I2-D10

Negotiation Analysis and Design: Reasoning on Policies for Verifying Properties

Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Project number:	IST-2004-506779
Project instrument:	EU FP6 Network of Excellence (NoE)
Project thematic priority:	Priority 2: Information Society Technologies (IST)
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	IST506779/Turin/I2-D10/D/PU/a1
Responsible editors:	Alberto Martelli
Reviewers:	Daniel Olmedilla and Bernhard Lorenz
Contributing participants:	Torino, Napoli
Contributing workpackages:	I2, A3
Contractual date of deliverable:	3 September 2006
Actual submission date:	5 September 2005

Abstract
The present report continues the work of deliverable I2-D2 [Bonatti and Olmedilla, 2005b] by presenting a series of reasoning techniques, that allow several kinds of verifications, such as compliance at run-time, verification of properties, goal achievement, capability checking, and interoperability. We will show how techniques, that are typically used in artificial intelligence, can be usefully adapted so to deal with policies on the web, improving the skill of negotiating of the involved entities. By reasoning on a policy before its adoption, an agent can customize it, refuse it if it does not allow the achievement of its own purposes, modify it to make it more profitable, avoid run-time errors.

Keyword List
semantic web, reasoning, policies, protocols, verification, conformance, temporal logics

Project co-funded by the European Commission and the Swiss Federal Office for Education and Science within the Sixth Framework Programme.

Negotiation Analysis and Design: Reasoning on Policies for Verifying Properties

Matteo Baldoni[1], Cristina Baroglio[1], Piero A. Bonatti[2], Laura Giordano[3], Alberto Martelli[1], Viviana Patti[1], and Claudio Schifanella[1]

[1] Dipartimento di Informatica, Università degli Studi di Torino
Email: {baldoni,baroglio,mrt,patti,schi}@di.unito.it

[2] Dipartimento di Scienze Fisiche - Sezione di Informatica,
Università di Napoli "Federico II"
Email: bonatti@na.infn.it

[3] Dipartimento di Informatica, Università del Piemonte Orientale
Email: laura@di.unipmn.it

5 September 2005

Abstract
The present report continues the work of deliverable I2-D2 [Bonatti and Olmedilla, 2005b] by presenting a series of reasoning techniques, that allow several kinds of verifications, such as compliance at run-time, verification of properties, goal achievement, capability checking, and interoperability. We will show how techniques, that are typically used in artificial intelligence, can be usefully adapted so to deal with policies on the web, improving the skill of negotiating of the involved entities. By reasoning on a policy before its adoption, an agent can customize it, refuse it if it does not allow the achievement of its own purposes, modify it to make it more profitable, avoid run-time errors.

Keyword List
semantic web, reasoning, policies, protocols, verification, conformance, temporal logics

Contents

1 Executive summary

The work in working group I2 is focussed on the representation and reasoning about policies. In a previous deliverable [Bonatti and Olmedilla, 2005b] (see also [Bonatti and Olmedilla, 2005a]) the syntax and semantics of the core of Protune, the policy language and metalanguage of REWERSE, have been presented. The language can specify access control policies, privacy policies, reputation-based policies, provisional policies, and a class of business rules.

This report continues the mentioned work by presenting a series of reasoning techniques, that allow several kinds of verifications, such as compliance at run-time, verification of properties, goal achievement, capability checking, and interoperability. We will show how techniques, that are typically used in artificial intelligence, can be usefully adapted so to deal with policies on the web, improving the skill of negotiating of the involved entities. By reasoning on a policy before its adoption, an agent can customize it, refuse it if it does not allow the achievement of its own purposes, modify it to make it more profitable, and avoid run-time errors.

In Section 2 we interpret policies as conversations by means of an example that has already been used in previous deliverables and that will also be used throughout this document. In Section 3, we sketch an alternative declarative representation of policies, based on Dynamic Linear Time Temporal Logic, that enables the verification of run-time compliance and of the properties of the interaction. In Section 4 we show how, by using representation and reasoning techniques derived from logics for reasoning about actions and change, it is possible to perform existential verifications on the possible policy execution traces. The aim is to check the reachability of goals of interest. In Section 5, we discuss the actual executability of an adopted policy, introducing the concepts of "capability" and "capability requirement" in a policy description. Last but not least, in Section 6 we consider the problem of policy interoperability, for verifying that agents can actually interact by means of exchanged and possibly customized policies. Each of these sections contains full references to the main papers, attached in the Appendix, where details about approaches and techniques can be found. Some conclusions end the document.

The authors would like to thank for their kindness and support the reviewers of this document, Bernhard Lorentz and Daniel Olmedilla.

