A priori conformance verification for guaranteeing interoperability in open environments *

Matteo Baldoni, Cristina Baroglio, Alberto Martelli, and Viviana Patti

Dipartimento di Informatica — Università degli Studi di Torino
C.so Svizzera, 185 — I-10149 Torino (Italy)
{baldoni,baroglio,mrt,patti}@di.unito.it

* This paper is registred as Technical Report no. 93/06 at the Dipartimento di Informatica, Università degli Studi di Torino, c.so Svizzera, 185, I–10149 Torino (Italy).

Abstract. An important issue, in open environments like the web, is guaranteeing the interoperability of a set of services. When the interaction scheme that the services should follow is given (e.g. as a choreography or as an interaction protocol), it becomes possible to verify, before the interaction takes place, if the interactive behavior of a service (e.g. a BPEL process specification) respects it. This verification is known as "conformance test". Recently some attempts have been done for defining conformance tests w.r.t. a protocol but these approaches fail in capturing the very nature of interoperability, turning out to be too restrictive. In this work we give a representation of protocol, based on message exchange and on finite state automata, and we focus on those properties that are essential to the verification the interoperability of a set of services. In particular, we define a conformance test that can guarantee, a priori, the interoperability of a set of services by verifying properties of the single service against the protocol. This is particularly relevant in open environments, where services are identified and composed on demand and dynamically, and the system as a whole cannot be analyzed.

1 Introduction

In this work we face the problem of verifying the interoperability of a set of peers by exploiting an abstract description of the desired interaction. On a hand, we will have an interaction protocol (possibly expressed by a choreography), capturing the global interaction of a desired system of services; on the other, we will have a set of service implementations which should be used to assemble the system. The protocol is a specification of the desired interaction, as thus, it might be used for defining several systems of services [3]. In particular, it contains a characterization of the various *roles* played by the services [6]. In our view, a role specification is not the exact specification of a process of interest, rather it identifies a *set of possible processes*, all those whose evolutions respect the dictates given by the role. In an open environment, the introduction of a new peer in an execution context will be determined provided that it satisfies the protocol that characterizes such an execution context; as long as the new entity satisfies the rules, the interoperability with the other components of the system is guaranteed.

The computational model to which web services are inspired is that of *distributed objects* [14]. An object cannot refuse to execute a method which is invoked on it and that is contained in its public interface, in the very same way as a service cannot refuse to execute over an invocation that respects its public interface. This, however, is not the only possible model of execution. In multi-agent systems, for instance, an agent sending a request message to another agent cannot be certain that it will ever be answered, unless the interaction is ruled by a protocol. The protocol plays, in a way, the role of the public interface: an agent conforming to a protocol must necessarily answer and must be able to handle messages sent by other agents in the context of the protocol itself. The difference between the case of objects and the case of protocols is that the protocol also defines an "execution context" in which using messages. Therefore, the set of messages that it is possible to use varies depending on the point at which the execution has arrived. In a way, the protocol is a *dynamic interface* that defines messages in the context of the occurring interaction, thus ruling this interaction. On the other hand, the user of an object is not obliged to use all of the methods offered in the public interface and it can implement more methods. The same holds when protocols are used to norm the interaction. Generally speaking, only part of the protocol will be used in an entity's interaction with another and they can understand more messages than the one forseen by the protocol. Moreover, we will assume that the initiative is taken from the entity that plays as a sender, which will commit to sending a specific message out of its set of alternatives. The receiver will simply execute the reception of the message. Of course, the senders should send a message that its counterpart can understand. For all these reasons, performing the conformance test is analogous to verifying at compilation time (that is, a priori) if a class implements an interface in a correct way and to execute a static typechecking.

Sticking to a specification, on the other hand, does not mean that the service must do *all* that the role specification defines; indeed, a role specification is just a

formal definition of what is lawful to say or to expect at any given moment of the interaction. Taking this observation into account we need to define some means for verifying that a single service implementation comforms to the specification of the role in the protocol that it means to play [13]. The idea is that if a service passes the conformance test it will be able to interact with a set of other services, equally proved individually conformant to the other roles in the protocol, in a way that respects the rules defined in the protocol itself.

