Verification of protocol conformance and agent interoperability*

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

Abstract. In open multi-agent systems agent interaction is usually ruled by public protocols defining the rules the agents should respect in message exchanging. The respect of such rules guarantees interoperability. Given two agents that agree on using a certain protocol for their interaction, a crucial issue (known as "a priori conformance test") is verifying if their interaction policies, i.e. the programs that encode their communicative behavior, will actually produce interactions which are conformant to the agreed protocol. An issue that is not always made clear in the existing proposals for conformance tests is whether the test preserves agents' capability of interacting, besides certifying the legality of their possible conversations. This work proposes an approach to the verification of a priori conformance, of an agent's conversation policy to a protocol, which is based on the theory of formal languages. The conformance test is based on the acceptance of both the policy and the protocol by a special finite state automaton and it guarantees the interoperability of agents that are individually proved conformant. Many protocols used in multi-agent systems can be expressed as finite state automata, so this approach can be applied to a wide variety of cases with the proviso that both the protocol specification and the protocol implementation can be translated into finite state automata. In this sense the approach is general. Easy applicability to the case when a logic-based language is used to implement the policies is shown by means of a concrete example, in which the language DyLOG, based on computational logic, is used.

1 Introduction

Multi-agent systems (MASs) often comprise heterogeneous components, that differ in the way they represent knowledge about the world and about other agents, as well as in the mechanisms used for reasoning about it. Protocols rule the agents' interaction. Therefore, they can be used to check if a given agent can, or cannot, take part into the system. In general, based on this abstraction, open

* This research is partially supported by MIUR Cofin 2003 "Logic-based development and verification of multi-agent systems" national project and by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779.

systems can be realized, in which new agents can dynamically join the system. The insertion of a new agent in an execution context is determined according to some form of reasoning about its behaviour: it will be added provided that it satisfies the body of the rules within the system, intended as a society.

In a protocol-ruled system of this kind, it is, however, not necessary to check the interoperability (i.e. the capability of *actually* producing a conversation) of the newly entered agent with the other agents in the system if, as long as the rules are satisfied, the property is guaranteed. The problem which amounts to verify if a given *implementation* (an agent interaction policy) respects a given *abstract protocol definition* is known as *conformance* testing. A conformance test can, then, be considered as a tool that, by verifying that agents respect a protocol, *should certify* their interoperability. In this perspective, we expect that two agents which conform to a protocol will *produce a conversation*, that is legal (i.e. correct w.r.t. the protocol), when interacting with one another.

The design and implementation of interaction protocols are crucial steps in the development of a MAS [18, 19]. Following [17], two tests must be executed in the process of interaction protocol engineering. One is the already mentioned conformance test, the other is the *validation* test, which verifies the consistency of an *abstract protocol definition* w.r.t. the *requirements*, derived from the analysis phase, that it should embody. In the literature validation has often been tackled by means of model checking techniques [7, 6, 22], and two kinds of conformance verifications have been studied: *a priori* conformance verification, and *run-time* conformance verification (or compliance) [9, 10, 15]. If we call a *conversation* a specific interaction between two agents, consisting only of communicative acts, the first kind of conformance is a property of the *implementation as a whole* –intuitively it checks if an agent will never produce conversations that violate the abstract interaction protocol specification– while the latter is a property of the *on-going conversation*, aimed at verifying if *that* conversation is legal.

In this work we focus on a priori conformance verification, defining a conformance test, based on the acceptance, of both the agent's policy and the public protocol, by a special finite state automaton. Many protocols used in multi-agent systems can be expressed as finite state automata, so this approach can be applied to a wide variety of cases with the proviso that both the protocol specification and the protocol implementation (policy) can be translated into finite state automata. In this sense the approach is general.

The test that we defined guarantees agent interoperability (see Theorem 1 in Section 3). The communicative behavior of an agent (the decision of which specific action to take) normally relies also on information like the private state of the agent and the social commitments. We will see that our notion of conformance is *orthogonal* to the framework in which one reasons about communication (mentalistic or social [15]). So, our approach works on sets of conversations without caring about the information used to obtain them.

The application of our approach is particularly easy in case a logic-based declarative language is used to implement the policies. In logic languages indeed policies are usually expressed by Prolog-like rules, which can be easily converted

in a formal language representation. In Section 4 we show this by means of a concrete example where the language DyLOG [5], based on computational logic, is used for implementing the agents' policies. On the side of the protocol specification languages, currently there is a great interest in using informal, graphical languages (e.g. UML-based) for specifying protocols and in the translation of such languages in formal languages [8, 11]. By this translation it is, in fact, possible to prove properties that the original representation does not allow. In this context, in [4] we have shown an easy algorithm for translating AUML sequence diagrams to finite state automata thus enabling the verification of conformance.

In [4] we already faced the problem of a priori conformance verification as a verification of properties of formal languages, but proposing a different approach with some limitations due to focussing on the legality issue. In fact, interpreting (as we did) the conformance test as the verification that all the conversations, allowed by an agent's policy, are also possible according to the protocol specification, does not entail interoperability. The next section is devoted to explain the expected relations among conformance and the crucial interoperability issue.

2 Conformant and interoperable agents

A *conversation policy* is a program that defines the communicative behavior of a specific agent, implemented in some programming language. A conversation *protocol* specifies the desired communicative behavior of a set of agents and it can be specified by means of many formal tools, such as (but not limited to) Petri nets, AUML sequence diagrams, automata.