2 Policies as conversations

Let us consider the example in [Bonatti and Olmedilla, 2005b]. *alice* requests a discount from the online provider *elearn*. *elearn* has the following policies:

E1 : in order to get a discount, *alice* must prove that she is a European citizen and a student;

E2 : its membership to Better Business Bureau (BBB) can be disclosed to anybody.

alice has the following policies:

A1 : her citizenship can be disclosed to anybody;

A2 : she is willing to release her student id only to companies member of BBB.

We will model the interactions between *alice* and *elearn*, by means of communicative actions such as:

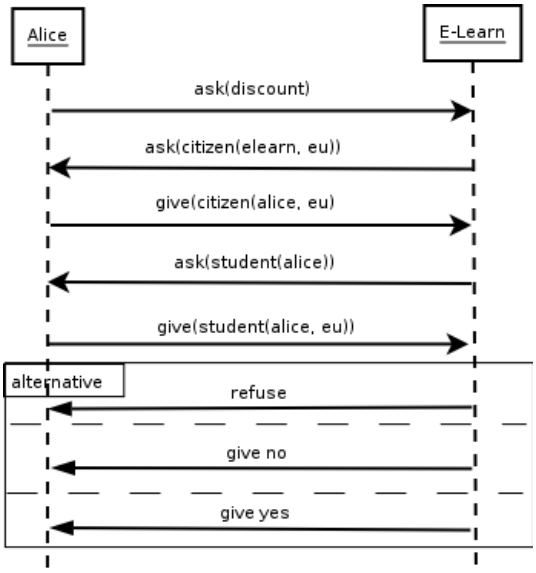


Figure 1: The policy E1, represented by means of UML sequence diagrams.

$$ask(S, R, X), grant(S, R, X), ask(S, R, Cred), give(S, R, Cred)$$

where S is the sender, R is the receiver, X is a permission to be granted, and $Cred$ is a credential.

The above policies can be represented by a sequence of communicative actions exchanged between *alice* and *elearn*. For instance, policy E1 can be described by the following sequence:

$$ask(alice, elearn, discount); ask(elearn, alice, citizen(A, eu)); give(alice, elearn, citizen(alice, eu));$$
$$ask(elearn, alice, stud(alice)); give(alice, elearn, stud(alice)); grant(elearn, alice, discount)$$

meaning that *alice* requests a discount to *elearn*, which, before granting it, asks to *alice* her credentials about citizenship and gets them, and similarly asks her student ID and gets it from her. Figure 1 describes the interaction as a UML sequence diagram.

This conversation can be interleaved with other conversations modeling other policies. For instance, before giving her student ID, *alice* will ask *elearn* about its membership to BBB.

By combining all policies, we can obtain all allowed conversations between *alice* and *elearn*.

From this example it is possible to identify two alternative situations:

- *elearn* publically declares its policies;
- *elearn* discloses some of its policies only when necessary and, even in this case, only if the disclosure is considered as favourable.

In the first situation, when *alice* tries to become one of its clients, *elearn* immediately sends to it all the information about itself (therefore the fact that it is a member of BBB) as well as its policies. In the second situation, instead, it will say that it is a member of BBB only

on demand (when the need arises). In the example *elearn* does not constrain the disclosure of this information any further, but in general it could do it, for instance by constraining it to the verification of other conditions, possibly related to further information obtained from *alice*. In the latter case, the interaction between the two parts could be mediated by a negotiator, like the one of Protune.

In both cases, seeing policies as sets of conversations, or more abstractly as "rules of conversation", is useful because it enables the possibility of performing various interesting forms of reasoning. The literature about systems of communicating agents comes to help in this perspective. In particular, given a set of policies it is possible to find out whether there are some conversations that allow the agent to achieve a given goal. Such conversations will respect all of the policies, therefore they will be by definition safe (or opportune or respectful of privacy conditions or legal, depending on the meaning and the purpose of the policies at hand). To this aim it is possible to consider communicative acts as atomic actions, policies as complex actions, and apply planning techniques. These issues are discussed in Section 3 and in Section 4. In the former, the properties of a set of policies by means of temporal logics and model checking techniques are proposed. By these techniques it is, for instance, possible to verify if a set of policies can run into a deadlock or if all the executions that allow reaching a goal of interest show a certain property. In Section 4, procedural planning techniques are applied in a modal action logic framework with the aim of pursueing goals.

An agent which adopts a policy should own a definition of all the actions that this includes. Such definitions must use the same action names and the same number and order of parameters as those expected by the policy. An alternative, as proposed in Protune, is to adopt a policy plus all the action definitions (ontology sharing). This is, however, quite a strong assumption. An interesting issue concerns reasoning on the adoption of the policy, possibly modifying it. This issue is discussed in Section 5. Modifying a policy could compromise the ability of interacting through the policy. For this reason, it is important to have techniques for checking the interoperability of a set of policies or of a policy against some specification. We discuss this issue in Section 6.

