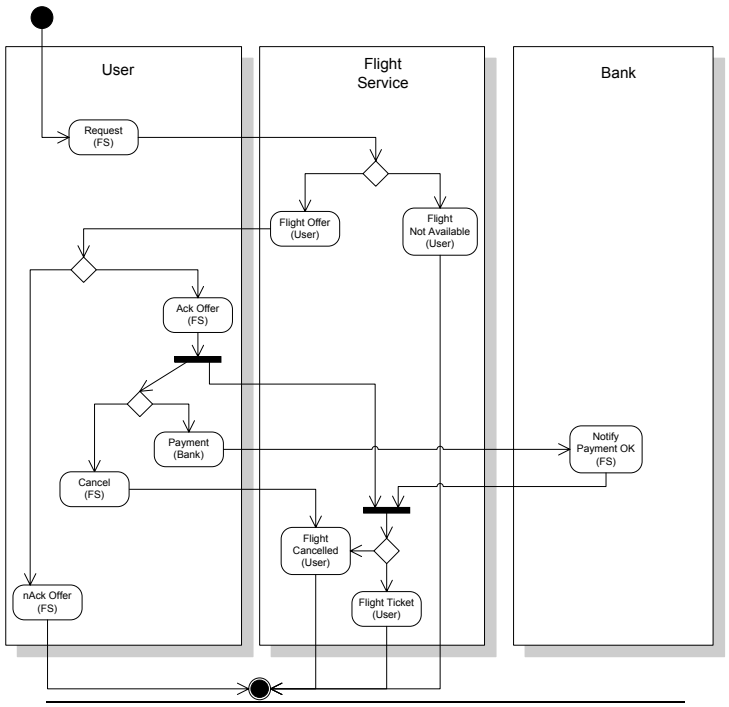

Figure 1. Graphical representation of a simple choreography

case, there was no need for a knowledge base of the choreography, so KB_{chor} is empty.

CLP constraints [20, 21] can be used to impose relations or restrictions on any of the variables that occur in an expectation, like imposing conditions on the role of the participants, or on the time instants the events are expected to happen. For example, time conditions might define orderings between the messages, or enforce deadlines.

A choreography can be *goal directed*, i.e. a specific goal $\mathcal{G}_{chor}$ can be specified: for example, a choreography used in an electronic auction system could have the goal of selling all the goods in the store. The syntax of the goal is the same as the condition of a clause ($Cond$ in Grammar 1). If no particular goal is required to be achieved, $\mathcal{G}_{chor}$ is bound to *true*.

2.2 Representing Web services

In an analogous way as we define the specification of a choreography, we describe the interface behaviour of a web service by means of an Abductive Logic Program. In particular, we restrict our analysis to the communicative aspects of the interface behaviour of a web service. A Web Service Interface Behaviour Specification $\mathcal{P}_{ws}$ is an Abductive Logic Program [22], represented with the triple

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

where:

- KB_{ws} is the *Knowledge Base* of the Web Service,
- $\mathcal{E}_{ws}$ is the set of abducible predicates, and
- $\mathcal{IC}_{ws}$ is the set of *Integrity Constraints*.

The *Knowledge Base* (KB_{ws}) specifies the knowledge of a Web Service. In KB_{ws}, clauses may contain in their body literals defined in KB_{ws}, expectations about the behaviour of the web service ws, or messages that ws expects to receive from other participants. It has the same syntax as the choreography's knowledge base, except for the expectations, that are indicated with the functor $\mathbf{E}_{ws}$ instead of $\mathbf{E}_{chor}$.

Specification 2.1 The specification of the choreography shown in Fig. 1.

$$\begin{aligned}&\mathbf{H}(request(User, FS, Flight), T_r)\\ \rightarrow&\mathbf{E}_{chor}(\textit{offer}(FS, User, Flight, Price), T_o)\\ \vee&\mathbf{E}_{chor}(\textit{notAvailable}(FS, User, Flight), T_{na})\end{aligned} \tag{3}$$

$$\begin{aligned}&\mathbf{H}(\textit{offer}(FS, User, Flight, Price), T_o)\\ \rightarrow&\mathbf{E}_{chor}(\textit{ackOffer}(User, FS, Flight, Price), T_a)\\ \vee&\mathbf{E}_{chor}(\textit{nAckOffer}(User, FS, Flight, Price), T_a)\end{aligned} \tag{4}$$

$$\begin{aligned}&\mathbf{H}(\textit{ackOffer}(User, FS, Flight, Price), T_a)\\ \rightarrow&\mathbf{E}_{chor}(\textit{payment}(User, Bank, Price, FS), T_f)\\ \vee&\mathbf{E}_{chor}(\textit{cancel}(User, FS, Flight), T_f)\end{aligned} \tag{5}$$

$$\begin{aligned}&\mathbf{H}(\textit{ackOffer}(User, FS, Flight, Price), T_a)\\ \wedge&\mathbf{H}(\textit{notifyPayment}(Bank, FS, Price), T_p)\\ \rightarrow&\mathbf{E}_{chor}(\textit{flightTicket}(FS, User, Flight), T_f)\\ \vee&\mathbf{E}_{chor}(\textit{flightCancelled}(FS, User, Flight), T_f)\end{aligned} \tag{6}$$

$$\begin{aligned}&\mathbf{H}(\textit{cancel}(User, FS, Flight), T_a)\\ \rightarrow&\mathbf{E}_{chor}(\textit{flightCancelled}(FS, User, Flight), T_f)\end{aligned} \tag{7}$$

$$\begin{aligned}&\mathbf{H}(\textit{payment}(User, Bank, Price, Creditor), T_p)\\ \rightarrow&\mathbf{E}_{chor}(\textit{notifyPayment}(Bank, Creditor, Price), T_n)\end{aligned} \tag{8}$$

$\mathcal{E}_{ws}$ is the set of abducible predicates. Similarly to the choreography specification, this set consists of both expectations (denoted by $\mathbf{E}_{ws}$) and happened events ($\mathbf{H}$). In the choreography specification the expectations are used for representing the global viewpoint of how things should go, hence all the expectations have the same meaning. In the web service specification instead we are expressing how the web service "perceives" the interaction: the viewpoint is local, and the expectations assume a slightly different meaning depending on who is expected to do what. More in detail:

- Expectations about messages where ws is the sender are intended as the possible messages that ws can indeed utter. Intuitively, expectations of the form $\mathbf{E}_{ws}(m_x(ws, Any, Content))$ represent the "active" behaviour of ws, i.e. the actions that it could perform. Hence they represent the "outgoing" communicative behaviour of ws. The conformance test should ensure that every possible message that ws could utter is indeed envisaged by the choreography.
- Expectations about messages where other participants are the senders and ws is the receiver can be intended instead as the messages that ws is able to understand. They are of the form $\mathbf{E}_{ws}(m_x(Any, ws, Content))$, with $Any \neq ws$.

Integrity Constraints $\mathcal{IC}_{ws}$ are forward rules, of kind *body* $\rightarrow$ *head*: from the syntactic viewpoint, they are identical to the $\mathcal{IC}_{chor}$ (except for the fact that expectations are from the web service's viewpoint: $\mathbf{E}_{ws}$ instead of $\mathbf{E}_{chor}$). While in the choreography specification we use them to specify the desired behaviour of the participants, $\mathcal{IC}_{ws}$ are used instead to describe the communication aspects of the interface behaviour of a web service ws.