A typical approach to the verification that a service implementation respects a role definition is to verify whether the execution traces of the service belong to the protocol [1, 11, 7]. This test, however, does not consider processes with different branching structures. Another approach, that instead takes this case into account, is to apply bisimulation and say that the implementation is conformant if it is bisimilar to its role or, more generally, that the composition of a set of policies is bisimilar to the composition of a set of roles [9, 18]. Bisimulation [16], however, does not take into account the fact that the implementor's decisions of cutting some interaction path not necessarily compromise the interaction. Many services that respect the intuitions given above will not be bisimilar to the specification. Nevertheless, it would be very restrictive to say that they are not conformant (see Section 3.1). Thus, in order to perform the conformance test we need a softer test, a test that accepts all the processes contained in a space defined by the role. In this work we provide such a test (Section 3). This proposal differs from previous work that we have done on conformance [7, 8] in various aspects. First of all, we can now tackle protocols that contain an arbitrary (though finite) number of roles. Second, we account also for the case of policies and roles which produce the same interactions but have different branching structures. This case could not be handled in the previous framework due to the fact that we based it exclusively on a trace semantics.

2 Protocols, policies, and conversations

A *conversation policy* is a program that defines the communicative behavior of an interactive entity, e.g. a service, implemented in some programming language [3]. A conversation *protocol* specifies the desired communicative behavior of a set of interactive entities. More specifically, a conversation protocol specifies the sequences of messages (also called speech acts) that can possibly be exchanged by the involved parties, and that we consider as lawful.

In languages that account for communication, speech acts often have the form $m(a_s, a_r, l)$, where m is the kind of message, or performative, a_s (sender) and a_r (receiver) are two interactive entities and l is the message content. In the following analysis it is important to distinguish the incoming messages from the outgoing messages w.r.t a role of a protocol or a policy. We will write $\mathsf{m}?$ (*incoming message*) and $\mathsf{m}!$ (*outgoing message*) when the receiver or the utterer and the content of the message is clear from the context or they are not relevant. So, for instance, $m(a_s, a_r, l)$ is written as $\mathsf{m}?$ from the point of view of a_r, and

m! from the point of view of the sender. By the term *conversation* we will, then, denote a sequence of speech acts that is a dialogue of a set of parties.

Both a protocol and a policy can be seen as sets of conversations. In the case of the protocol, it is intuitive that it will be the set of all the possible conversations allowed by its specification among the partners. In the case of the single policy, it will be the set of the possible conversations that the entity can carry on according to its its implementing program. Although at execution time, depending on the interlocutor and on the circumstances, only one conversation at a time will actually be expressed, in order to verify conformance *a priori* we need to consider them all as a set. It is important to remark before proceeding that other proposal, e.g. [2], focus on a different kind of conformance: *run-time* conformance, in which only the ongoing conversation is checked against a protocol.

Let us then introduce a formal representation of policies and protocols. We will use *finite state automata* (FSA). This choice, though simple, is the same used by the well-known verification system SPIN [15], whose notation we adopt. FSA will be used for representing individual processes that exchange messages with other processes. Therefore, FSA will be used both for representing the *roles* of a protocol, i.e. the abstract descriptions of the interacting parties, as well as for representing the policies of specific entities involved in the interaction. In this work we do not consider the translation process necessary to turn a protocol (e.g. a WS-CDL choreography) or an entity's policy (e.g. a BPEL process) in a FSA; our focus is, in fact, conformance and interoperability. It is possible to find in the literature some works that do this kind of translations. An example is [11].

Definition 1 (Finite State Automaton). *A finite state automaton is a tuple* (S, s_0, L, T, F)*, where* S *is a finite set of states,* $s_0 \in S$ *is a distinguished initial state,* L *is a finite set of labels,* $T \subseteq (S \times L \times S)$ *is a set of transitions,* $F \in S$ *is a set of final states.*

Similarly to [15] we will denote by the "dot" notation the components of a FSA, for example we use $A.s$ to denote the state s that belongs to the automaton A. The definition of run is taken from [15].