3 Declarative representations of policies

In this section we outline a general approach for modeling policies and reasoning about them.

The approach is based on the Dynamic Linear Time Temporal Logic ($DLTL$), which extends linear-time temporal logic by strengthening the *until* operator by indexing it with the regular programs of dynamic logic. We adopt a social approach to agent communication, in which communicative actions affect the "social state" of the system, rather than the internal (mental) states of the agents. The social state records social facts, like the permissions and the commitments of the agents. The dynamics of the system emerges from the interactions of the agents, which must respect these permissions and commitments (if they are compliant with the protocol). The social approach allows a high level specification of the protocol, and does not require the rigid specification of the allowed action sequences. It is well suited for dealing with "open" multiagent systems, where the history of communications is observable, but the internal states of the single agents may not be observable.

The communicative actions are specified in an action theory, which, by means of DLTL, describes effects and preconditions of the actions on a global (social) state. A state consists of a set of *fluent literals*, where a fluent literal l is an atomic proposition f or its negation $\neg f$. In the

following we will make use of *epistemic fluents* of the form $\mathcal{K}_A(l)$, to represent the knowledge of agent A.

A *domain description* D is defined as a set of *action laws*, *causal laws*, and a set of *constraints*.

Action laws have the form: $\Box(\alpha \rightarrow [a]\beta)$, where a is an action and α, β arbitrary formulas, meaning that executing action a in a state where precondition α holds causes the effect β to hold.

Causal laws have the form: $\Box((\alpha \wedge \bigcirc\beta) \rightarrow \bigcirc\gamma)$, meaning that if α holds in a state and β holds in the next state, then γ also holds in the next state. Such laws are intended to expresses "causal" dependencies among fluents.

Constraints are arbitrary temporal formulas of $DLTL$. In particular, the set of constraints $\mathcal{C}$ contains all the temporal formulas which might be needed to constrain the behaviour of an interaction protocol, including the value of fluents in the initial state. In particular, the set of constraints $\mathcal{C}$ includes the precondition laws.

Precondition laws have the form: $\Box(\alpha \rightarrow [a]\bot)$, meaning that the execution of an action a is not possible if α holds (i.e. there is no resulting state following the execution of a if α holds). This formulation is required because we are using a linear-time logic. Preconditions determine when an action can be executed by an agent: they describe *permissions* of agents.

For instance, in our example, we can express the fact that *alice* cannot give her student ID if she does not know whether *elearn* is member of BBB by the precondition law:

$$\Box(\neg\mathcal{K}_{alice}(member(elearn, BBB)) \rightarrow [give(alice, elearn, stud(alice))]\bot)$$

while the action law specifying the effect of the action will be

$$\Box[give(alice, elearn, stud(alice))]\mathcal{K}_{elearn}(stud(alice))$$

that is, after execution of the action, the fact $stud(alice)$ will be known in the social state, in particular to *elearn*.

In the social semantics, the effects of an action can be *commitments*, which in our formulation are special fluents. These fluents can represent *base-level commitments* and have the form $C(i, j, \alpha)$, meaning that agent i is committed to agent j to bring about α, where α is an arbitrary formula, or they can be *conditional commitments* of the form $CC(i, j, \beta, \alpha)$ (agent i is committed to agent j to bring about α, if the condition β is brought about).

We will be interested in the executions of the protocol in which all commitments have been fulfilled. We can express the condition that the commitment $C(i, j, \alpha)$ will be fulfilled by the following constraint:

$$\Box(C(i, j, \alpha) \rightarrow \Diamond\alpha)$$

For instance, we can introduce conditional commitments to express that *alice* will be committed to execute a $give(alice, elearn, X)$ action whenever an $ask(elearn, alice, X)$ is received, and similarly for actions of *elearn*. Of course, *give* can be executed only after its preconditions have been satisfied.

In the same way, we could extend the example by introducing actions for refusing to give requested information.

In the following we assume a *static* formulation of the example, i.e. we assume all policies to be known at the beginning. In this case, all policies can be described by a set of formulas as outlined above, that we call *domain description*, which defines all possible interactions between *alice* and *elearn*. More precisely we can say that, given the specification of the interactions

between two agents by a domain description D, the "runs" satisfying the specification are exactly the models of D (remember that we are using a linear-time logic and thus each model consist of a sequence of actions).

Once the interactions between two agents have been defined by specifying their policies, several kinds of *verifications* can be performed, such as the verification of the compliance at runtime of the agents with the specification, or the verification of properties of the interaction. For instance, we may want to check if there is at least a run allowing *alice* to get the discount, or if there is a run such that *alice* gets the discount without disclosing her student ID. The latter case can be formulated as a satisfiability problem, by checking if there is a model satisfying the constraint $\neg\Diamond\langle give(alice, elearn, stud(A))\rangle\top$.