In this work we face the problem of conformance verification and interpret a priori conformance as a property that relates two formal languages, the language of the conversations allowed by the conversation policy of an agent, and the language of the conversations allowed by the specification of a communication protocol. They will respectively be denoted by $L(p^{ag}_{lang})$ and $L(p_{spec})$, where *spec* is the specification language, *lang* is the language in which the policy executed by agent *ag* is written, and p is the name of the policy or of the protocol at issue. The assumption that we make throughout this paper is that the two languages are *regular sets*. This choice restricts the kinds of protocols to which our proposal can be applied, because finite state automata cannot represent concurrent operations, however, it is still significant because a wide family of protocols (and policies) of practical use can be expressed in a way that can be mapped onto such automata. Moreover, the use of regular sets ensures decidability. Another assumption is that the conversation protocol encompasses only *two agents*. The extension to a greater number of agents will be tackled as future work. Notice that when the MAS is *heterogeneous*, the agents might be implemented in *different languages*.

A conversation protocol specifies the sequences of speech acts that can possibly be exchanged by the involved agents, and that we consider as legal. In agent languages that account for communication, speech acts often have the form $m(ag_s, ag_r, l)$, where m is the performative, ag_s (sender) and ag_r (receiver)

are two agents and l is the message content. It is not restrictive to assume that speech acts have this form and to assume that conversations are sequences of speech acts of this form. Depending on the semantics of the speech acts, the conversation will take place in a framework based either on the *mentalistic* or on the *social state* approach [12, 21, 14]. The speech acts semantics, actually, does not play a part in our proposal, which fits both the approaches.

In the following analysis it is important to distinguish the incoming messages, w.r.t. a specific agent ag of the MAS, from the messages uttered by it. We respectively denote the former, where ag plays the role of the receiver, by $\mathsf{m}(\overleftarrow{ag})$, and the latter, where ag is the sender, by and $\mathsf{m}(\overrightarrow{ag})$. We will also simply write $\overleftarrow{\mathsf{m}}$ (*incoming message*) and $\overrightarrow{\mathsf{m}}$ (*outgoing message*) when the agent that receives or utters the message is clear from the context. Notice that these are just short notations, that underline the *role* of a given agent from the *individual perspective* of *that* agent. So, for instance, $m(ag_s, ag_r, l)$ is written as $\mathsf{m}(\overleftarrow{ag_r})$ from the point of view of ag_r, and $\mathsf{m}(\overrightarrow{ag_s})$ from the point of view of the sender but the three notions denote the same object.

A *conversation*, denoted by σ, is a sequence of speech acts that represents a dialogue of a set of agents. We say that a conversation is legal w.r.t. a protocol specification if it respects the specifications given by the protocol. Since $L(p_{spec})$ is the set of all the legal conversations according to p, the definition is as follows.

Definition 1 (Legal conversation). *We say that a conversation σ is legal w.r.t. a protocol specification p_{spec} when $\sigma \in L(p_{spec})$.*

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.

> *Interoperability is the capability of an agent of actually producing a conversation when interacting with another.*

Often the introduction of a new agent in an execution context is determined according to some form of reasoning about its behaviour: it will be added provided that it satisfies the body of the rules within the system, intended as a society. As long as the rules are satisfied, the property is guaranteed and it will not be necessary to verify interoperability with the single components of the system. This can be done by checking the communicative behavior of the agent against the rules of the society, i.e. against an *interaction protocol*. Such a proof is known as *conformance test*. Intuitively, this test must guarantee the following *definition of interoperability*. This work focuses on it.

> *We expect that two agents, that conform to a protocol, will produce a legal conversation, when interacting with one another.*

Let us begin with considering the following case: suppose that the communicative behavior of the agent ag is defined by a policy that accounts for two conversations $\{\mathsf{m}_1(\overrightarrow{ag})\mathsf{m}_2(\overleftarrow{ag}), \mathsf{m}_1(\overrightarrow{ag})\mathsf{m}_3(\overleftarrow{ag})\}$. This means that after uttering a message m_1, the agent 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. Is the policy conformant? According to Def. 1 the answer should be no, because the policy allows an illegal conversation. Nevertheless, when the agent will interact with another agent that is conformant to the protocol, the message m_3 will never be received because the other agent 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, in which the *protocol specification* states that after the agent ag has uttered m_1, the other agent can alternatively answer m_2 or m_4 (agent ag's policy, instead, is the same as above). In this case, the expectation is that ag's policy is *not conformant* because, according to the protocol, there is a possible legal conversation (the one with answer m_4) that can be enacted by the *interlocutor* (which is not under the control of ag), which, however, ag cannot handle. So it does not comply to the specifications.

> *As a first observation we expect the policy to be able to handle any incoming message, foreseen by the protocol, and we ignore those cases in which the policy foresees an incoming message that is not supposed to be received at that point of the conversation, according to the protocol specification.*

Let us, now, suppose that agent ag's policy can produce the following conversations $\{\mathsf{m}_1(\overleftarrow{ag})\mathsf{m}_2(\overrightarrow{ag}),\ \mathsf{m}_1(\overleftarrow{ag})\mathsf{m}_3(\overrightarrow{ag})\}$ and that the set of conversations allowed by the protocol specification is $\{\mathsf{m}_1(\overleftarrow{ag})\mathsf{m}_2(\overrightarrow{ag})\}$. Trivially, this policy is *not conformant* to the protocol because ag can send a message (m_3) that cannot be handled by any interlocutor that is conformant to the protocol.