Definition 2 (Runs and strings). *A run* σ *of a FSA* (S, s_0, L, T, F) *is an ordered, possibly infinite, set of transitions (a sequence)* $(s_0, l_0, s_1), (s_1, l_1, s_2), (s_2, l_2, s_3), \ldots$ *such that* $\forall i \geq 0$*,* $(s_i, l_i, s_{i+1}) \in T$*, while the sequence* $l_0 l_1 \ldots$ *is the corresponding string* $\overline{\sigma}$*.*

Definition 3 (Acceptance). *An accepting run of a finite state automaton* (S, s_0, L, T, F) *is a finite run* σ *in which the final transition* (s_{n-1}, l_{n-1}, s_n) *has the property that* $s_n \in F$*. The corresponding string* $\overline{\sigma}$ *is an accepted string.*

Given a FSA A, we say that a state $A.s_1 \in A.S$ is *alive* if there exists a finite run $(s_1, l_1, s_2), \ldots, (s_{n-1}, l_{n-1}, s_n)$ and $s_n \in A.F$. Moreover, we will write $A_1 \subseteq A_2$ iff every string of A_1 is also a string of A_2.

In order to represent compositions of policies or of individual protocol roles we need to introduce the notions of *free* and of *synchronous* product. These

definitions are an adaptation to the problem that we are tackling of the analogous ones presented in [4] for Finite Transition Systems.

Definition 4 (Free product). *Let A_i, $i = 1, \ldots, n$, be n FSA's. The* free product $A_1 \times \cdots \times A_n$ *is the FSA $A = (S, s_0, L, T, F)$ defined by:*

- *S is the set $A_1.S \times \cdots \times A_n.S$;*
- *s_0 is the tuple $(A_1.s_0, \ldots, A_n.s_0)$;*
- *L is the set $A_1.L \times \cdots \times A_n.L$;*
- *T is the set of tuples $((A_1.s_1, \ldots, A_n.s_n), (l_1, \ldots, l_n), (A_1.s'_1, \ldots, A_n.s'_n))$ such that $(A_i.s_i, l_i, A_i.s'_i) \in A_i.T$, for $i = 1, \ldots, n$; and*
- *F is the set of tuples $(A_1.s_1, \ldots, A_n.s_n) \in A.S$ such that $s_i \in A_i.F$, for $i = 1, \ldots, n$.*

We will assume, from now on, that every FSA A has an *empty* transition (s, ε, s) for every state $s \in A.S$. When the finite set of labels L used in a FSA is a set of *speech acts*, strings will represent *conversations*.

Definition 5 (Synchronous product). *Let A_i, $i = 1, \ldots, n$, be n FSA's. The* synchronous product *of the A_i's, written $A_1 \otimes \cdots \otimes A_n$, is the FSA obtained as the free product of the A_i's containing only the transitions $((A_1.s_1, \ldots, A_n.s_n), (l_1, \ldots, l_n), (A_1.s'_1, \ldots, A_n.s'_n))$ such that there exist i and j, $1 \leq i \neq j \leq n$, $l_i = \mathsf{m}!$, $l_j = \mathsf{m}?$, and for any k not equal to i and j, $l_k = \varepsilon$.*

The synchronous product allows a system that exchanges messages to be represented. It is worth noting that a synchronous product does not imply that messages will be exchanged in a synchronous way; it simply represents a message exchange without any assumption on how the exchange is carried on.

In order to represent a protocol, we use the synchronous product of the set of such FSA's associated with each role (where each FSA represents the communicative behavior of the role). Moreover, we will assume that the automata that compound the synchronous product have some "good properties", which meet the commonly shared intuitions behind protocols. In particular, we assume that for the set of such automata the following properties hold:

1. *any message that can possibly be sent, at any point of the execution, will be handled by one of its interlocutor*;
2. *whatever point of conversation has been reached, there is a way to bring it to an end.*

An arbitrary synchronous product of n FSA's might not meet these requirements, which can, however, be verified by using automated systems, like SPIN [15].

Note that protocol specification languages, like UML sequence (activity) diagrams and automata [17], naturally follow these requirements: an arrow starts from the lifeline of a role, ending into the lifeline of another role, and thus corresponds to an outgoing or to an incoming message depending on the point of view. Making an analogy with the computational model of distributed objects, one could say that the only messages that are sent are those which can be understood. Moreover, usually protocols contain finite conversations.