In [Giordano et al., 2004] we show how in many cases verifications can be performed by means of a model checking approach, by suitably adapting standard model checking techniques to the DLTL. More details can be found in

> *Laura Giordano, Alberto Martelli, and Camilla Schwind. Verifying communicating agents by model checking in a temporal action logic. In J. Alferes and J. Leite, editors, 9th European Conference on Logics in Artificial Intelligence (JELIA'04), volume 3229 of LNAI, pages 57-69, Lisbon, Portugal, Sept. 2004. Springer-Verlag.*

attached in the Appendix.

4 Reasoning on policies for pursueing goals

In Section 2 we have seen, by means of an example, how a set of policies can be represented by sequences of communicative acts. One of the most significant approaches to the representation of communicative acts is to consider them as special actions, which have an effect only on the state of the agents that exchange them. In this section we, first, outline an approach for representing communicative acts as *atomic actions* and policies as *complex actions*, and, second, we see a reasoning technique, that can be applied by agents to a set of exchanged policies for deciding whether it is opportune to adopt them for interacting, before the interaction actually takes place.

In our proposal each agent has a personal view of the world consisting of a private mental state, which contains a consistent set of beliefs about the world and beliefs about the beliefs of other agents. The modal operator $\mathbf{B}^{ag_i}$ models the beliefs of the agent ag_i. The modal operator $\mathbf{M}^{ag_i}$ is defined as the dual of $\mathbf{B}^{ag_i}$ ($\mathbf{M}^{ag_i}\varphi \equiv \neg\mathbf{B}^{ag_i}\neg\varphi$); intuitively it represents the fact that agent ag_i considers φ possible. A belief state provides, for each agent, a three-valued interpretation of all the possible belief arguments L, that can either be true, false, or undefined when both $\neg\mathbf{B}^{ag_i}L$ and $\neg\mathbf{B}^{ag_i}\neg L$ hold. $\mathbf{U}^{ag_i}L$ expresses the ignorance of ag_i about L.

A communicative act can be represented as an *atomic action* of the form $\mathsf{speech\_act}(sender, receiver, l)$ where *sender* and *receiver* are agents while l represents the content of the communication. The way in which an agent's beliefs are modified by the execution of a speech act depend on whether it acts as the sender or the receiver. The specification is, therefore, twofold: one definition holds when the agent is the *sender*, the other when it is the *receiver*. When ag_i is the receiver, the action is supposed as being *always* executable (ag_i has no control over a communication performed by another agent). With reference to our example, the communicative act *ask*, that is used by *elearn* to ask *alice* if it is a student, could be represented as $\mathsf{ask}(elearn, alice, student(alice))$, where ask is a speech act defined as:

$\mathsf{ask}(Self, Other, l)$
a) $\Box(\mathbf{U}^{Self} l \wedge \neg \mathbf{B}^{Self} \mathbf{U}^{Other} l \supset \langle \mathsf{ask}(Self, Other, l) \rangle \top)$
b) $\Box(\top \supset \langle \mathsf{ask}(Other, Self, l) \rangle \top)$
c) $\Box([\mathsf{ask}(Other, Self, l)] \mathbf{B}^{Self} \mathbf{U}^{Other} l)$

By ask an agent queries another agent if it believes that l is true. The precondition to perform a ask act is that $Self$ is ignorant of l (in the example *elearn* must ignore if *alice* is a student) and it must also believe that the receiver does not ignore l (*elearn* believes that *alice* knows if it is a student or not), clause (a). The speech act has an effect also on the mental state of the receiver, in fact after a ask act, the receiver will believe that the sender ignores l (by receiving this request, agent *alice* will believe that *elearn* is ignorant about its being a student).

Another example of atomic action is give which is used in the example for sending a piece of information:

$\mathsf{give}(Self, Other, l)$
a) $\Box(\mathbf{B}^{Self} l \wedge \mathbf{B}^{Self} \mathbf{U}^{Other} l \wedge \mathbf{B}^{Self} public(l) \supset \langle \mathsf{give}(Self, Other, l) \rangle \top)$
b) $\Box([\mathsf{give}(Self, Other, l)] \mathbf{M}^{Self} \mathbf{B}^{Other} l)$
c) $\Box(\mathbf{B}^{Self} \mathbf{B}^{Other} authority(Self, l) \supset [\mathsf{give}(Self, Other, l)] \mathbf{B}^{Self} \mathbf{B}^{Other} l)$
d) $\Box(\top \supset \langle \mathsf{give}(Other, Self, l) \rangle \top)$
e) $\Box([\mathsf{give}(Other, Self, l)] \mathbf{B}^{Self} \mathbf{B}^{Other} l)$
f) $\Box(\mathbf{B}^{Self} authority(Other, l) \supset [\mathsf{give}(Other, Self, l)] \mathbf{B}^{Self} l)$
g) $\Box(\mathbf{M}^{Self} authority(Other, l) \supset [\mathsf{give}(Other, Self, l)] \mathbf{M}^{Self} l)$