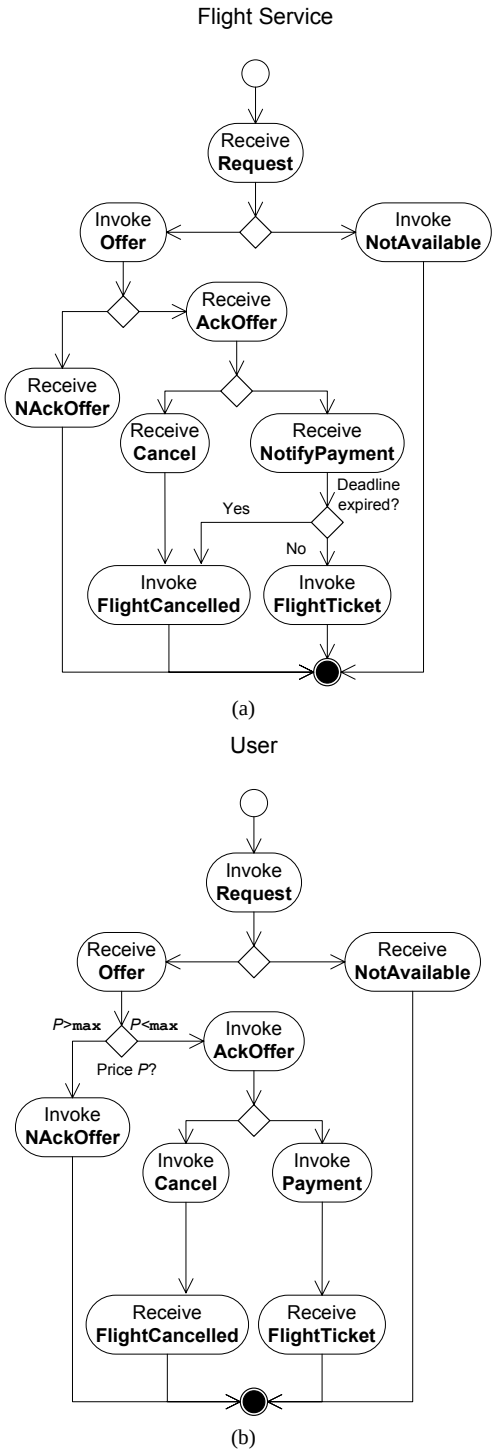

Figure 2. Example of behavioural interfaces

In Fig. 2(a) the communicative part of the interface behaviour of a web service is represented. The corresponding translation in terms of $\mathcal{IC}_{ws}$ is given in Spec. 2.2; in this case the knowledge base KB_{ws} is empty.

Specification 2.2 The interface behaviour specification of the web service shown in Fig. 2(a).

$$\begin{aligned}&\mathbf{H}(request(User, fs, Flight), T_r)\\ \rightarrow&\mathbf{E}_{fs}(\textit{offer}(fs, User, Flight, Price), T_o)\\ \vee&\mathbf{E}_{fs}(notAvailable(fs, User, Flight), T_{na})\end{aligned} \tag{9}$$

$$\begin{aligned}&\mathbf{H}(\textit{offer}(fs, User, Flight, Price), T_o)\\ \rightarrow&\mathbf{E}_{fs}(\textit{ackOffer}(User, fs, Flight, Price), T_a)\\ \vee&\mathbf{E}_{fs}(\textit{nAckOffer}(User, fs, Flight, Price), T_a)\end{aligned} \tag{10}$$

$$\begin{aligned}&\mathbf{H}(\textit{ackOffer}(User, fs, Flight, Price), T_a)\\ \wedge&\mathbf{H}(\textit{notifyPayment}(Bank, fs, Price), T_p)\\ \rightarrow&\mathbf{E}_{fs}(\textit{flightCancelled}(fs, User, Flight), T_c)\\ &\wedge T_p > T_a + \delta \wedge T_c > T_p\\ \vee&\mathbf{E}_{fs}(\textit{flightTicket}(fs, User, Flight), T_t)\\ &\wedge T_t > T_p\end{aligned} \tag{11}$$

$$\begin{aligned}&\mathbf{H}(\textit{ackOffer}(User, fs, Flight, Price), T_a)\\ \rightarrow&\mathbf{E}_{fs}(\textit{notifyPayment}(Bank, fs, Price), T_p)\\ \vee&\mathbf{E}_{fs}(\textit{cancel}(User, fs, Flight), T_c)\end{aligned} \tag{12}$$

$$\begin{aligned}&\mathbf{H}(\textit{cancel}(User, fs, Flight), T_a)\\ \rightarrow&\mathbf{E}_{fs}(\textit{flightCancelled}(fs, User, Flight), T_f)\end{aligned} \tag{13}$$

As for the choreographies, also web service specifications can be *goal directed*, by specifying a goal $\mathcal{G}_{ws}$, with the same syntax ($Cond$ in Grammar 1), in which the expectations are $\mathbf{E}_{ws}$ instead of $\mathbf{E}_{chor}$.

3. Conformance: declarative semantics

Intuitively, conformance is the characteristics of a web service to comply to a choreography, provided that the other peers will behave according to the choreography. From the declarative semantics viewpoint, the test of conformance requires to assume further hypotheses about events ws expects to utter, and events that the choreography expects other peers to utter. Both can be mapped into constraints, provided that we consider the predicate $\mathbf{H}$ as abducible. We use the web service's interface behaviour $\mathcal{P}_{ws}$ to foresee the messages the web service will send in every possible situation, provided that the other peers behave as specified by the choreography. Formally, all the messages the web service ws expects to send will be executed, i.e.:

$$\mathbf{E}_{ws}(m_x(ws, R, C), T) \rightarrow \mathbf{H}(m_x(ws, R, C), T) \tag{14}$$

Symmetrically for the messages exchanged by other peers as prescribed by the choreography specification $\mathcal{P}_{chor}$:

$$\mathbf{E}_{chor}(m_x(S, R, C), T), S \neq ws \rightarrow \mathbf{H}(m_x(S, R, C), T) \tag{15}$$

The possible interactions amongst the web service ws and the other peers will be the sets $\mathbf{HAP}^*$ satisfying equations 14 and 15.

DEFINITION 1. *Given the abductive program $\langle KB_U, \mathcal{E}_U, \mathcal{IC}_U \rangle$, where:*

- $KB_U \triangleq KB_{chor} \cup KB_{ws}$
- $\mathcal{E}_U \triangleq \mathcal{E}_{chor} \cup \mathcal{E}_{ws}$
- $\mathcal{IC}_U \triangleq \mathcal{IC}_{chor} \cup \mathcal{IC}_{ws}$

a possible interaction *amongst a web service ws in a choreography $chor$ for a goal $\mathcal{G}_U \triangleq \mathcal{G}_{ws} \cup \mathcal{G}_{chor}$ is a pair $(\mathbf{HAP}^*, \mathbf{EXP})$ such that:*

$$KB_U \cup \mathbf{HAP}^* \cup \mathbf{EXP} \models G_U \quad (16)$$
$$KB_U \cup \mathbf{HAP}^* \cup \mathbf{EXP} \models \mathcal{IC}_U \quad (17)$$
$$KB_U \cup \mathbf{HAP}^* \cup \mathbf{EXP} \models (14) \cup (15) \quad (18)$$

(where by Eq. 18 we mean that equations 14 and 15 must hold). The set $\mathbf{HAP}^*$ is also called possible history.

When the goal G_U is $true$, the empty set is typically one of the possible histories. The empty history is often of little (or no) interest for proving conformance. When the interesting histories are only those containing at least one event, the expectation of such event can be inserted as the goal G_U. Typically, we use as goal the expectation (both from the web service's viewpoint, $\mathbf{E}_{ws}$ and from the choreography's viewpoint, $\mathbf{E}_{chor}$) of the first event of an interaction. This poses no serious restriction on the types of protocols that can be tested, as if there is not a unique starting event, a dummy event can be inserted as initiation of the protocol.

EXAMPLE 1. *Suppose a choreography prescribes the following protocol:*

$$\mathbf{H}(ask(ws, R, X)) \rightarrow \mathbf{E}_{chor}(answer(R, ws, X)) \quad (19)$$
$$\mathbf{H}(answer(R, ws, X)) \rightarrow \mathbf{E}_{chor}(ack(ws, R, X))$$

while the web service's integrity constraints contain only the first rule

$$\mathbf{H}(ask(ws, R, X)) \rightarrow \mathbf{E}_{ws}(answer(R, ws, X)).$$

Let $G_U = \mathbf{E}_{ws}(ask(ws, peer, X)), \mathbf{E}_{chor}(ask(ws, peer, X))$. Given the goal G_U, the web service ws has the intention to send an ask message to the $peer$, so all the possible histories for G_U will contain the event $\mathbf{H}(ask(ws, peer, X))$. The $peer$'s behaviour is simulated through the rules in the choreography specification. Since the choreography has an expectation (generated by rule 19) $\mathbf{E}_{chor}(answer(peer, ws, X))$, this will become a happened event in all the possible histories: $\mathbf{H}(answer(peer, ws, X))$. Now, the second rule provides a choreography's expectation about the third message: the web service ws is supposed to send an ack message. But, as we can see from the web service's specification, ws does not have an expectation to send such message, so the simulation will not suppose it will comply to the choreography's expectation. So, the (only) possible history for the goal G_U is

$$\mathbf{HAP}^* = \{\mathbf{H}(ask(ws, peer, X)), \mathbf{H}(answer(peer, ws, X))\}. \quad (20)$$

In a possible history, the messages uttered by the peers comply by definition to the choreography. However, the messages uttered by the web service under test might be non conformant. The web service ws is conformant if all the possible histories are conformant. Also, ws should be able to understand all the messages in a possible history, otherwise there might be requests of other peers in the given choreography which ws is unable to serve. We require that all the possible histories satisfy both the choreography and the web service expectations.