We will say that a conversation is *legal w.r.t. a protocol* if it respects the specifications given by the protocol, i.e. if it is an accepted string of the protocol.

3 Interoperability and conformance test

We are now in position to explain, with the help of a few simple examples, the intuition behind the terms "conformance" and "interoperability", that we will, then, formalize. By *interoperability* we mean the capability of a set of entities of actually producing a conversation when interacting with one another [5]. Interoperability is a *desired property* of a system of interactive entities and its verification is fundamental in order to understand whether the system works. Such a test passes through the analysis of all the entities involved in the interaction. In an open system, however, it is quite unlikely to have a global view of the system either because it is not possible to read part of the necessary information (e.g. some services do not publish their behavior) or because the interactive entities are identified at different moments, when necessary. Protocols are adopted to solve such problems, in fact, having an interaction schema allows the *distribution of the tests in time*, by checking a single entity at a time against the role that it should play. The protocol, by its own nature guarantees the interoperability of the roles that are part of it. One might argue why we do not simply verify the system obtained by substituting the policy instead of its role within the protocol and, then, check whether any message that can be sent will be handled by some of the interlocutor roles, bringing to an end the conversations. Actually, this solution presents some flaws, as the following counter-example proves. Let us consider a protocol with three roles: A_1 sends m_1 to A_2, A_2 waits for m_1 and then it waits for m_2, and A_3 sends m_2 to A_2. Let us know substitute to role A_2 the policy which, first, waits for m_2 and then it waits for m_1. The three partners will perfectly interoperate and successfully conclude their conversations but the conversation that is produced is not legal w.r.t. the protocol. In protocol-based systems, the proof of the interoperability of an entity with others, obtained by checking the communicative behavior of the entity against the rules of the system (i.e. against an *interaction protocol* itself), is known as *conformance test*. Intuitively, this test must guarantee the following *definition of interoperability*.

Definition 6 (Interoperability w.r.t. an interaction protocol). *Interoperability w.r.t. an interaction protocol is the capability of a set of entities of producing a conversation that is legal w.r.t. the protocol.*

Let us now consider a given service that should play a role in a protocol. In order to include it in the interaction we need to understand if it will be able to interact with the possible players of the other roles. If we assume that the other players are conformant to their respective roles, we can represent them by the roles themselves. Roles, by the definition of protocol, are interoperable. Therefore, in order to prove the interoperability of our service, it will be sufficient to prove for it the "good properties" of its role. First of all, we should prove that its policy does not send messages that the others cannot understand, which means that it will not send messages that are not accounted for by the role. Moreover, we should prove that it can tackle every incoming message that the other roles might send to it, which means that it must be able to handle all the incoming

messages handled by the role. Another important property is that whatever point of conversation has been reached, there is a way to bring it to an end. In practice, if a role can bring to an end a conversation in which it has been engaged, so must do the service. To summarize, in order to check a service interoperability it will be sufficient to check its *conformance w.r.t. the desired role* and this check will guarantee that the service will be able to interact with services equally, and separately, proved conformant to the other roles. This, nevertheless, does not mean that the policy of the service must be a precise "copy" of the role.

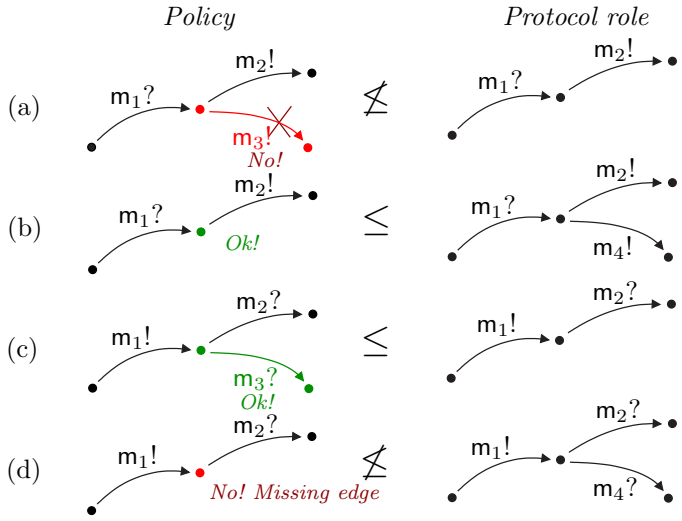

Fig. 1. A set of cases that exemplifies our expectations about a conformant policy: cases (b) and (c) do not compromise interoperability, hence they should pass the conformance test; cases (a) and (d) instead should not pass the conformance test.