Clause (a) specifies those mental conditions that make this action executable in a state. Intuitively, $Self$ can execute a give act only if it believes l and it believes that the receiver ($Other$) does not know l. In turn, l must be declared as *public*. According to clause (b), the agent also considers possible that the receiver will adopt its belief, although it cannot be sure that this will happen (*autonomy assumption*). Nevertheless, if agent $Self$ thinks to be considered by the receiver a *trusted authority* about l, it is also confident that $Other$ will adopt its belief, clause (c). Since executability preconditions can be tested only on the mental state of $Self$, when $Self$ is the receiver the action is considered as *always* executable, clause (d). When $Self$ is the receiver, the effect of a give act is that $Self$ will believe that l is believed by the sender ($Other$), clause (e), but $Self$ will adopt l as an own belief only if it thinks that $Other$ is a trusted authority, clause (f).

Policies require to specify patterns of communication. In [Baldoni et al., 2006c] patterns of conversation are specified by the so called *conversation protocols*, which define the context in which speech acts are executed [Mamdani and Pitt, 2000]. They are modelled by means of *procedure axioms*, having the form:

$$\langle p_0 \rangle \varphi \subset \langle p_1; p_2; \ldots; p_m \rangle \varphi \tag{1}$$

where p_0 is a procedure name, ";" is the *sequencing operator* of dynamic logic, and the p_i's, $i \in [1, m]$, are procedure names, atomic actions, or test actions. Procedure definitions may be *recursive* and procedure clauses can be executed in a *goal-directed* way, similarly to standard logic programs. Since agents have a subjective perception of the communication, each protocol has as many procedural representations as the possible *roles* in the conversation. So, for instance, policy E1 in the example would include two roles: one for *alice*, the discount requester (role $\mathsf{E1}_{alice}$), and one for *elearn* (role $\mathsf{E1}_{elearn}$), which has to decide about it. Here is one possible definition of the role $\mathsf{E1}_{alice}$ of policy E1 (see Figure 1) in the outlined framework:

(a) $\langle \mathsf{E1}_{alice}(Self, Other, Discount)\rangle\varphi \subset$
$\langle \mathsf{ask}(Self, Other, Discount);$
$\mathsf{get\_ask}(Self, Other, citizen(Self, eu)); \mathsf{give}(Self, Other, citizen(Self, eu));$
$\mathsf{get\_ask}(Self, Other, student(Self)); \mathsf{give}(Self, Other, student(Self));$
$\mathsf{get\_answer}(Self, Other, Discount)\rangle\varphi$

(b) $[\mathsf{get\_answer}(Self, Other, Fluent)]\varphi \equiv$
$[\mathsf{give}(Other, Self, Fluent) \cup \mathsf{give}(Other, Self, \neg Fluent) \cup$
$\mathsf{refuseInform}(Other, Self, Fluent)]\varphi$

(c) $[\mathsf{get\_ask}(Self, Other, Fluent)]\varphi \equiv [\mathsf{ask}(Other, Self, Fluent)]\varphi$

In $\mathsf{E1}_{alice}$ agent $Self$ (*alice* in our case) performs a ask speech act then it waits for a request of agent $Other$ about its being a citizen of the EU. Afterwards, when it guarantees being an EU citizen, it waits for a request of $Other$ about its being a student. After certifying also this, $Self$ waits for the decision of $Other$. Such an answer can be positive, negative or refused, according to the definition of get_answer reported in (b). Notice that the preconditions of give impose that the information can be sent only if it holds in $Self$ mental state. If it does not hold, the execution of the policy will fail.

In this context it is possible to formalize the *temporal projection* and the *planning* problems by means of existential queries of form:

$$\langle p_1\rangle\langle p_2\rangle \ldots \langle p_m\rangle Fs \tag{2}$$

where each p_k, $k = 1, \ldots, m$ may be an (atomic or complex) action executed by ag_i or a speech act, in which the agent plays the role of the receiver. Checking if a query of form (2) succeeds corresponds to answering the question "Is there an execution trace of the sequence p_1, ..., p_m that leads to a state where the conjunction of belief fluents Fs holds for agent ag_i?". In case all the p_k's are atomic actions, it amounts to predicting if the condition of interest will be true after their execution. In case complex actions are involved, the execution trace that is returned in the end is a, possibly conditional, *plan* to bring about Fs. In this process, the actions used for querying the interlocutor about something are treated as sensing actions, whose outcome is not known at planning time – agents cannot know in advance the answers that they will receive. Therefore, all of the possible alternatives are to be taken into account.