DEFINITION 2. *A possible history $\mathbf{HAP}^*$ is* Feeble Conformant *if there exists a set $\mathbf{EXP}$ such that*[1]

$$KB_U \cup \mathbf{HAP}^* \cup \mathbf{EXP} \models \mathcal{G}_U \quad (21)$$
$$KB_U \cup \mathbf{HAP}^* \cup \mathbf{EXP} \models \mathcal{IC}_U \quad (22)$$
$$\mathbf{HAP}^* \cup \mathbf{EXP} \models \mathbf{E}_{ws}(X) \rightarrow \mathbf{H}(X) \quad (23)$$
$$\mathbf{HAP}^* \cup \mathbf{EXP} \models \mathbf{E}_{chor}(X) \rightarrow \mathbf{H}(X) \quad (24)$$

A web service is feeble conformant if all the possible histories are feeble conformant. A pair $(\mathbf{HAP}^*, \mathbf{EXP})$ is a Feeble Conformant Interaction *if $\mathbf{HAP}^*$ is a feeble conformant history and $\mathbf{EXP}$ is a set of expectations satisfying equations (21-24) which is minimal with respect to set inclusion.*

EXAMPLE 2. *Consider again the situation in Example 1. Given the possible history of Eq. 20, the expectation of the choreography for the third message (ask) remains not fulfilled, so the web service ws is clearly non conformant.*

Feeble conformance ensures that the web service ws will utter all the messages requested by the choreography, but it still does not require ws to avoid the messages forbidden by the choreography. We extend feeble conformance to a stronger version in the following.

A possible history is strong conformant if (it is feeble conformant and) all the happened events were expected both by the choreography and the web service. We include in this concept only the communications that involve the web service under observation (the other events, e.g., messages exchanged by other peers in a multi-party interaction, are always considered conformant). Also, by definition the messages sent by the web service comply to its own specifications, and symmetrically the messages sent by the other peers comply to the choreography.

DEFINITION 3. *A feeble conformant interaction $(\mathbf{HAP}^*, \mathbf{EXP})$ is also a* Strong Conformant Interaction *if the following conditions hold:*

$$\mathbf{H}(m_x(ws, R, C)) \leftrightarrow \mathbf{E}_{chor}(m_x(ws, R, C)) \quad (25)$$
$$\mathbf{H}(m_x(S, ws, C)) \leftrightarrow \mathbf{E}_{ws}(m_x(S, ws, C)). \quad (26)$$

A Strongly Conformant History *is a history for which there exists a strongly conformant interaction. A web service is* Strongly Conformant *if all the possible histories are strongly conformant.*

EXAMPLE 3. *Let us change in the previous example the specifications of the choreography and of the web service, i.e., the web service specification is*

$$\mathbf{H}(ask(ws, R, X)) \rightarrow \mathbf{E}_{ws}(answer(R, ws, X))$$
$$\mathbf{H}(answer(R, ws, X)) \rightarrow \mathbf{E}_{ws}(ack(ws, R, X))$$

and the choreography is

$$\mathbf{H}(ask(ws, R, X)) \rightarrow \mathbf{E}_{chor}(answer(R, ws, X)).$$

In this case, the web service ws has the intention to send the ack, so it will indeed send it in all the possible histories. The choreography does not prescribe this third message. The possible history becomes

$$\mathbf{HAP}^*{}_2 = \{\mathbf{H}(ask(ws, peer, X)), \mathbf{H}(answer(peer, ws, X)), \mathbf{H}(ack(ws, peer, X))\}.$$

[1] Note the difference between Equations (23-24) and Equations (14-15): Equation (23) is used as a test, and requires *all* the expectations of the web service to be fulfilled, while Equation (14) is used to *generate* the behaviour of the web service and imposes only the fulfilment of the expectations the web service has about *itself*. Analogously for Equations (24) and (15).

All the expectations of the choreography are fulfilled by one message of ws, so it is feeble conformant. However, ws will also send an unrequested message ack, that might confound the other $peer$, undermining the interoperability. There exists no expectation from the choreography for the ack message, therefore ws is non strong conformant.

3.1 Operational semantics

The operational semantics is based on two abductive proof-procedures, $\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF, developed in the SOCS project. The $\mathcal{S}$CIFF proof procedure considers the $\mathbf{H}$ events as a predicate defined by a set of incoming atoms, and is devoted to generate expectations corresponding to a given history and to check that expectations indeed match with happened events. $\mathcal{S}$CIFF was developed to check the compliance of agents to protocols [7].

3.1.1 The $\mathcal{S}$CIFF proof procedure

The $\mathcal{S}$CIFF proof procedure is based on a rewriting system transforming one node to another (or to others) as specified by rewriting steps called *transitions*. A node can be either the special node *false*, or defined by the following tuple

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

where

- R is the resolvent (initially set to the goal G),
- CS is the constraint store (à la CLP [20])
- $PSIC$ is a set of implications, derived from the $\mathcal{IC}$s
- $\mathbf{PEND}$ is the set of (pending) expectations
- $\mathbf{HAP}$ is the history of happened events
- $\mathbf{FULF}$ is a set of fulfilled expectations
- $\mathbf{VIOL}$ is a set of violated expectations.

Initial Node and Success A derivation D is a sequence of nodes $T_0 \rightarrow T_1 \rightarrow \cdots \rightarrow T_n$. Given a goal G, an initial history $\mathbf{HAP}^i$ (which can be empty) and a set of integrity constraints $\mathcal{IC}_S$, the first node is:

$$T_0 \equiv \langle \{G\}, \emptyset, \mathcal{IC}_S, \emptyset, \mathbf{HAP}^i, \emptyset, \emptyset \rangle \quad (27)$$

i.e., the resolvent is initially the query ($R_0 = \{G\}$) and the set *PSIC* contains the integrity constraints ($PSIC_0 = \mathcal{IC}_S$).

The other nodes $T_j, j > 0$, are obtained by applying the transitions defined in the next section, until no transition can be applied anymore (quiescence).

Transitions The transitions are those of the IFF proof procedure, enlarged with those of CLP [20], and with specific transitions accommodating the concepts of fulfilment of expectations, and dynamically growing history. A complete description of all the transitions is given in [7], and is not given here due to space limitations.

IFF-like transitions We borrow the transitions of the IFF, given shortly as:

Unfolding[1] $p(s) \Rightarrow (s = t_1 \wedge B_1) \vee \cdots \vee (s = t_j \wedge B_j)$

Unfolding[2] $[p(s) \wedge B \rightarrow H] \Rightarrow [s = t_1 \wedge B_1 \wedge B \rightarrow H], \ldots, [s = t_j \wedge B_1 \wedge B \rightarrow H]$

Propagation $[a(s) \wedge B \rightarrow H] \wedge a(t) \wedge R \Rightarrow [s = t \wedge B \rightarrow H] \wedge a(t) \wedge R$

Case analysis $[c \wedge B \rightarrow H] \Rightarrow c \wedge [B \rightarrow H] \vee \neg c$

Equality rewriting integrated in the CLP solver

Logical equivalence $true \rightarrow L \Rightarrow L$, $L \wedge false \Rightarrow false$, $L \wedge true \Rightarrow L, \ldots$

where *a* is either an abducible literal in $\mathbf{PEND}_k$ or a $\mathbf{H}$ event in the history $\mathbf{HAP}_k$, p is a predicate defined by clauses $p(t_1) \leftarrow B_1$, $\ldots$, $p(t_j) \leftarrow B_j$, and c is a constraint.