> *The second observation is that we expect a policy to never utter a message that, according to the specification, is not supposed to be uttered at that point of the conversation.*

Instead, in the symmetric case in which the policy contains only the conversation $\{\mathsf{m}_1(\overleftarrow{ag})\mathsf{m}_2(\overrightarrow{ag})\}$ while the protocol states that ag can answer to m_1 alternatively by uttering m_2 or m_3, *conformance holds*. The reason is that at any point of its conversations the agent will always utter legal messages. The restriction of the set of possible alternatives (w.r.t. the protocol) depends on the agent implementor's own criteria. However, the agent must foresee *at least one* of such alternatives otherwise the conversation will be interrupted. Trivially, the case in which the policy contains only the conversation $\{\mathsf{m}_1(\overleftarrow{ag})\}$ is *not conformant*.

> *The third observation is that we expect that a policy always allows the agent to utter one of the messages foreseen by the protocol at every point of the possible conversations. However, it is not necessary that a policy envisions all the possible alternatives.*

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). Incoming and outgoing messages are, therefore, *not handled in the same way.*

These expectations are motivated by the desire to define a minimal set of conditions which guarantee the construction of a conformance test that guarantess the *interoperability* of agents. Let us recall that one of the aims (often implicit) of conformance is, indeed, interoperability, although sometimes research on this topic restricts its focus to the legality issues. We claim –and we will show– that two agents that respect this minimal set of conditions (w.r.t. an agreed protocol) will *actually* be able to interact, respecting at the same time the protocol. The relevant point is that this certification is a property that can be checked on the single agents, rather than on the agent society. This is interesting in application domains (e.g. web services) with a highly dynamic nature, in which agents are searched for and composed at the moment in which specific needs arise.

3 Conformance test

In order to decide if a policy is conformant to a protocol specification, it is not sufficient to perform an inclusion test; instead, as we have intuitively shown by means of the above examples, it is necessary to prove mutual properties of both $L(p^{ag}_{lang})$ and $L(p_{spec})$. The method that we propose, for proving such properties, consists in verifying that both languages are recognized by a special finite state automaton, whose construction we are now going to explain. Such an automaton is based on the automaton that accepts the *intersection* of the two languages. All the conversations that belong to the intersection are certainly legal. This, however, is not sufficient, because there are further conditions to consider, for instance there are conversations that we mean to allow but that do not belong to the intersection. In other words, the "intersection automaton" does not capture all the expectations reported in Section 2. We will extend this automaton in such a way that it will accept the converations in which the agent expects messages that are not foreseen by the specification as well as those which include outgoing messages that are not envisioned by the policy. On the other hand, the automaton will not accept conversations that include incoming messages that are not expected by the policy nor it will accept conversations that include outgoing messages, that are not envisioned by the protocol (see Fig. 1).

3.1 The automaton M_{conf}

If $L(p^{ag}_{lang})$ and $L(p_{spec})$ are regular, they are accepted by two (deterministic) *finite automata*, that we respectively denote by $M(p^{ag}_{lang})$ and $M(p_{spec})$, that we can assume as having the *same alphabet* (see [16]). An automaton is a five-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite input alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and δ is a transition

function mapping $Q \times \Sigma$ to Q. In a finite automaton we can always classify states in two categories: *alive states*, that lie on a path from the initial state to a final state, and *dead states*, the other ones. Intuitively, alive states accept the language of the prefixes of the strings accepted by the automaton.

For reasons that will be made clear shortly, we request the two automata to show the following property: the edges that lead to the same state must *all* be labelled either by incoming messages or by outgoing messages w.r.t. ag.

Definition 2 (IO-automaton). *Given an automaton $M = (Q, \Sigma, \delta, q_0, F)$, let $E_q = \{m \mid \delta(p, m) = q\}$ for $q \in Q$. We say that M is an IO-automaton iff for every $q \in Q$, E_q alternatively consists only of incoming or only of outgoing messages w.r.t. an agent ag.*

Notice that an automaton that does not show this property can always be transformed so as to satisfy it, in linear time w.r.t. the number of states, by splitting those states that do not satisfy the property. We will denote a state q that is reached only by incoming messages by the notation $\overleftarrow{q}$ (we will call it an I-state), and a state q that is reached only by outgoing messages by $\overrightarrow{q}$ (an O-state).

Finally, let us denote by $M^{\times}(p^{ag}_{lang}, p_{spec})$ the deterministic finite automaton that accepts the language $L(p^{ag}_{lang}) \cap L(p_{spec})$. It is defined as follows. Let $M(p^{ag}_{lang})$ be the automaton $(Q^P, \Sigma, \delta^P, q^P_0, F^P)$ and $M(p_{spec})$ the automaton $(Q^S, \Sigma, \delta^S, q^S_0, F^S)$:

$$M^{\times}(p^{ag}_{lang}, p_{spec}) = (Q^P \times Q^S, \Sigma, \delta, [q^P_0, q^S_0], F^P \times F^S)$$

where for all q^P in Q^P, q^S in Q^S, and m in Σ, $\delta([q^P, q^S], m) = [\delta^P(q^P, m), \delta^S(q^S, m)]$. We will briefly denote this automaton by $M^{\times}$.

Notice that all the *conversations* that are accepted by $M^{\times}$ are surely *conformant* (Def. 1). For the so built automaton, it is easy to prove the following property.