Reasoning techniques of this kind can be applied by an agent which has to decide *whether or not to adopt a policy* that it has just received from some counterpart, which has disclosed it. In our example, let us suppose, for instance, that *alice* has received from *elearn* policy $\mathsf{E1}_{alice}$. It makes sense for *alice* to adopt it if it can be sure that there is a possibility for it to actually get a discount by following it. In this case, it could apply the planning process to $\mathsf{E1}_{alice}$ to answer to the existential query:

$$\langle \mathsf{E1}_{alice}\rangle \mathbf{B}^A discount$$

If we suppose that the information required by *elearn* (that *alice* is a student and a EU citizen) are available and public, then *alice* will conclude that such a possibility exists, identifying the linear plan:

$\mathsf{ask}(alice, elearn, discount)$;
$\mathsf{ask}(elearn, alice, citizen(alice, eu))$;
$\mathsf{give}(alice, elearn, citizen(alice, eu))$;

ask($elearn, alice, student(alice)$) ;
give($alice, elearn, student(alice)$) ;
give($elearn, alice, discount$)

Instead, if the fact that it is a student is private, no plan will be found. Therefore, *alice* will decide that it is useless to interact with *elearn*, keeping also the information about its citizenship private. Besides finding out that it can actually achieve its goal, *alice* will also find out the assumptions that it has to make in order to achieve it. In the plan above the final decision about giving a discount is up to *elearn*, so *alice* cannot be sure that it will be received at execution time.

Details about the approach can be found in

> *Matteo Baldoni, Cristina Baroglio, Alberto Martelli, and Viviana Patti. Reasoning about interaction protocols for customizing web service selection and composition. Journal of Logic and Algebraic Programming, special issue on Web Services and Formal Methods, 2006. In press.*

attached in the Appendix.

5 Capabilities

In the previous section, we have made the implicit assumption that when an agent adopts a policy, that it has received, it already owns a definition for all the atomic actions (the specific communicative acts) that are involved. Such definitions must use the same action names and the same number and order of parameters as those expected by the policy. This is, however, quite a strong assumption. As an example, consider action ask defined and used above. It is quite easy to recognise by its definition that it corresponds to the FIPA *queryIf* speech act but the use of a different name would not allow agents adopting the standard FIPA naming to use policy E1. An easy way of solving this problem is to send not only the policy that should be adopted but also a reference to the speech act ontology that should be used.

A more subtle problem concerns the ability of an agent of building the information that is requested from it. In policy E1, *elearn* asks *alice* two things: if it is a student and if it is an EU citizen. *alice* can execute the policy only if it can produce this information (see Figure 2). In other words, we should consider an adopted policy as a *skeleton* but a problem arises: policies only concern communication patterns, abstracting from all references to the internal state of the player and from all actions/instructions that do not concern the interaction ruled by the policy. Nevertheless, the policy is to be *executed* by the agent and, for permitting the execution, it is necessary to express to some extent also this kind of actions. We will refer to such action descriptions as *capability requirements* [Baldoni et al., 2006d].

The term "capability" has recently been used by [Padgham and Lambrix, 2000] (the work is inspired by JACK and it is extended in [Padmanabhan et al., 2001]), in the BDI framework, for identifying the "ability to react rationally towards achieving a particular goal". More specifically, an agent has the capability to achieve a goal if its plan library contains at least one plan for reaching the goal. The authors incorporate this notion in the BDI framework so as to constrain an agent's goals and intentions to be compatible with its capabilities. This notion of capability is orthogonal w.r.t. what proposed in [Baldoni et al., 2006d]. In this perspective,

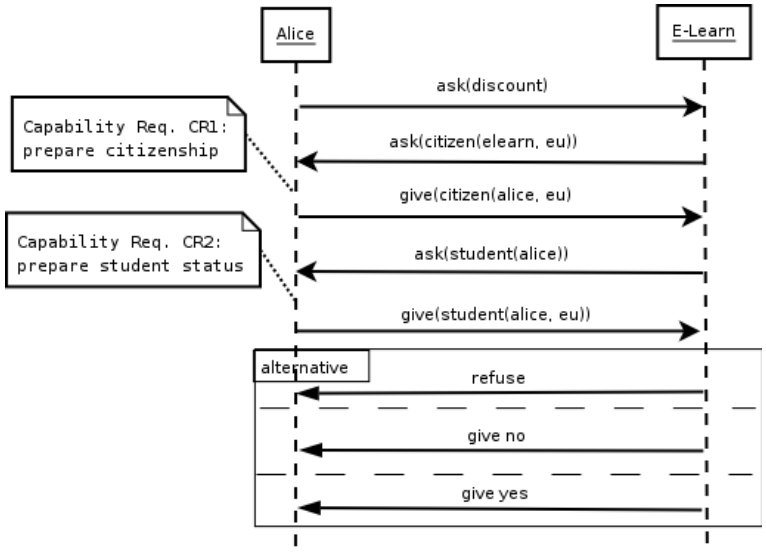


Figure 2: The E1 policy, represented by means of UML sequence diagrams, and enriched with capability specifications.