IFF transitions have been extended for dealing with CLP constraints (unification, in particular, is dealt with by the constraint solver).

Dynamically growing history The $\mathcal{S}$CIFF proof-procedure takes as input a stream of events, that are considered in an external queue. The happening of events is dealt with by a transition *Happening*, that takes an event $\mathbf{H}(Event)$ from the external queue and puts it in the history $\mathbf{HAP}$. Transition *Happening* is applicable only if an $Event$ such that $\mathbf{H}(Event) \notin \mathbf{HAP}$ is in the external queue. Formally, from a node N_k transition *Happening* produces a single successor $\mathbf{HAP}_{k+1} = \mathbf{HAP}_k \cup \{\mathbf{H}(Event)\}$.

At the end of the stream, we apply a transition *closure* that declares closed the set of happened events. After the application of *closure*, the set of happened events cannot grow further, so a closed world assumption is possible.

Fulfilment The *fulfilment* transition is devoted to prove that an expectation $\mathbf{E}(X, T_x)$ has been fulfilled by an event $\mathbf{H}(Y, T_y)$. Two nodes are generated: in the first, X and Y are unified, and the expectation is fulfilled (i.e., it is moved to the set $\mathbf{FULF}$); in the second the new constraint $X \neq Y$ is added to the constraint store CS.

Formally, *Fulfilment* is applicable to a node N as follows:

$$\mathbf{PEND}_k = \mathbf{PEND}' \cup \{\mathbf{E}(E_1)\},$$
$$\mathbf{HAP}_k = \mathbf{HAP}' \cup \{\mathbf{H}(E_2)\}$$

and generates two nodes, N^1 and N^2; in node N^1 we assume that the expectation and the happened event unify,

- $\mathbf{PEND}^1_{k+1} = \mathbf{PEND}'$
- $\mathbf{FULF}^1_{k+1} = \mathbf{FULF}_k \cup \{\mathbf{E}(E_1)\}$
- $CS^1_{k+1} = CS_k \cup \{E_1 = E_2\}$

and in N^2 we hypothesise the opposite:

- $\mathbf{PEND}^2_{k+1} = \mathbf{PEND}_k$
- $\mathbf{FULF}^2_{k+1} = \mathbf{FULF}_k$
- $CS^2_{k+1} = CS_k \cup \{E_1 \neq E_2\}$

Note that N^2 is not necessarily a failure node, as $\mathbf{E}(E_1)$ might be confirmed by other events.

Violation. Violation of $\mathbf{E}$ expectations can be proved only if no event will ever match with the expected one. In other words, we apply this transition only if the history is *closed*, i.e., after application of the *closure* transition. Given a node:

1. $\mathbf{PEND}_k = \{\mathbf{E}(X, T)\} \cup \mathbf{PEND}'$
2. $\mathbf{HAP}_k$ is closed
3. $\forall E_1, T_1 : \mathbf{H}(E_1, T_1) \in \mathbf{HAP}_k$, $CS_k \cup \{(E_1, T_1) = (X, T)\} \models false$

transition *Violation* $\mathbf{E}$ is applicable and creates a node

$$\begin{array}{rcl} \mathbf{PEND}_{k+1} & = & \mathbf{PEND}', \\ \mathbf{VIOL}_{k+1} & = & \mathbf{VIOL}_k \cup \{\mathbf{E}(X, T)\}. \end{array}$$

Operationally, one can avoid checking condition 3 (i.e., (X, T) does not unify with every event in the history) by choosing a preferred order of application of the transitions. By applying *Violation* $\mathbf{E}$ only if no other transition is applicable, the check can be safely

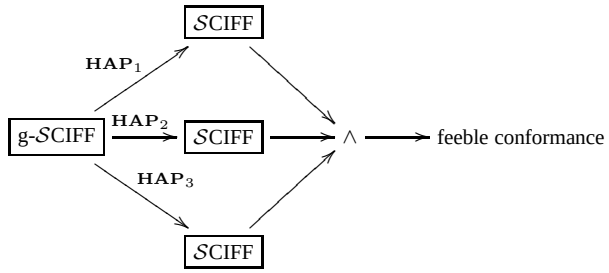

Figure 3. A^{l}LoWS architecture

avoided, as the test of fulfilment is already performed by *Fulfilment* **E**.

The $\mathcal{S}$CIFF proof procedure terminates [17] for acyclic programs.

3.1.2 The g-$\mathcal{S}$CIFF proof-procedure

The g-$\mathcal{S}$CIFF proof procedure, instead, considers **H** as an abducible predicate and aims at finding both the set of expectations and the history that fulfils some property requested as goal. g-$\mathcal{S}$CIFF has been used to prove properties of protocols, such as security protocols [3], and others. It contains the same rules in $\mathcal{S}$CIFF; in the version adopted in this paper, we also added as integrity constraints the rules (14) and (15). In g-$\mathcal{S}$CIFF, we generate events, so we do not apply the *closure* transition, that enforces a closed world assumption on the set of happened events. We call *open* a derivation in which the *closure* transition is not applied.

3.1.3 A^{l}LoWS operational semantics

In order to prove conformance, we apply the two proof procedures to the two phases implicitly defined in the previous section. We decompose the proof of feeble conformance into a *generative* phase and a *test* phase. In the generative phase, we generate, by means of g-$\mathcal{S}$CIFF, all the possible histories. Of course, those histories need not be generated as ground histories (the set of ground histories can be infinite), but intensionally: the **H** events can contain variables, possibly with constraints à la Constraint Logic Programming [20].

In the test phase, we check with $\mathcal{S}$CIFF the compliance of the generated histories both with respect to the web service and the choreography specifications. If all the histories are conformant, the web service is feeble conformant to the choreography, as depicted in Figure 3. Otherwise, if there exists at least one history that is not conformant, the web service is not (feeble) conformant.

Finally, we can prove strong conformance by checking that all the happened events were indeed expected both by the choreography and by the web service. This can in principle be done by using a version of $\mathcal{S}$CIFF that does not have abducibles, but it can also be performed during the second phase ($\mathcal{S}$CIFF) by adopting the same technique used in the fulfilment transition: if a **H** event matches both with an $\mathbf{E}_{ws}$ and a $\mathbf{E}_{chor}$ expectation, it is labelled *expected*; after the application of the *closure* transition, all events that were not expected are considered *unexpected*, showing that the web service was not strongly conformant.

3.2 Examples

Baldoni et al. [8] show various examples of conformance and non-conformance of a web service to a choreography, and propose a framework based on Finite State Automata, to prove conformance. We show how their examples are addressed in A^{l}LoWS, based on Computational Logics.

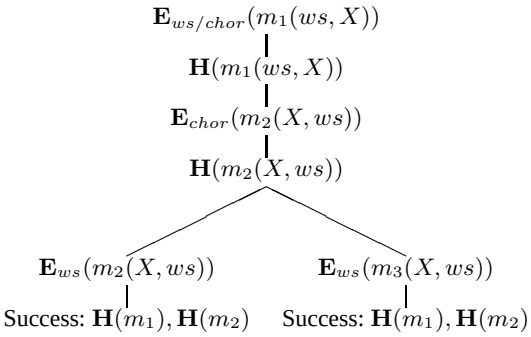

Figure 4. g-$\mathcal{S}$CIFF derivation for Example 3.2.1

3.2.1 Web service with more capabilities

The first example taken by [8] is the following. The choreography specification defines only one allowed interaction: ws sends a message m_1 and the other peer will reply with m_2:

$$\mathbf{H}(m_1(ws, X)) \rightarrow \mathbf{E}_{chor}(m_2(X, ws))$$

The web service specification is wider: after the first message the web service accepts as reply either m_2 or m_3:

$$\mathbf{H}(m_1(ws, X)) \rightarrow \mathbf{E}_{ws}(m_2(X, ws)) \vee \mathbf{E}_{ws}(m_3(X, ws))$$

In this case, Baldoni et al. state that the web service is conformant. In fact, in a legal conversation the message m_3 will never be received by ws, so the interoperability is ensured.