Proposition 1. *$M^{\times}(p^{ag}_{lang}, p_{spec})$ is an IO-automaton if $M(p^{ag}_{lang})$ and $M(p_{spec})$ are two IO-automata.*

Definition 3 (Automaton M_{conf}). *The finite state automaton $M_{conf}(p^{ag}_{lang}, p_{spec})$ is built by applying the following steps to $M^{\times}(p^{ag}_{lang}, p_{spec})$ until none is applicable:*

(a) if $\overleftarrow{q} = [\overleftarrow{a^P}, \overleftarrow{d^S}]$ in Q is an I-state, such that $\overleftarrow{a^P}$ is an alive state and $\overleftarrow{d^S}$ is a dead state, we set $\delta(\overleftarrow{q}, m) = \overleftarrow{q}$ for every m in Σ, and we put $\overleftarrow{q}$ in F;

(b) if $\overleftarrow{q} = [\overleftarrow{d^P}, \overleftarrow{a^S}]$ in Q is an I-state, such that $\overleftarrow{d^P}$ is dead and $\overleftarrow{a^S}$ is alive, we set $\delta(\overleftarrow{q}, m) = \overleftarrow{q}$ for every m in Σ, without modifying F;

(c) if $\overrightarrow{q} = [\overrightarrow{a^P}, \overrightarrow{d^S}]$ in Q is an O-state, such that $\overrightarrow{a^P}$ is alive and $\overrightarrow{d^S}$ is dead, we set $\delta(\overrightarrow{q}, m) = \overrightarrow{q}$ for every m in Σ (without modifying F);

(d) if $\overrightarrow{q} = [\overrightarrow{d^P}, \overrightarrow{d^S}]$ in Q is an O-state, such that $\overrightarrow{d^P}$ is dead and $\overrightarrow{d^S}$ is alive, we set $\delta(\overrightarrow{q}, m) = \overrightarrow{q}$ for every m in Σ, and we put $\overrightarrow{q}$ in F.

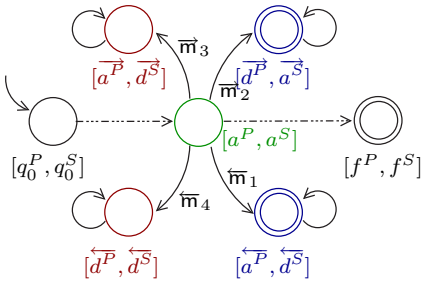

Fig. 1. A general schema of the M_{conf} automaton. From bottom-right, anticlockwise, cases (a), (b), (c), and (d).

These four transformation rules can, intuitively, be explained as follows. Rule (a) handles the case in which, at a certain point of the conversation, according to the policy it is possible to receive a message that, instead, cannot be received according to the specification (it is the case of message $\overleftarrow{\mathsf{m}}_1$ in Fig. 1). Actually, if the agent will interact with another agent that respects the protocol, this message can never be received, so we can ignore the paths generated by the policy from the message at issue onwards. Since this case does not compromise conformance, we want our automaton to accept all these strings. For this reason we set the state as final. Rule (b) handles the symmetric case (Fig. 1, message $\overleftarrow{\mathsf{m}}_4$), in which at a certain point of the conversation it is possible, according to the specification, to receive a message, that is not accounted for by the implementation. In this case the state at issue is turned into a trap state (a state that is not final and that has no transition to a different state); by doing so, all the conversations that are foreseen by the specification from that point onwards will not be accepted by M_{conf}. Rule (c) handles the cases in which a message can possibly be uttered by the agent, according to the policy, but it is not possible according to the specification (Fig. 1, message $\overrightarrow{\mathsf{m}}_3$). In this case, the policy is not conformant, so we transform the current state in a trap state. By doing so, part of the conversations possibly generated by the policy will not be accepted by the automaton. The symmetric case (d) (Fig. 1, message $\overrightarrow{\mathsf{m}}_2$), instead, does not prevent conformance, in fact, an agent is free not to utter a message foreseen by the protocol. However, the conversations that can be generated from that point according to the specification are to be accepted as well. For this reason the state is turned into an accepting looping state. Finally, since a policy is expected to envision at least one of the messages forseen by the protocol, we require that for those states $q^S \in Q^S$, that emit edges labelled with outgoing messages, w.r.t. ag, which are part of strings accepted by $M(p_{spec})$ (legal conversations according to the protocol specification), there is at least one $\mathsf{m}(\overrightarrow{ag})$ such that $\delta^S(q^S, \mathsf{m}(\overrightarrow{ag})) = p^S$ and p^S is an *alive state*. We will denote by $Mess_{q^S}$ the set of all such messages. In this case, we say the automaton is *complete*.

Definition 4 (Complete automaton). *We say that the automaton M_{conf} is* complete *iff for all states of form $[q^P, q^S]$ of M_{conf}, such that $Mess_{q^S} \neq \emptyset$, there is a message $\mathsf{m}(\overrightarrow{ag})' \in Mess_{q^S}$ such that $\delta([q^P, q^S], \mathsf{m}(\overrightarrow{ag})')$ is a state of M_{conf} composed of two alive states.*

One may wonder if the application of rules (b) and (c) could prevent the *reachability of states*, that have been set as accepting states by the other two rules. Notice that their application, cannot prevent the reachability of *alive-alive* accepting states, i.e. those that accept the strings belonging to the intersection of the two languages, because all the four rules only work on dead states. If a state has been set as a trap state (either by rule (b) or (c)), whatever conversation is possibly generated after it by the policy is illegal w.r.t. the specification. So it is correct that the automaton is modified in such a way that the policy language is not accepted by it and that the final state cannot be reached any more.