3.1 Expectations for interoperability

Let us now discuss some typical cases in which a policy and a role specification that differ in various ways are compared in order to decide if the policy conforms to the role so as to guarantee its interoperability with its future interlocutors that will play the other roles in the protocol. With reference to Figure 1, let us begin with considering the case reported in row (a): here, the service can possibly utter a message m_3 that is not foreseen by the role specification. Trivially, this policy is *not conformant* to the protocol because the service might send a message that cannot be handled by any interlocutor that conforms to the protocol. The symmetric case in which the policy accounts for less outgoing messages than the role specification (Figure 1, row (b)) is, instead, legal. The reason is

that at any point of its conversations the entity will anyway always utter only messages that the entities playing the other roles will surely understand. Hence, interoperability is preserved. The restriction of the set of possible alternatives (w.r.t. the protocol) depends on the implementor's own criteria.

Let us now consider the case reported in Figure 1, row (c). Here, the service policy accounts for two conversations in which, after uttering a message m_1, the entity expects one of the two messages m_2 or m_3. Let us also suppose that the protocol specification only allows the first conversation, i.e. that the only possible incoming message is m_2. When the entity will interact with another that is conformant to the protocol, the message m_3 will never be received because the other entity will never utter it. So, in this case, we would like the a priori conformance test to accept the policy as *conformant* to the specification.

Talking about incoming messages, let us now consider the symmetric case (Figure 1, row (d)), in which the *protocol specification* states that after an outgoing message m_1, an answer m_2 or m_4 will be received, while the policy accounts only for the incoming message m_2. In this case, the expectation is that the policy is *not conformant* because there is a possible incoming message (the one with answer m_4) that can be enacted by the *interlocutor*, which, however, cannot be handled by the policy. This compromises interoperability.

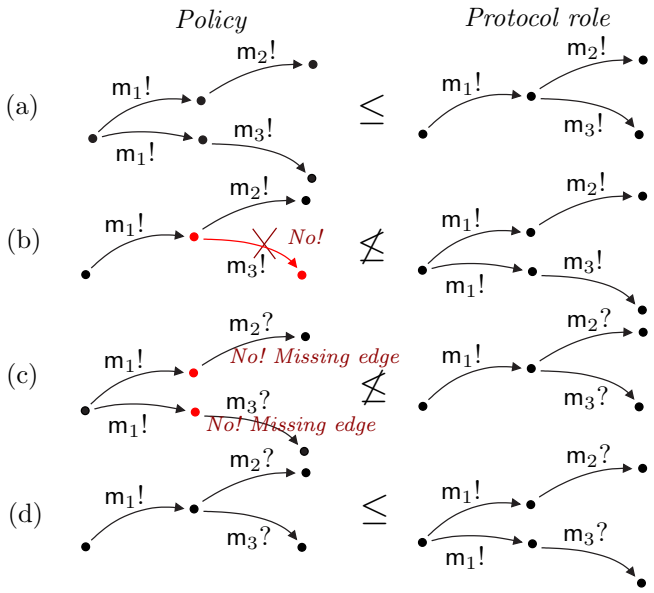

Fig. 2. A set of cases that exemplifies our expectations about a conformant policy: differently than in Figure 1, for every row, the policy and the role produce the same conversations but the structure of their implementations differ.

To summarize, at every point of a conversation, we expect that a conformant policy never utters speech acts that are not expected, according to the protocol, and we also expect it to be able to handle any message that can possibly be received, once again according to the protocol. However, the policy is not obliged to foresee (at every point of conversation) an outgoing message for every alternative included in the protocol but it must foresee at least one of them if this is necessary to proceed with the conversation. Trivially, in the example of row (b), a policy containing only the conversation $\mathsf{m}_1?$ (not followed either by $\mathsf{m}_2!$ or by $\mathsf{m}_4!$) would not be *conformant*.