The g-$\mathcal{S}$CIFF proof procedure is started with the goal containing the expectation, both from the web service's and from the choreography's viewpoint, of the first event:

$$G_U = \mathbf{E}_{ws}(m_1(ws, X)) \wedge \mathbf{E}_{chor}(m_1(ws, X)).$$

g-$\mathcal{S}$CIFF in this case derives that there exists one possible history: $\{m_1, m_2\}$. A simplified representation of the derivation[2] is reported in Fig. 4. There are two alternative sets of expectations from the web service viewpoint, $\{\mathbf{E}_{ws}(m_1), \mathbf{E}_{ws}(m_2)\}$ and $\{\mathbf{E}_{ws}(m_1), \mathbf{E}_{ws}(m_3)\}$, but in the first phase the correspondence between expectations and happened events is not required (open derivation). In the second phase, the (only) generated history is checked; since there exists one set of expectations that is fulfilled by the generated history, the web service is considered feeble conformant. Since in the generated history there are no unexpected events, the web service is also strong conformant.

3.2.2 Missing capability

The second example by Baldoni et al. is dual to the first: the web service accepts as reply only m_2

$$\mathbf{H}(m_1(ws, X)) \rightarrow \mathbf{E}_{ws}(m_2(X, ws))$$

while the choreography defines as valid two interactions

$$\mathbf{H}(m_1(ws, X)) \rightarrow \mathbf{E}_{chor}(m_2(X, ws)) \vee \mathbf{E}_{chor}(m_4(X, ws))$$

In this case, g-$\mathcal{S}$CIFF provides two possible histories: $\{\mathbf{H}(m_1), \mathbf{H}(m_2)\}$ and $\{\mathbf{H}(m_1), \mathbf{H}(m_4)\}$. In the second phase, $\mathcal{S}$CIFF detects non conformance of the history $\{\mathbf{H}(m_1), \mathbf{H}(m_4)\}$, because the web service's expectation $\mathbf{E}_{ws}(m_2(S, ws, C))$ remains unfulfilled in all possible derivation paths. This means that the web service is blocked waiting for message m_2, and will not process other messages, so it is non (feeble) conformant.

[2] The derivation does not report all the events and expectations, but for each node it gives only those that are added. We use $\mathbf{E}_{ws/chor}$ to indicate that the event is expected both by the choreography and by the web service.

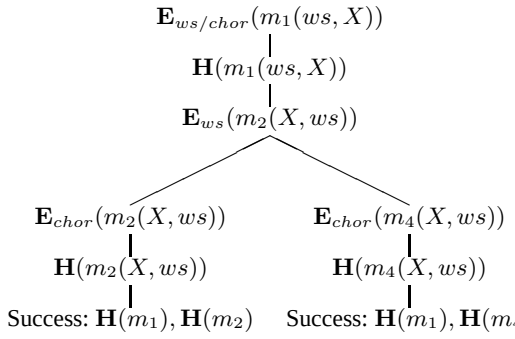

Figure 5. g-$\mathcal{S}$CIFF derivation for Example 3.2.2

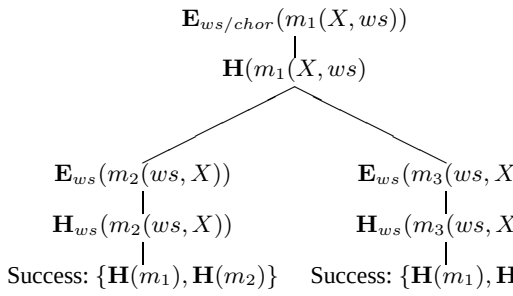

Figure 6. g-$\mathcal{S}$CIFF derivation for Example 3.2.3

3.2.3 Wrong reply

In the third example the web service assumes to have the freedom to reply either m_2 or m_3 to a question m_1

$$\mathbf{H}(m_1(X,ws)) \rightarrow \mathbf{E}_{ws}(m_2(ws,X)) \vee \mathbf{E}_{ws}(m_3(ws,X))$$

while the choreography does not grant such a freedom: only m_2 is legal

$$\mathbf{H}(m_1(X,ws)) \rightarrow \mathbf{E}_{chor}(m_2(ws,X))$$

This case is judged non conformant by Baldoni et al., as there might be paths in which the web service utters the forbidden message m_3. g-$\mathcal{S}$CIFF computes two possible histories ($\{\mathbf{H}(m_1), \mathbf{H}(m_2)\}$ and $\{\mathbf{H}(m_1), \mathbf{H}(m_3)\}$). The first is compliant, according to $\mathcal{S}$CIFF, while in the second the choreography's expectation $\mathbf{E}_{chor}(m_2)$ remains pendent.

3.2.4 Predefined answer

The dual of example 3.2.3 is when the choreography lets the web service choose to reply m_2 or m_3 to a question m_1,

$$\mathbf{H}(m_1(X,ws)) \rightarrow \mathbf{E}_{chor}(m_2(ws,X)) \vee \mathbf{E}_{chor}(m_3(ws,X))$$

while the web service sticks to the reply m_2

$$\mathbf{H}(m_1(X,ws)) \rightarrow \mathbf{E}_{ws}(m_2(ws,X)).$$

Again, A^{l}LoWS provides a correct proof: g-$\mathcal{S}$CIFF gives one possible history, which is reported (feeble and strong) conformant by $\mathcal{S}$CIFF in the second phase.

Thus, in all the examples by Baldoni et al., A^{l}LoWS provides the same answer proposed in [8].

3.2.5 Forbidden message

In all the previous cases, feeble and strong conformance coincide. However, there might be instances in which the choreography assumes that the interaction has finished, while the web service continues sending messages. For example, the choreography expects only one message:

$$G_{chor} = \mathbf{E}_{chor}(m_1(X,ws))$$

while ws will send back a message m_2:

$$\mathbf{H}(m_1(X,ws)) \rightarrow \mathbf{E}_{ws}(m_2(ws,X)).$$

In this case, the only possible history is $\mathbf{HAP}^* = \{\mathbf{H}(m_1), \mathbf{H}(m_2)\}$. This history does not leave any pending expectations, both from the choreography and from the web service's viewpoints, so ws is judged feeble conformant. However, the message m_2 was not expected by the choreography, and ws is not strong conformant.

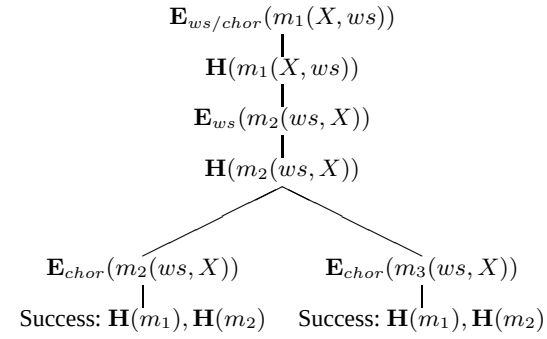

Figure 7. g-$\mathcal{S}$CIFF derivation for Example 3.2.4

We now propose other examples that highlight the enhanced expressive power provided by computational logics, in particular the use of constraints, that are embedded in the $\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF proof procedures.

3.2.6 Mutual exclusion

Many protocols include mutual exclusion between choices: for instance a choreography might prescribe that if a given condition on a message m_1 holds, a message m_2 should be exchanged, otherwise another message m_3 should be sent. In A^{l}LoWS, conditions can be expressed by means of constraints (either the ones predefined in the underlying solver, i.e., CLP(FD), or user-defined) or by means of defined predicates. As a simple example, consider the following: the choreography prescribes to reply either m_2 or m_3, depending on the content of the previous message m_1:

$$\begin{array}{rcl} \mathbf{H}(m_1(X,ws,C)) & \rightarrow & \mathbf{E}_{chor}(m_2(ws,X,C_2)), C > 0 \\ & \vee & \mathbf{E}_{chor}(m_3(ws,X,C_3)), C \leq 0 \end{array}$$

while the web service always replies m_2:

$$\mathbf{H}(m_1(X,ws,C)) \rightarrow \mathbf{E}_{ws}(m_2(ws,X,C_2))$$

g-$\mathcal{S}$CIFF generates two possible histories, with variables and constraints upon the variables (Fig 8). In both the messages m_1 and m_2 are generated, but while in the first the proof procedure assumes that C takes a value greater than 0, in the second C is non positive. In the second phase, $\mathcal{S}$CIFF takes as input both the happened events and the constraint store, and accepts as conformant the first history, while discarding as non-conformant the second.