3.2 Conformance and interoperability

We can now discuss how to check that an agent conforms to a given protocol. The following is our definition of conformance test. It guarantees the expectations that we have explained by examples in Section 2.

Definition 5 (Policy conformance test). *A policy p^{ag}_{lang} is conformant to a protocol specification p_{spec} iff the automaton $M_{conf}(p^{ag}_{lang}, p_{spec})$ is complete and it accepts both languages $L(p^{ag}_{lang})$ and $L(p_{spec})$.*

We are now in position to state that a policy that passes the above test can carry on *any* conformant conversation it is involved in.

Proposition 2. *Given a policy p^{ag}_{lang} that is conformant to a protocol specification p_{spec}, according to Def. 5, for every prefix σ' that is common to the two languages $L(p_{spec})$ and $L(p^{ag}_{lang})$, there is a conversation $\sigma = \sigma'\sigma''$ such that σ is in the intersection of $L(p^{ag}_{lang})$ and $L(p_{spec})$.*

Proof. (*sketch*) If σ' is a common prefix, then it leads to a state of the automaton M_{conf} of the kind $[a^P, a^S]$ (see Figure 1). *If* there is a conversation $\sigma = \sigma'\sigma''$ in $L(p^{ag}_{lang})$, then this must be a legal conversation. In fact, let us consider the general schema of M_{conf} in Figure 1. If p^{ag}_{lang} is conformant, $L(p^{ag}_{lang})$ is accepted by M_{conf}. Then, by construction M_{conf} does not contain any state $[\overrightarrow{a^P}, \overrightarrow{d^S}]$ due to illegal messages uttered by the agent nor it contains any state $[\overleftarrow{d^P}, \overleftarrow{a^S}]$ due to incoming messages that are not accounted for by the policy. Obviously, no conversation σ in $L(p^{ag}_{lang})$ can be accepted by states of the kind $[\overrightarrow{d^P}, \overrightarrow{a^S}]$ because the agent does not utter the messages required to reach such states. Finally, no conversation produced by the agent will be accepted by states of the kind $[\overleftarrow{a^P}, \overleftarrow{d^S}]$ if the interlocutor is also conformant to the protocol, because the latter cannot utter illegal messages. Now, at every step after the state $[a^P, a^S]$ mentioned above, due to *policy conformance* all the incoming messages (w.r.t.

the agent) must be foreseen by the policy. Moreover, due to the *completeness* of M_{conf}, in the case of outgoing messages, the policy must foresee at least one of them. Therefore, from $[a^P, a^S]$ it is possible to perform one more common step. **q.e.d.**

Notice that the *intersection* of $L(p^{ag}_{lang})$ and $L(p_{spec})$ cannot be empty because of policy conformance, and also that Proposition 2 does not entail that the two languages coincide (i.e. the policy is not necessarily a full implementation of the protocol). As a consequence, given that the conversation policies of two agents ag_1 and ag_2, playing the different roles of an interaction protocol p_{spec}, are *conformant* to the specification according to Def. 5, and denoting by I the intersection $\cap_{ag_i}^{i=1,2} L(p^{ag_i}_{lang_i})$, we can prove ag_1 and ag_2 *interoperability*. The demonstration is similar to the previous one. Roughly, it is immediate to prove that every prefix, that is common to the two policies, also belongs to the protocol, then, by performing reasoning steps that are analogous to the previous demonstration, it is possible to prove that a common legal conversation must exist when both policies satisfy the conformance test given by Def. 5.

Theorem 1 (Interoperability). *For every prefix σ' that is common to the two languages $L(p^{ag_1}_{lang_1})$ and $L(p^{ag_2}_{lang_2})$, there is a conversation $\sigma = \sigma'\sigma''$ such that $\sigma \in I$.*

Starting from regular languages, all the steps that we have described that lead to the construction of M_{conf} and allow the verification of policy conformance, are decidable and the following theorem holds.

Theorem 2. *Policy conformance is decidable when $L(p^{ag}_{lang})$ and $L(p_{spec})$ are regular languages.*

Notice that if we do not require M_{conf} to be complete, we could not guarantee the *third expectation* reported in Section 2, which requires that, at every state of the conversation, if a role is supposed to utter a message out of a set of possibilities, the agent's policy envisions *at least one* of them. Thus, we could not guarantee that the two agents, playing the two roles of a same protocol, would be able to lead to an end their conversations. On the other hand, the definition would be sufficient to satisfy the first two expectations. In other words, we can prove that whatever conversation is in the intersection I, it is legal, but we cannot prove that I is not empty.

Proposition 3. *All the conversations that a policy p^{ag}_{lang}, that is conformant according to Def. 5 (without requiring M_{conf} to be complete) to a protocol specification p_{spec}, will produce when it interacts with any agent that is equally conformant to p_{spec}, are always legal w.r.t. this protocol, according to Def. 1.*

4 The DyLOG language: a case study

In this section we show how the presented approach particularly fits logic languages, using as a case of study the DyLOG language [5], previously developed

in our group. The choice is due to the fact that this language explicitly supplies the tools for representing communication protocols and that we have already presented an algorithm for turning a DyLOG program in a regular grammar (therefore, into a finite state automaton) [4]. This is, however, just an example. The same approach could be applied to other logic languages. For this reason we will confine its description to the strict necessary.