Let us now consider a completely different set of situations, in which the "structure" of the policy implemented and the structure of the role specification are taken into account. These situations are taken from the literature on communicating processes [12]. Figure 2 reports a set of cases in which the role description and the policy allow the *same conversations* but their structure differs: in rows (a) and (c) the policy decides which message to send (receive, respectively) after m_1 from the very beginning, while in the protocol this decision is taken after m_1 is sent. In row (b) and (d) the situation is inverted.

The case of row (a) does not compromise conformance in the same way as the case reported at row (b) of Figure 1 does not: after a non-deterministic choice the set of alternative outgoing messages is restricted but in both cases only legal messages that can be handled by the interlocutor will be sent. The analogous case reported in row (c), concerning incoming messages, instead, compromises the conformance. In fact, after the non-deterministic step the policy might receive a message that it cannot handle, similarly to row (d) of Figure 1.

The case of row (b), Figure 2, compromises the conformance because after the non-deterministic choice the role specification allows a single outgoing message with no alternatives. The policy, instead, might utter one out of two alternative messages (similarly to row (a) of Figure 1). Finally, the case of row (d) does not compromise the conformance, following what reported in Figure 1, row (c).

3.2 Conformance and interoperability

In this section we define a test, for checking conformance, that is derived from the observations above. A first consideration is that a conformance test *is not an inclusion test* w.r.t. the set of possible conversations that are produced. In fact, for instance, in row (d) of Figure 1 the policy produces a subset of the conversations produced by the role specification but interoperability is not guaranteed. Instead, if we consider row (c) in the same figure, the set of conversation traces, produced by the policy, is a superset of the one produced by the protocol; despite this, interoperability is guaranteed. A second consideration is that a conformance test is not a bisimulation test w.r.t. the role specification. Actually, the (bi)simulation-based test defined in concurrency theory [16] is too strict, and it imposes constraints, that would exclude policies which instead would be able to interoperate, within the context given by the protocol specification. In particular, all the cases reported in Figure 2 would not be considered as conformant because they are all pairs of processes with different branching structures.

Despite this, we would like our test to recognize cases (a) and (d) as conformant because they do not compromise interoperability.

The solution that we propose is inspired by (bi)simulation, but it distinguishes the ways in which incoming and outgoing messages are handled, when a policy is compared to a role [1]. In the following, we will use "$A_1 \leq A_2$" to denote the fact that A_1 conforms to A_2. This choice might seem contradictory after the previous discussion, in fact, in general $A_1 \leq A_2$ does not entail $A_1 \subseteq A_2$. However, with symbol "$\leq$" we capture the fact that A_1 will actually produce a subset of the conversations forseen by the role, *when interacting with entities that play the other roles in the protocol* (see Propositions 1 and 2). This is what we expect from a conformant policy and from our definition of interoperability.

Definition 7 (Conformant simulation). *Given two FSA's A_1 and A_2 we say that A_1 is a* conformant simulation *of A_2, written $A_1 \leq A_2$ iff there is a binary relation $\mathcal{R}$ between A_1 and A_2 such that*

- $A_1.s_0 \mathcal{R} A_2.s_0$;
- *for every outgoing message* $\mathsf{m}! \in A_1.L$ *and for every state* $s_i \in A_1.S$, *for every* $s_j \in A_2.S$ *such that* $s_i \mathcal{R} s_j$ *and* $(s_i, \mathsf{m}!, s_{i+1}) \in A_1.T$, *then there is a state* $s_{j+1} \in A_2.S$ *such that* $(s_j, \mathsf{m}!, s_{j+1}) \in A_2.T$ *and* $s_{i+1} \mathcal{R} s_{j+1}$;
- *for every incoming message* $\mathsf{m}? \in A_2.L$ *and for every state* $s_j \in A_2.S$, *for every* $s_i \in A_1.S$ *such that* $s_i \mathcal{R} s_j$ *and* $(s_j, \mathsf{m}?, s_{j+1}) \in A_2.T$, *then there is a state* $s_{i+1} \in A_1.S$ *such that* $(s_i, \mathsf{m}?, s_{i+1}) \in A_1.T$ *and* $s_{i+1} \mathcal{R} s_{j+1}$.