Notice that constraints scope is not restricted only to variables in the content, but might involve all the variables in the message, including time. The choreography might contain conditions on deadlines (if you receive a message within 5 minutes answer *ok*, otherwise reply *too late*), or on participants (if the sender is authorised, etc.).

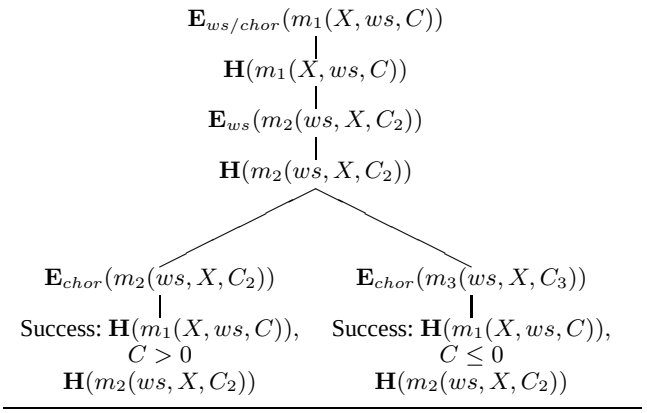

Figure 8. g-$\mathcal{S}$CIFF derivation for Example 3.2.6

3.2.7 Deadlines

Suppose that the choreography specifies a deadline for the receipt of a given message m_2:

$$\begin{array}{rcl}\mathbf{H}(m_1(X, ws, C_1), T_1) & \rightarrow & \mathbf{E}_{chor}(m_2(ws, X, C_2), T_2) \\ & \wedge & T_2 < T_1 + \delta_{chor}\end{array}$$

The web service ws, however, replies within a deadline that might be different:

$$\begin{array}{rcl}\mathbf{H}(m_1(X, ws, C_1), T_1) & \rightarrow & \mathbf{E}_{ws}(m_2(ws, X, C_2), T_2) \\ & \wedge & T_2 < T_1 + \delta_{ws}.\end{array}$$

In this case, the only possible history is

$$\begin{array}{rcl}\mathbf{HAP}^* & = \{ & \mathbf{H}(m_1(X, ws, C_1), T_1), \\ & & \mathbf{H}(m_2(ws, X, C_2), T_2), T_2 < T_1 + \delta_{ws} \quad \}\end{array}$$

Applying $\mathcal{S}$CIFF to the generated history, we get the expectation $\mathbf{E}_{chor}(m_2(ws, X, C_2), T_2) \wedge T_2 < T_1 + \delta_{chor}$; this expectation matches with the second item of the $\mathbf{HAP}^*$ history if a further condition holds: $T_2 < T_1 + \delta_{chor}$. Coherently with the philosophy of Constraint Logic Programming, $\mathcal{S}$CIFF provides this constraint in output, as a conditional answer: the web service is conformant provided that the answer arrives before the deadline in the choreography specification. Depending on the propagation performed by the adopted constraint solver, the information provided could be even more significant. For example, if the two values δ_{ws} and δ_{chor} are ground, a CLP(FD) solver would provide the conditional answer only if necessary, i.e., if $\delta_{chor} \leq \delta_{ws}$, (the deadline imposed by the choreography is more tight than the one the web service will meet).

4. A test conformance example

In this section we exemplify the proposed approach by using a simple choreography specification, shown in Fig. 1. The interaction is initiated by a $User$ that asks the Flight Service FS to book a flight. If there are seats available on the plane, the FS will reply with *flightOffer*, specifying the $Price$ for booking the seat. Otherwise, the FS replies with *notAvailable*.

The *offer* can be accepted (with *ackOffer*) or refused (with *nAckOffer*) by the $User$. If the *offer* is accepted, the flight company will book the seat. The $User$, after booking, has still the freedom to $Cancel$ the booking. Otherwise, it will issue a payment order ($payment$) to the $Bank$, that will send the notification ($notifyPayment$) to the creditor, the FS.

When the FS has received both the booking order (*ackOffer*) and the payment (*notifyPayment*), it will normally issue the $flightTicket$ to the $User$; however, the FS retains the right to refuse the ticket and send a $flightCancelled$ message in case of problems (e.g., overbooking or other error conditions).

Fig. 2 shows the behavioural interface of a Flight Server web service; the specification in terms of ICs is in Spec. 2.2. The FS establishes that the late payment is an error condition, and will cancel the booking if the payment notification does not arrive within δ time units after the booking.

In the next section, we show how the conformance of *fs* is proven in A^{l}LoWS.

4.1 Conformance of the Flight Service

The test of conformance of the Flight Service fs is performed by generating, through g-$\mathcal{S}$CIFF, the set of the possible histories. The g-$\mathcal{S}$CIFF derivation provides five possible histories:

$$\begin{array}{rl}\mathbf{HAP}^*{}_1 = \{ & \mathbf{H}(request(U, fs, F), T_r), \\ & \mathbf{H}(\textit{offer}(fs, C, F, P), T_o), \\ & \mathbf{H}(\textit{ackOffer}(C, fs, F, P), T_a), \\ & \mathbf{H}(\textit{payment}(C, B, P, fs), T_p), \\ & \mathbf{H}(\textit{notifyPayment}(B, fs, P)), T_n) \wedge T_n > T_a + \delta \\ & \mathbf{H}(\textit{flightCancelled}(fs, C, F), T_c)\}, \\ \mathbf{HAP}^*{}_2 = \{ & \mathbf{H}(request(U, fs, F), T_r), \\ & \mathbf{H}(\textit{offer}(fs, C, F, P), T_o), \\ & \mathbf{H}(\textit{ackOffer}(C, fs, F, P), T_a), \\ & \mathbf{H}(\textit{payment}(C, B, P, fs), T_p), \\ & \mathbf{H}(\textit{notifyPayment}(B, fs, P)), T_n) \wedge T_n \leq T_a + \delta \\ & \mathbf{H}(\textit{flightTicket}(fs, C, F), T_t)\}, \\ \mathbf{HAP}^*{}_3 = \{ & \mathbf{H}(request(U, fs, F), T_r), \\ & \mathbf{H}(\textit{offer}(fs, C, F, P), T_o), \\ & \mathbf{H}(\textit{ackOffer}(C, fs, F, P), T_a), \\ & \mathbf{H}(\textit{cancel}(C, fs, F), T_c), \\ & \mathbf{H}(\textit{flightCancelled}(fs, C, F))\}, \\ \mathbf{HAP}^*{}_4 = \{ & \mathbf{H}(request(U, fs, F), T_r), \\ & \mathbf{H}(\textit{offer}(fs, C, F, P), T_o), \\ & \mathbf{H}(\textit{nAckOffer}(C, fs, F, P), T_a)\}, \\ \mathbf{HAP}^*{}_5 = \{ & \mathbf{H}(request(U, fs, F), T_r), \\ & \mathbf{H}(\textit{notAvailable}(fs, C, F, P), T_n)\}.\end{array}$$

Two of the histories include time constraints. All the possible histories are trivially conformant: they satisfy both the expectations of the choreography, and those of the web service fs. Thus, fs is feeble conformant. Moreover, all the generated events are expected, and this shows that fs is also strong conformant.

4.2 Conformance of the User web service

Suppose now that the user web service has the behavioural interface in Spec. 4.1.

Note that the $User$ implements a policy for deciding whether to accept (*ackOffer*) or refuse (*nAckOffer*) the *offer* of the FS: if the $Price$ is less than a max quota, then the *offer* is accepted (and declined otherwise). Also, this $User$ web service does not have expectations on the $Bank$'s reply: in fact, in this choreography, the $Bank$ does not need to notify the $User$ about the payment. Finally, $User$ always expects to receive a ticket after paying.