DyLOG [5] is a logic programming language for modeling rational agents, based upon a modal logic of actions and mental attitudes, in which modalities represent actions as well as beliefs that are in the agent's mental state. It accounts both for atomic and complex actions, or procedures, for specifying the agent behavior. DyLOG agents can be provided with a *communication kit* that specifies their communicative behavior [3]. In DyLOG *conversation policies* are represented as procedures that compose speech acts (described in terms of their preconditions and effects on the beliefs in the agent's mental state). They specify the agent communicative behavior and are expressed as prolog-like procedures of form:

$$p_0 \textbf{ is } p_1; p_2; \ldots; p_m$$

where p_0 is a procedure name, the p_i's in the body are procedure names, atomic actions, or test actions, and ';' is the sequencing operator.

Besides speech acts, protocols can also contain *get message actions*, used to read incoming communications. From the perspective of an agent, expecting a message corresponds to a query for an external input, thus it is natural to interpret this kind of actions as a special case of sensing actions. As such, their outcome, though belonging to a predefined set of alternatives, cannot be predicted before the execution. A get_message action is defined as:

$$\begin{array}{l}\mathsf{get\_message}(ag_i, ag_j, l) \textbf{ is} \\ \qquad \mathsf{speech\_act}_1(ag_j, ag_i, l) \textbf{ or } \ldots \textbf{or } \mathsf{speech\_act}_k(ag_j, ag_i, l)\end{array}$$

On the right hand side the finite set of alternative incoming messages that the agent ag_i expects from the agent ag_j in the context of a given conversation. The information that is actually received is obtained by looking at the effects that occurred on ag_i's mental state.

From the specifications of the interaction protocols and of the relevant speech acts contained in the domain description, it is possible to trigger a *planning* activity by executing *existential queries* of form *Fs* **after** $p_1; p_2; \ldots; p_m$, that intuitively amounts to determine if there is a possible execution of the enumerated actions after which the condition *Fs* holds. If the answer is positive, a conditional plan is returned. Queries of this kind can be given an answer by a goal-directed proof procedure that is described in [3].

The example that we consider involves a reactive agent. The program of its interlocutor is not given: we will suppose that it adheres to the public protocol specification against which we will check our agent's conformance. The example rephrases one taken from the literature, that has been used in other proposals (e.g. [13]) and, thus, allows a better comprehension as well as comparison. We just set the example in a realistic context. The agent is a web service [2] that answers queries about the movies that are played. Its interlocutor is the requester

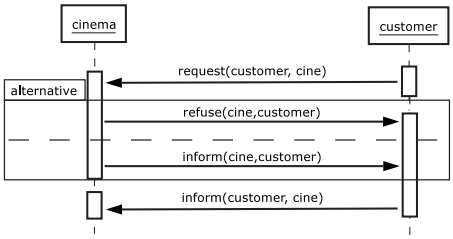

Fig. 2. AUML sequence diagram.

of information (that we do not detail supposing that it respects the agreed protocol). This protocol is described in Fig. 2 as an AUML sequence diagram [20]. It is very simple: the agent that plays the role "cinema" waits for a request from another agent (if a certain movie is played), then, it can alternatively send the requested information (yes or no) or refuse to supply information; the protocol is ended by an acknowledgement message from the customer to the cinema. Hereafter, we consider the implementation of the web service of a specific cinema, written as a DyLOG communication policy. This program has a different aim: it allows answering to a sequence of information requests from the same customer and it never refuses an answer.

(a) get_info_movie$(cine, customer)$ **is**
  get_request$(cine, customer, available(Movie))$;
  send_answer$(cine, customer, available(Movie))$;
  get_info_movie$(cine, customer)$
(b) get_info_movie$(cine, customer)$ **is** get_ack$(cine, customer)$

(c) send_answer$(cine, customer, available(Movie))$ **is**
  $\mathcal{B}^{cinema} available(Movie)?$; inform$(cine, customer, available(Movie))$
(d) send_answer$(cine, customer, available(Movie))$ **is**
  $\neg\mathcal{B}^{cinema} available(Movie)?$; inform$(cine, customer, \neg available(Movie))$

(e) get_request$(cine, customer, available(Movie))$ **is**
  request$(customer, cine, available(Movie)$
(f) get_ack$(cine, customer, ack)$ **is** inform$(customer, cine, ack)$

The question that we try to answer is whether this policy is *conformant* to the given protocol, and we will discuss whether another agent that plays as a customer and that is proved conformant to the protocol will actually be able to *interoperate* with this implementation of the cinema service. For what concerns the AUML sequence diagram, we have proved in [4] that diagrams containing only message, alternative, loop, exit, and reference to a subprotocol operators can be represented as a right-linear grammar, that generates a regular language. The automaton reported in Fig. 3(b) is obtained straightforwardly from this grammar. For what concerns the implementation, by applying the results reported in [4] it is possible to turn a DyLOG program in a context-free language. This grammar captures the structure of the possible conversations

disregarding the semantics of the speech acts. When we have only right-recursion in the program, then, the obtained grammar is right-linear. So also in this case a regular language is obtained, hence the automaton in Fig. 3(a). Notice that all the three automata are represented from the perspective of agent *cine*, so all the short notation for the messages are to be interpreted as incoming or outgoing messages w.r.t. this agent.