Particularly relevant is the case in which A_2 is a role in a protocol and A_1 is a policy implementation. Notice that, in this case, conformance is defined only w.r.t. the role that the single policy implements, *independently* from the rest of the protocol. As anticipated above, Definition 7 does not imply the fact that "$A_1 \leq A_2$ entails $A_1 \subseteq A_2$". Instead, the following proposition holds.

Proposition 1. *Let $A_1 \otimes \cdots \otimes A_i \otimes \cdots \otimes A_n$ be a protocol, and A'_i a policy such that $A'_i \leq A_i$, then $A_1 \otimes \cdots \otimes A'_i \otimes \cdots \otimes A_n \subseteq A_1 \otimes \cdots \otimes A_i \otimes \cdots \otimes A_n$.*

This proposition catches the intuition that a conformant policy is able to produce a subset of the legal conversations defined by the protocol but only when it is executed in the context given by the protocol.

The above proposition can be generalized in the following way. Here we consider a set of policies that have been individually proved as being conformant simulations of the various roles in a protocol. The property states that the dialogues that such policies can produce will be legal *w.r.t. the protocol.*

Proposition 2. *Let $A_1 \otimes \cdots \otimes A_n$ be a protocol and let $A'_1, \ldots, A'_n$ be n policies such that $A'_i \leq A_i$, for $i = 1, \ldots, n$, then $A'_1 \otimes \cdots \otimes A'_n \subseteq A_1 \otimes \cdots \otimes A_n$*

In order to prove interoperability we need to prove that our policies will actually produce a conversation when interacting, while so far we have only proved that if a conversation will be generated, it will be legal. By assumption, in a protocol

[1] All proofs are omitted for lack of space, they will be supplied on demand.

it is always possible to conclude a conversation whatever the point at which the interaction arrived. We expect a similar property to hold also for a set of policies that have been proved conformant to the roles of a protocol. The relation $\leq$ is too weak, so we need to introduce the notion of *complete conformant simulation.*

Definition 8 (Complete conformant simulation). *Given two FSA's A_1 and A_2 we say that A_1 is a* complete conformant simulation *of A_2, written $A_1 \trianglelefteq A_2$, iff there is a A_1 is a conformant simulation of A_2 under a binary relation $\mathcal{R}$ and*

- *for all $s_i \in A_1.F$ such that $s_i \mathcal{R} s_j$, then $s_j \in A_2.F$;*
- *for all $s_j \in A_2.S$ such that s_j is alive and $s_i \mathcal{R} s_j$, $s_i \in A_1.S$, then s_i is alive.*

Now, we are in the position to give the following fundamental result.

Theorem 1 (Interoperability). *Let $A_1 \otimes \cdots \otimes A_n$ be a protocol and let $A'_1, \ldots, A'_n$ be n policies such that $A'_i \trianglelefteq A_i$, for $i = 1, \ldots, n$. For any common string $\overline{\sigma'}$ of $A'_1 \otimes \cdots \otimes A'_n$ and $A_1 \otimes \cdots \otimes A_n$ there is a run $\sigma'\sigma''$ such that $\overline{\sigma'\sigma''}$ is an accepted string of $A'_1 \otimes \cdots \otimes A'_n$.*

Intuitively, whenever two policies, that have independently been proved conformant to the two roles of a protocol, start an interaction, thanks to Proposition 2, they will be able to conclude their interaction producing a legal accepted run. Therefore, Theorem 1 implies Definition 6 (interoperability).

4 Conclusions and related works

In this work we have given a definition of conformance and of interoperability that is suitable to application in open environments, like the web. Protocols have been formalized in the simplest possible way (by means of FSA) to capture the essence of interoperability and to define a fine-grain conformance test.