Specification 4.1 The behavioural interface of a web service that is not conformant for the role of $User$

$$\begin{aligned}&\mathbf{H}(request(user, FS, Flight), T_r)\\ \rightarrow&\mathbf{E}_{user}(\textit{offer}(FS, user, Flight, Price), T_o)\\ \vee&\mathbf{E}_{user}(\textit{notAvailable}(FS, user, Flight), T_{na})\end{aligned} \tag{28}$$

$$\begin{aligned}&\mathbf{H}(\textit{offer}(FS, user, Flight, Price), T_o)\\ \rightarrow&\mathbf{E}_{user}(\textit{ackOffer}(user, FS, Flight, Price), T_a)\\ \wedge&\ Price \leq max\\ \vee&\mathbf{E}_{user}(\textit{nAckOffer}(user, FS, Flight, Price), T_a)\\ \wedge&\ Price > max\end{aligned} \tag{29}$$

$$\begin{aligned}&\mathbf{H}(\textit{ackOffer}(user, FS, Flight, Price), T_a)\\ \rightarrow&\mathbf{E}_{user}(\textit{payment}(user, Bank, Price, FS), T_f)\\ \vee&\mathbf{E}_{user}(\textit{cancel}(user, FS, Flight), T_f)\end{aligned} \tag{30}$$

$$\begin{aligned}&\mathbf{H}(\textit{ackOffer}(user, FS, Flight, Price), T_a)\\ \wedge&\ \mathbf{H}(\textit{payment}(user, Bank, Price, FS), T_p)\\ \rightarrow&\mathbf{E}_{user}(\textit{flightTicket}(FS, user, Flight), T_f)\end{aligned} \tag{31}$$

$$\begin{aligned}&\mathbf{H}(\textit{cancel}(user, FS, Flight), T_a)\\ \rightarrow&\mathbf{E}_{user}(\textit{flightCancelled}(FS, user, Flight), T_f)\end{aligned} \tag{32}$$

$$\begin{aligned}&\mathbf{H}(\textit{payment}(User, Bank, Price, Creditor), T_p)\\ \rightarrow&\mathbf{E}_{user}(\textit{notifyPayment}(Bank, Creditor, Price), T_n)\end{aligned} \tag{33}$$

Invoked with the goal $\mathbf{E}_{chor}(request) \wedge \mathbf{E}_{user}(request)$, g-$\mathcal{S}$CIFF generates the possible histories, that in this case are:

$$\begin{aligned}\mathbf{HAP}^*{}_1 = \{&\mathbf{H}(request(U, fs, F), T_r),\\ &\mathbf{H}(\textit{offer}(fs, C, F, P), T_o),\\ &\mathbf{H}(\textit{ackOffer}(C, fs, F, P), T_a) \wedge P \leq max,\\ &\mathbf{H}(\textit{payment}(C, B, P, fs), T_p),\\ &\mathbf{H}(\textit{notifyPayment}(B, fs, P)), T_n),\\ &\mathbf{H}(\textit{flightCancelled}(fs, C, F), T_c)\},\\ \mathbf{HAP}^*{}_2 = \{&\mathbf{H}(request(U, fs, F), T_r),\\ &\mathbf{H}(\textit{offer}(fs, C, F, P), T_o),\\ &\mathbf{H}(\textit{ackOffer}(C, fs, F, P), T_a) \wedge P \leq max,\\ &\mathbf{H}(\textit{payment}(C, B, P, fs), T_p),\\ &\mathbf{H}(\textit{notifyPayment}(B, fs, P)), T_n),\\ &\mathbf{H}(\textit{flightTicket}(fs, C, F), T_t)\},\\ \mathbf{HAP}^*{}_3 = \{&\mathbf{H}(request(U, fs, F), T_r),\\ &\mathbf{H}(\textit{offer}(fs, C, F, P), T_o),\\ &\mathbf{H}(\textit{ackOffer}(C, fs, F, P), T_a) \wedge P \leq max,\\ &\mathbf{H}(\textit{cancel}(C, fs, F), T_c),\\ &\mathbf{H}(\textit{flightCancelled}(fs, C, F), T_{fc})\},\\ \mathbf{HAP}^*{}_4 = \{&\mathbf{H}(request(U, fs, F), T_r),\\ &\mathbf{H}(\textit{offer}(fs, C, F, P), T_o) \wedge P > max,\\ &\mathbf{H}(\textit{nAckOffer}(C, fs, F, P), T_a)\},\\ \mathbf{HAP}^*{}_5 = \{&\mathbf{H}(request(U, fs, F), T_r),\\ &\mathbf{H}(\textit{notAvailable}(fs, C, F, P), T_n)\}.\end{aligned}$$

However, in the second phase, $\mathcal{S}$CIFF applied to the history $\mathbf{HAP}^*{}_1$ signals that the $user$'s expectation $\mathbf{E}_{user}(\textit{flightTicket}(FS, user, Flight), T_t)$ remains unfulfilled, proving that $user$ is not (feeble) conformant. In fact, this $user$ web service undermines the interoperability with other web services conformant with the same choreography. As an example, we can easily see that in case the $Bank$ does not provide the *notifyPayment* within the deadline imposed by *fs* (Spec. 2.2), the two web services are unable to complete the choreography.

Concerning efficiency, an experimental evaluation of $\mathcal{S}$CIFF was published in [1]. The examples presented in this section were solved by A^{I}LoWS in reasonable time: feeble conformance of the *fs* was proven in 80.085 seconds, and its strong conformance in 80.666 seconds. The *user* was proven not feeble conformant in 3.305 s, and not strong conformant in 6.399 s. All experiments were performed on a Pentium M715, 1 GHz, 512 MB RAM. The examples in Section 3.2 were solved in negligible time (less than 0.1 seconds).

5. RELATED WORK

A number of languages for specifying service choreographies and testing "a priori" and/or "run-time" conformance have been proposed in the literature. Two examples of these languages are represented by state machines [10] and Petri nets [12].

Our work is highly inspired by Baldoni et al. [8]. We adopt, like them, a Multi-Agent Systems point of view, in defining a priori conformance in order to guarantee interoperability. As in [8], we give an interpretation of the a-priori conformance as a property that relates two formal specifications: the global one determining the conversations allowed by the choreography and the local one related to the single web service. But, while in [8] a global interaction protocol is represented as a finite state automation, we claim that the formalisms and technologies developed in the area of Computational Logic in providing a declarative representation of the social knowledge could be applied also in the context of choreographies with respect to the conversation aspects and conformance checking of Web Services. This paper can be considered as a first step in this direction. For example, a difference between our work and [8] can be found in the number of parties as they can manage only 2-party choreographies while we do not impose any limit. We also manage concurrency, which they do not consider at the moment.

Endriss et al. [13, 14] apply a formalism based on computational logic to the a-priori conformance in the MAS field. Their formalism is similar to the one we propose, but they restrict their analysis to a particular type of protocols (named *shallow protocols*). Doing this, they address only 2-party interactions, without the possibility of expressing conditions over the content of the exchanged messages, and without considering concurrency. While the two works agree on the notion of strong/exhaustive conformance, we have dual notions of feeble/weak conformance: in [13, 14] weak conformant is an agent that does not perform forbidden actions, but we have no knowledge on its capability to perform requested actions. Dually, in this work, we call feeble conformant an agent that does execute all the required actions, but there is no knowledge on its ability to avoid forbidden actions.

The verification of concurrent systems has been widely studied in many different research areas. Typical properties that are subject of verification are *safety* (such as deadlock avoidance) and *liveness* (such as starvation avoidance) properties. Our work is not particularly focussed on these properties, but rather on the verification of interoperability. In a sense, interoperability can be considered both a safety and a liveness property: each web service should never send an undesirable message and should always end up in sending the right message. Amongst the many works on verification of concurrent systems, we cite Tempo [19, 18], a declarative concurrent

language based on first order logics. Tempo uses an ordering relation which propagates information amongst events through transitivity and irreflexivity rules, and uses a special value *eternity* for indicating events that will never actually happen (due, e.g., to deadlock). A^{l}LoWS is based on a generic CLP solver, that can include the same rules in Tempo; in particular the current implementation is based on CLP(FD), which propagates through arc-consistency, that entails the transitivity and irreflexivity rules of the *precedence* constraint. Moreover, CLP(FD) also contains equality constraints, that are not present in [19].