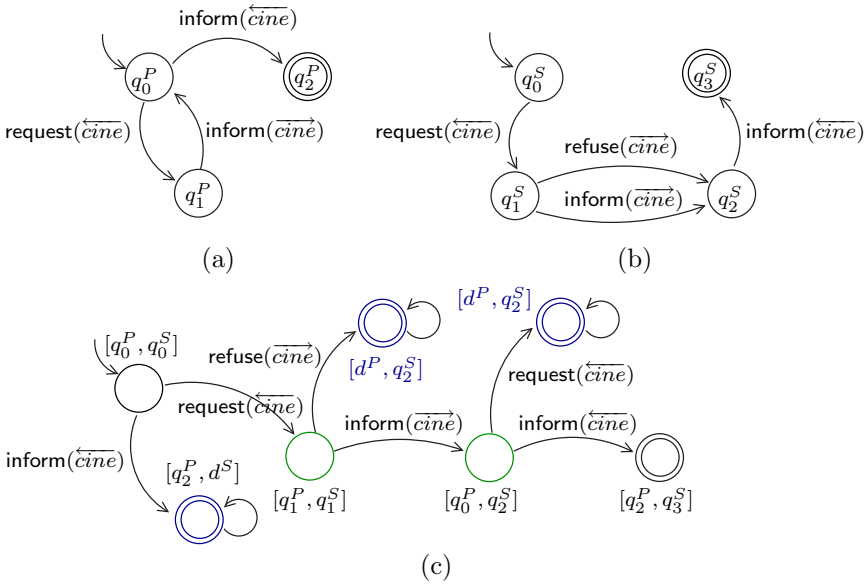

Fig. 3. (a) Policy of agent *cine*; (b) protocol specification; (c) M_{conf} automaton. Only the part relevant to the discussion is shown.

The protocol allows only two conversations between *cine* and *customer* (the content of the message is not relevant in this example, so we skip it): request(*customer*, *cine*) inform(*cine*, *customer*) inform(*customer*, *cine*) and request(*customer*, *cine*) refuse(*cine*, *customer*) inform(*customer*, *cine*). Let us denote this protocol by get_info_movie$_{AUML}$ (AUML is the specification language).

Let us now consider an agent (*cine*), that is supposed to play as cinena. This agent's policy is described by the above DyLOG program. The agent has a *reactive behavior*, that depends on the message that it receives, and its policy allows an infinite number of conversations of any length. Let us denote this language by get_info_movie$^{cine}_{DyLOG}$. In general, it allows all the conversations that begin with a (possibly empty) series of exchanges of kind request($\overleftarrow{cine}$) followed by inform($\overrightarrow{cine}$), concluded by a message of kind inform($\overleftarrow{cine}$).

To verify its conformance to the protocol, and then state its interoperability with other agents that respect such protocol, we need to build the M_{conf}

automaton for the policy of *cine* and the protocol specification. For brevity, we skip its construction steps and directly report M_{conf} in Fig. 3(c).

Let us now analyse M_{conf} for answering our queries. Trivially, the automaton is complete and it accepts both languages (of the policy and of the specification), therefore, $\mathsf{get\_info\_movie}^{cine}_{DyLOG}$ is policy conformant to $\mathsf{get\_info\_movie}_{AUML}$. Moreover, when the agent interacts with another agent *customer* whose policy is conformant to $\mathsf{get\_info\_movie}_{AUML}$, the messages $\mathsf{request}(\overleftarrow{cine})$ and $\mathsf{inform}(\overleftarrow{cine})$ will not be received by *cine* in all the possible states it expects them. The reason is simple: for receiving them it is necessary that the interlocutor utters them, but by definition (it is conformant) it will not. The fact that $\mathsf{refuse}(\overrightarrow{cine})$ is never uttered by *cine* does not compromise conformance.

5 Conclusions and related work

In this work we propose an approach to the verification of the conformance of an agent's conversation policy to a public conversation protocol, which is based on the theory of *formal languages*. Differently than works like [1], where the compliance of the agents' communicative behavior to the protocol is verified at run-time, we tackled the verification of *a priori* conformance, a property of the *policy* as a whole and not of the on-going conversation only.

This problem has been studied by other researchers, the most relevant analysis probably being the one by Endriss et al. and reported in [10]. Here, the problem was faced in a logic framework; the authors introduce three degrees of conformance, namely *weak*, *exhaustive*, and *robust conformance*. An agent is weakly conformant to a protocol iff it never utters any dialogue move which is not a legal continuation (w.r.t. the protocol) of any state of the dialogue the agent might be in. It is *exhaustively conformant* to a protocol iff it is weakly conformant to it and, for every received legal input, it will utter one of the expected dialogue moves. It is *robustly conformant* iff it is exhaustively conformant and for any illegal input move received it will utter a special dialogue move (such as not-understood) indicating this violation. Under the assumption that in their conversations the agents strictly alternate in uttering messages (ag_1 tells something to ag_2 which answers to ag_1 and so on), Endriss and collegues show that by their approach it is possible to prove *weak conformance* in the case of logic-based agents and shallow protocols [1].

Our *Policy conformance* (Def. 5) guarantees that an agent, at any point of its conversations, can only utter messages which are legal w.r.t. the protocol, because of the M_{conf} construction step, given by rule (c). In this respect it entails *weak conformance* [10], however, our notion of conformance differs from it because it also guarantees that whatever incoming message the agent may receive, in any conversation context, its policy will be able to handle it.