our notion of capability resembles more closely (sometimes unnamed) concepts, that emerge in a more or less explicit way in various frameworks/languages, in which there is a need for defining interfaces. One example is Jade [Jade, 2001], the well-known platform for developing multi-agent systems. In this framework policies are supplied as partial implementations with "holes" that the programmer must fill with code when creating agents. Such holes are represented by methods whose body is not defined. The task of the programmer is to implement the specified methods, whose name and signature is, however, fixed in the partial policy. Another example is powerJava [Baldoni et al., 2006e, Baldoni et al., 2006f], an extension of the Java language that accounts for roles and institutions. Without getting into the depths of the language, a role in powerJava represents an interlocutor in the interaction schema. A role definition contains only the implementation of the interaction schema and leaves to the role-player the task of implementing the internal actions. Such calls to the player's internal actions are named "requirements" and are represented as method prototypes.

Checking whether an agent has the capabilities required by a policy means to check whether it is possible to tie the description of the policy to the execution environment defined by the agent. In the example, *alice* should check if it has the capability of deciding if it is a student and of finding out if it is an EU citizen. In [Baldoni et al., 2006d] it is considered as particularly promising to adopt *semantic matchmaking* techniques based on *ontologies* of concepts. In fact semantic matchmaking supports the matching of capabilities with different names, though connected by an ontology, and with different numbers (and descriptions) of input/output parameters. Semantic matchmaking has been thoroughly studied and formalized also in the Semantic Web community, in particular in the context of the DAML-S [Paolucci et al., 2002] and WSMO initiatives [Keller et al., 2004].

More in general, a capability requirement could be associated, by the matching process, with

complex actions rather than with simple queries of the agent's mental state. In our running example, we can associate by the matching process the capability requirement CR1 in Figure 2 with policy A1, while CR2 will be associated with A2. In this way, we *dynamically obtain a composition* of policy E1 (the one sent by *elearn* to be adopted by *alice*) with policies A1 and A2 of *alice*. After this, it is possible to perform the analysis of the composed policy, for instance, by means of the techniques presented in the other sections. The compositions are ruled by the matching rules. So an agent might perform forms of hypothetical reasoning on the policy to adopt and on its own capabilities by means of different matching rules with the aim of finding the most convenient composition.

Reasoning on capabilities allows to indentify execution paths that involve capabilities that the agent has or accepts to use. Let us take into account a policy and all the capabilities required by it. Each execution trace corresponds to a branch in the policy. It is likely that only a subset of the capabilities associated with a role will be used along a given branch. We could consider only the execution traces concerning the specific call, that the service would like to enact. This set will tell us which capabilities are actually necessary in our execution context (i.e. given the specified input parameter values). In this perspective, it is not compulsory that the service has all the capabilities associated to the policy but it will be sufficient that it has those used in this set of execution traces.

Last but not least, the set of capabilities of a service could be not completely predefined but depending on the context and on privacy or security policies defined by the user: I might have a capability which I do not want to use in that circumstance. In this perspective, it would be interesting to explore the use of the notion of *opportunity* proposed by Padmanabhan et al. [Padmanabhan et al., 2001] in connection with the concept of capability (but with the meaning proposed in [Padgham and Lambrix, 2000]).

More details can be found in

> *Matteo Baldoni, Cristina Baroglio, Alberto Martelli, Viviana Patti, and Claudio Schifanella. Interaction Protocols and Capabilities: A Preliminary Report. In J. J. Alferes, J. Bailey, W. May, and U. Schwertel, editors, Post-Proc. of the Fourth Workshop on Principles and Practice of Semantic Web Reasoning, PPSWR 2006, volume 4187 of LNCS, pages 63-77. Springer, 2006.*

attached in the Appendix.

6 Interoperability

In the previous sections we have implicitly assumed that the considered set of entities can interact by using the policies owned by each of them. However, with reference to policy E1 (see Figure 1), what would happen if *elearn* executes an ask (e.g. ask(*student*(*alice*))) and *alice* does not execute the corresponding get_ask? In other words, what happens if a policy foresees the act of sending a message but the interlocutor's policy does not account for a corresponding receive? In this case the interaction will enter a deadlock state and the transaction will not be completed. This situation compromises the expectation that one has, when representing policies on the base of message exchange, that the interaction will be completed, independently from the outcome (*alice* will or will not get a discount). The policies have to be *interoperable*. It is desirable to prove the interoperablity of a set of entities before their interaction begins.