Fagin et al. [16] study the interaction of knowledge-based systems, together with the environment. The formal definition of the aggregated system contains an admissibility condition that should be satisfied; in this sense the work can be thought as proving safety conditions. The admissibility condition is given in temporal logics, while our work uses a constraint solver. These different logics provide different expressivity: in A^{l}LoWS we can naturally express deadlines, that cannot be represented in temporal logics; on the other hand, there might be some temporal logics formulas that cannot be easily represented with constraints.

The use of abduction for verification was also explored in other work. Noteworthily, Russo et al. [26] use an abductive proof procedure for analysing event-based requirements specifications. Their method uses abduction for analysing the correctness of specifications, while our system is more focussed on the check of compliance/conformance of a set of web services.

In [24], the authors tackle the problem of verifying (general and specific) properties of a Service obtained from the composition of many web services. Each web service specification (written in BPEL4WS) is translated in a *labelled state transition system*; then, by applying a composition operator, they get the state transition system representing the composed service. Finally, model checking techniques are applied to this latter model, to the end of verifying the properties. Note that, by appropriately defining and extending the validity of the composed state transition system, they tackle different communication models (synchronous, ordered asynchronous and unordered asynchronous communications) that appear to be quite common in real cases.

Both our work and [24] focus on verifying interoperability, but while we concentrate on the interoperability issue of a single web service w.r.t. a global choreography, in [24] the authors address the interoperability of a group of web services, referring only to the interface behavior of each web service. However, we share the same intuition that *"the situation where some messages can be emitted without being ever consumed should not occur in valid composition."*. The authors address also the problem of proving properties about the possible interactions between a group of interoperating web services, by means of model checking techniques. We are currently working on this issue, and some preliminary results have been published in [2].

6. Conclusions and Future Work

This paper represents a first step in checking the conformance of web services to choreographies in computational logics. We have proposed a framework, called A^{l}LoWS, for defining web services and choreographies, and checking conformance of web services to choreographies. We have shown various examples of the expressivity of computational logics, including reasoning on constraints, deadlines, and other structures that are not currently easily addressed by classical tools.

We are aware that many issues should be addressed in the future. For example, A^{l}LoWS generates the possible histories, though in intensional version, so it cannot currently handle histories of unbounded length, such as those possible in choreographies containing *cycles*. In such a case, the resulting program could be non acyclic, so the proof of termination might not hold [17]. Those choreographies might be tested using an iterative deepening search strategy. Although still unable to prove conformance for infinite length histories, A^{l}LoWS could be used to test the conformance of all the histories up to N messages, for any given N.

We are currently developing a graphical language to define choreographies, and an interpreter that will generate automatically the integrity constraints defining a choreography. The final aim of our work will be to take the concrete web service specifications used in reality (e.g., BPEL4WS) and translate them into the language interpreted by A^{l}LoWS; we are currently studying such issues.

Acknowledgments

This work has been partially supported by the MIUR PRIN 2005 projects *Specification and verification of agent interaction protocols*, and *Constraints and preferences as a unifying formalism for system analysis and solution of real-life problems*.

References

[1] M. Alberti and F. Chesani. The computational behaviour of the $\mathcal{S}$CIFF abductive proof procedure and the SOCS-SI system. *Intelligenza Artificiale*, II(3):45–51, 2005.

[2] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Security protocols verification in abductive logic programming: a case study. In O. Dikenelli, M.-P. Gleizes, and A. Ricci, editors, *Proceedings of ESAW'05*, pages 283–295. Department of Computer Engineering Ege University, 2005.

[3] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Security protocols verification in Abductive Logic Programming: A case study. In A. Pettorossi, M. Proietti, and V. Senni, editors, *CILC 2005 - Convegno Italiano di Logica Computazionale*. Università degli Studi di Roma Tor Vergata, June 21-22 2005.

[4] 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.

[5] M. Alberti, A. Ciampolini, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. A social ACL semantics by deontic constraints. In V. Mařík, J. Müller, and M. Pěchouček, editors, *Multi-Agent Systems and Applications III. Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, volume 2691 of *Lecture Notes in Artificial Intelligence*, pages 204–213, Prague, Czech Republic, June 16–18 2003. Springer-Verlag.

[6] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Specification and verification of agent interactions using social integrity constraints. *Electronic Notes in Theoretical Computer Science*, 85(2), 2003.

[7] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. The sciff 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.

[8] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Verifying the conformance of web services to global interaction protocols: A first step. In M. Bravetti, L. Kloul, and G. Zavattaro, editors, *EPEW/WS-FM*, volume 3670 of *LNCS*. Springer, 2005.

[9] A. Barros, M. Dumas, and P. Oaks. A critical overview of the web services choreography description language (WS-CDL). *BPTrends*, 2005.

[10] B. Benattallah, F. Casati, F. Toumani, and R. Hamadi. Conceptual modeling of web service conversations. 2681:449–467, 2003.

[11] F. Chesani, P. Mello, M. Montali, M. Alberti, M. Gavanelli, E. Lamma, and S. Storari. Abduction for specifying and verifying web service choreographies. In J. L. Ambite, J. Blythe, J. Koehler, S. McIlraith, M. Pistore, and B. Srivastava, editors, *4th International Workshop on AI for Service Composition, Riva del Garda, Trento, Italy, Aug 28 - Sept 1, 2006*, Aug. 2006. To appear.

[12] R. Dijkman and M. Dumas. Service-oriented design: A multi-viewpoint approach. *International Journal of Cooperative Information Systems*, 13(4):337–378, 2004.

[13] U. Endriss, N. Maudet, F. Sadri, and F. Toni. Protocol conformance for logic-based agents. In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico (IJCAI-03)*. Morgan Kaufmann Publishers, Aug. 2003.

[14] U. Endriss, N. Maudet, F. Sadri, and F. Toni. Logic-based agent communication protocols. In F. Dignum, editor, *Advances in Agent Communication*, volume 2922 of *LNAI*, pages 91–107. Springer-Verlag, 2004.

[15] K. Eshghi. Abductive planning with the event calculus. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington*, Cambridge, MA, 1988. MIT Press.

[16] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. Knowledge-based programs. *Distributed Computing*, 10(4):199–225, 1997.

[17] M. Gavanelli, E. Lamma, and P. Mello. Proof of properties of the SCIFF proof-procedure. Technical Report CS-2005-01, Computer science group, Dept. of Engineering, Ferrara University, 2005. http://www.ing.unife.it/informatica/tr/.

[18] S. Gregory. Derivation of concurrent algorithms in Tempo. In M. Proietti, editor, *Logic Programming Synthesis and Transformation, 5th International Workshop, LOPSTR'95, Utrecht, The Netherlands, September 20-22, 1995, Proceedings*, volume 1048 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 1996.

[19] S. Gregory and R. Ramirez. Tempo: A declarative concurrent programming language. In L. Sterling, editor, *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, June 13-16, 1995, Tokyo, Japan*, pages 515–529. MIT Press, 1995.

[20] J. Jaffar and M. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19-20:503–582, 1994.

[21] J. Jaffar, M. Maher, K. Marriott, and P. Stuckey. The semantics of constraint logic programs. *Journal of Logic Programming*, 37(1-3):1–46, 1998.

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

[23] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. *Web services choreography description language version 1.0*, 2005. Available at http://www.w3.org/TR/ws-cdl-10.

[24] R. Kazhamiakin and M. Pistore. A parametric communication model for the verification of bpel4ws compositions. In M. Bravetti, L. Kloul, and G. Zavattaro, editors, *EPEW/WS-FM*, volume 3670 of *Lecture Notes in Computer Science*, pages 318–332. Springer, 2005.

[25] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd extended edition, 1987.

[26] A. Russo, R. Miller, B. Nuseibeh, and J. Kramer. An abductive approach for analysing event-based requirements specifications. In P. Stuckey, editor, *Logic Programming, 18th International Conference, ICLP 2002*, volume 2401 of *Lecture Notes in Computer Science*, pages 22–37, Berlin Heidelberg, 2002. Springer-Verlag.

[27] Societies Of ComputeeS (SOCS): a computational logic model for the description, analysis and verification of global and open societies of heterogeneous computees. IST-2001-32530. Home page: `http://lia.deis.unibo.it/Research/SOCS/`.