The second very important property that is guaranteed by our proposal is that, given two policies *each* of which is *conformant* to a protocol specification,

[1] A protocol is shallow when the current state is sufficient to decide the next action to perform. This is not a restriction.

their *interoperability* is guaranteed. In other words, we captured the expectation that conformance, a property of the single policy w.r.t. the public protocol, should in some way guarantee agents (legal) interoperability. Interoperability is not mentioned by Endriss et al., who do not formally prove that it is entailed by (all or some of) their three definitions.

Moreover, we do not limit in any way the structure of the conversations (in particular, we do not require a strict alternation of the uttering agents) nor we focus on a specific class of implementation or specification languages. One further characteristic is that, according to Def. 5, a policy may also expect incoming messages, that are not expected by the protocol specification, for this does not prevent the correct interaction with another conformant agent, and it is not requested to implement a whole set of alternative outgoing messages that are considered possible by the protocol.

This work is, actually, a deep revision of the work that the authors presented at [4], where the verification of a priori conformance was faced only in the specific case in which DyLOG [5] is used as the policy implementation language and AUML [20] is used as the protocol specification language. Basically, in that work the idea was to turn the problem into a problem of *formal language inclusion*. The two considered languages are the set of all the possible conversations foreseen by the protocol specification, let us denote it by $L(p_{AUML})$, and the set of all the possible conversations according to the policy of agent ag, let us denote it by $L(p^{ag}_{dylog})$. The conformance property could then be expressed as the following inclusion: $L(p^{ag}_{dylog}) \subseteq L(p_{AUML})$. The current proposal is more general than the one in [4], being independent from the implementation and specification languages. Moreover, as we have explained in the introduction, the interpretation of conformance as an inclusion test is too restrictive, on a hand, and not sufficient to express all the desiderata connected to this term, which are, instead, well-captured by our definitions of policy conformance.

Finally, to the best of our knowledge, in those works that address the problem of verifying the conformance in systems of communicating agents by using model checking techniques (e.g. [13]), the issue of interoperability is not tackled or, at least, this does not clearly emerge. For instance, Giordano, Martelli and Schwind [13] based their approach on the use of a dynamic linear time logic. Protocols are specified, according to a social approach, by means of temporal constraints representing permissions and commitments. Following [15] the paper shows how to prove that an agent is compliant with a protocol, given the program executed by the agent, by assuming that all other agents participating in the conversation are compliant with the protocol, i.e. they respect their permissions and commitments. However, this approach does not guarantee interoperability.

References

1. 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.

2. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services*. Springer, 2004.
3. M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about self and others: communicating agents in a modal action logic. In *ICTCS'2003*, volume 2841 of *LNCS*, pages 228–241. Springer, October 2003.
4. M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Verifying protocol conformance for logic-based communicating agents. In *Proc. of 5th Int. Workshop on Computational Logic in Multi-Agent Systems, CLIMA V*, LNCS, 2005.
5. M. Baldoni, L. Giordano, A. Martelli, and V. Patti. Programming Rational Agents in a Modal Action Logic. *Annals of Mathematics and Artificial Intelligence, Special issue on Logic-Based Agent Implementation*, 41(2-4):207–257, 2004.
6. J. Bentahar, B. Moulin, J. J. Ch. Meyer, and B. Chaib-Draa. A computational model for conversation policies for agent communication. In *Pre-Proc. of CLIMA V*, pages 66–81, 2004.
7. R. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model Checking AgentSpeak. In *Proc. of AAMAS 2003*, 2003.
8. L. Cabac and D. Moldt. Formal semantics for auml agent interaction protocol diagrams. In *Proc. of AOSE 2004*, pages 47–61, 2004.
9. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Protocol conformance for logic-based agents. In *Proc. of IJCAI-2003*, pages 679–684, 2003.
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. R. Eshuis and R. Wieringa. Tool support for verifying UML activity diagrams. *IEEE Trans. on Software Eng.*, 7(30), 2004.
12. FIPA. Fipa 97, specification part 2: Agent communication language. Technical report, FIPA (Foundation for Intelligent Physical Agents), November 1997.
13. L. Giordano, A. Martelli, and C. Schwind. Verifying communicating agents by model checking in a temporal action logic. In *JELIA'04*, volume 3229 of *LNAI*, pages 57–69, Lisbon, Portugal, 2004. Springer-Verlag.
14. F. Guerin. *Specifying Agent Communication Languages*. PhD thesis, Imperial College, London, April 2002.
15. 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.
16. J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley Publishing Company, 1979.
17. M. P. Huget and J.L. Koning. Interaction Protocol Engineering. In *Communication in Multiagent Systems*, *LNAI* 2650, pages 179–193. Springer, 2003.
18. A. Mamdani and J. Pitt. Communication protocols in multi-agent systems: A development method and reference architecture. In *Issues in Agent Communication*, volume 1916 of *LNCS*, pages 160–177. Springer, 2000.
19. N. Maudet and B. Chaib-draa. Commitment-based and dialogue-based protocols: new trends in agent communication languages. *Knowledge engineering review*, 17(2), 2002.
20. J. Odell, H. V. D. Parunak, and B. Bauer. Extending UML for agents. In *Proc. of the Agent-Oriented Information System Workshop at AAAI'00*. 2000.
21. M. P. Singh. A social semantics for agent communication languages. In *Proc. of IJCAI-98 Workshop on Agent Communication Languages*, Berlin, 2000. Springer.
22. C. Walton. Model checking agent dialogues. In *Int. Workshop on Declarative Agent Languages and Technology*, pages 156–171, 2004.