The issue of conformance is widely studied in the literature in different research fields, like multi-agent systems (MAS) and service-oriented computing (SOA). In particular, in the area of MAS, in [7, 5] we have proposed two preliminary versions of the current proposal, the former, based on a trace semantics, consisting in an inclusione test, the latter, disregarding the case of different branching structures. The second technique was also adapted to web services [8]. Both works were limited to protocols with only two roles while, by means of the framework presented in this paper we can deal with protocols with an arbitrary finite number of roles. Inspired to this work the proposal in [1]: here an abductive framework is used to verify the conformance of services to a choreography with any number of roles. The limit of this work is that it does not consider the cases in which policies and roles have different branching structures. The first proposal of a formal notion of conformance in a declarative setting is due to Endriss *et al.* [10], the authors, however, do not prove any relation between their definitions of conformance and interoperability. Moreover, they consider protocols in which two partners strictly alternate in uttering messages.

In the SOA research field, conformance has been discussed by Foster *et al.* [11], who defined a system that translates choreographies and orchestrations in

labeled transition systems so that it becomes possible to apply model checking techniques and verify properties of theirs. In particular, the system can check if a service composition complies with the rules of a choreography by equivalent interaction traces. Violations are highlighted back to the engineer. Once again, as we discussed, basing on traces can be too much restrictive. In [9], instead, "conformability bisimulation" is defined, a variant of the notion of bisimulation. This is the only work that we have found in which different branching structures are considered but, unfortunately, the test is too strong. In fact, with reference to Figure 1, it excludes the cases (b) and (c), and it also excludes cases (a) and (d) from Figure 2, which do not compromise interoperability. A recent proposal, in this same line, is [18], which suffers of the same limitations.

Acknowledgements. This research has partially been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. http://rewerse.net), and it has also been supported by MIUR PRIN 2005 "Specification and verification of agent interaction protocols" national project.

References

1. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and M. Montali. An abductive framework for a-priori verification of web services. In *Principles and Practice of Declarative Programming, PPDP'06)*. ACM Press, 2006.
2. M. Alberti, D. Daolio, P. Torroni, M. Gavanelli, E. Lamma, and P. Mello. Specification and verification of agent interaction protocols in a logic-based system. In *ACM SAC 2004*, pages 72–78. ACM, 2004.
3. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services*. Springer, 2004.
4. André Arnold. *Finite Transition Systems*. Pearson Education, 1994.
5. M. Baldoni, C. Baroglio, A. Martelli, and Patti. Verification of protocol conformance and agent interoperability. In *Post-Proc. of CLIMA VI*, volume 3900 of *LNCS State-of-the-Art Survey*, pages 265–283. Springer, 2006.
6. M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about interaction protocols for customizing web service selection and composition. *J. of Logic and Alg. Progr., special issue on Web Services and Formal Methods*, 2006. To appear.
7. M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Verifying protocol conformance for logic-based communicating agents. In *Proc. of CLIMA V*, number 3487 in LNCS, pages 192–212. Springer, 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 *Proc. of WS-FM 2005*, volume 3670 of *LNCS*, pages 257–271. Springer, September, 2005.
9. N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and orchestration: a synergic approach for system design. In *Proc. of 4th International Conference on Service Oriented Computing (ICSOC 2005)*, 2005.
10. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Logic-based agent communication protocols. In *Advances in agent communication languages*, volume 2922 of *LNAI*, pages 91–107. Springer-Verlag, 2004. invited contribution.

11. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based analysis of obligations in web service choreography. In *Proc. of IEEE International Conference on Internet&Web Applications and Services 2006*, 2006.
12. R.J. van Glabbeek. Bisimulation. Encyclopedia of Distributed Computing (J.E. Urban & P. Dasgupta, eds.), Kluwer, 2000. Available at `http://Boole.stanford.edu/pub/DVI/bis.dvi.gzz`.
13. F. Guerin and J. Pitt. Verification and Compliance Testing. In H.P. Huget, editor, *Communication in Multiagent Systems*, volume 2650 of *LNAI*, pages 98–112. Springer, 2003.
14. B. Heckel. *Thinking Java.* Prentice Hall, 2005.
15. Gerard J. Holzmann. *The SPIN Model Checker : Primer and Reference Manual.* Addison-Wesley Professional, 2003.
16. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.
17. OMG. Unified modeling language: Superstructure, 2005.
18. X. Zhao, H. Yang, and Z. Qui. Towards the formal model and verification of web service choreography description language. In *Proc. of WS-FM 2006.* 2006.