The problem of verifying interoperability has been tackled, in the literature, using as a reference a given schema of interaction (or interaction protocol). The idea is that the entities will be interoperable if they are conformant to the specification of the various roles in the protocol. In the literature, two kinds of conformance verifications have been studied: *a priori* conformance verification, and *run-time* conformance verification (or compliance) [Endriss et al., 2003, Endriss et al., 2004, Guerin and Pitt, 2003]. 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. When an entity has to decide whether to adopt a policy, it should take a decision *before* beginning the interaction. For this reason a priori conformance is more suitable.

The issue of a priori 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 [Baldoni et al., 2005a, Baldoni et al., 2006a] we have proposed two approaches, the former, based on a trace semantics, consisting in an inclusion test, the latter, disregarding the case of different branching structures. The second technique was also adapted to web services [Baldoni et al., 2005b]. Both works were limited to protocols with only two roles, extensions to protocols with an arbitrary finite number of roles are currently under investigation [Baldoni et al., 2006b]. Inspired to this work is the proposal in [Alberti et al., 2006]: here an abductive framework is used to verify the conformance of services to a choreography with any number of roles. The limit 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.* [Endriss et al., 2004], 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.* [Foster et al., 2006], 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 execution traces can be too much restrictive. In [Busi et al., 2005], 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. A recent proposal, in this same line, is [Zhao et al., 2006], which suffers of the same limitations.

The problem of adopting a priori conformance in the application framework that we are considering is twofold. On the one hand, we do not have a protocol against which verifying the conformance; on the other hand, policies are disclosed little by little. Indeed, in general rather than a protocol the entities interact by means of a set of policies which often do not refer to a same abstract schema of interaction. Our claim is that if the policies were public we could adopt techniques that are analogous to some of those mentioned above or derived from model checking. The question is: can we reasonably expect that such policies are public? Going back to our example, *elearn* is a service aimed at selling e-learning courses. For this reason, besides the selling criteria (e.g. sell only to EU citiziens who are students), it will have all the interest of guaranteeing the interoperability with its clients. In this perspective, we can expect that it will agree on making its interaction policy public, keeping its decisional policy private.

Approaches like [Baldoni et al., 2005a, Baldoni et al., 2006a, Baldoni et al., 2006b] work taking into account the mere message exchange, disregarding (internal) decision criteria. In particular, the technique described in

> *Matteo Baldoni, Cristina Baroglio, Alberto Martelli and Viviana Patti, Verification of protocol conformance and agent interoperability, Post-Proc. of CLIMA VI, Post-Proc. of Sixth International Workshop on Computational Logic in Multi-Agent Systems, CLIMA VI, 2006, vol. 3900 of LNCS State-of-the-Art Survey, 265–283, Springer.*

and attached in the Appendix. could be used to compare the policies of two entities, pretending that they are the two opposite roles of a same protocol.

The fact that entities disclose their policies little by little, when necessary, could be tackled by adopting an alternating approach, in which first the entities analyse the policies disclosed so far and, then, they execute them. The reasoning process is applied whenever an entity has to decide whether to adopt a policy, in the line of what proposed in the previous sections.

A different perspective is that of the approach taken so far in I2. In Protune interoperability is obtained by means of rule exchange, see [Bonatti and Olmedilla, 2005b]. An agent interacts with another by means of the policy that it has received by it. If it attains to such a policy, interoperability is guaranteed, of course provided that the policy itself is interoperable. The policy is adopted "as is". In other words, with reference to the previous section, in Protune the problem of interoperability is solved by adopting both the policy and the definitions of the requested capabilities, which must be included (*ontology sharing*). This solution is analogous to the ones proposed in [Ancona and Mascardi, 2004, S. Costantini, 2005]. CooBDI [Ancona and Mascardi, 2004] extends the BDI (*Belief, Desire, Intention*) model in such a way that agents are enabled to cooperate through a mechanism, which allows them to exchange plans and which is used whenever it is not possible to find a plan, for pursuing a goal of interest, by just exploiting the local agent's knowledge. The ideas behind the CooBDI theory have been implemented by means of WS technologies, leading to CooWS agents [Bozzo et al., 2005]. Another recent work in this line of research is [S. Costantini, 2005]. Here, in the setting of the DALI language, agents can cooperate by exchanging sets of rule that can either define a procedure, or constitute a module for coping with some situation, or be just a segment of a knowledge base. Moreover, agents have reasoning techniques that enable them to evaluate how useful the new information is.

As shown in Section 5, it is often desirable for an agent to use *its own* definitions of capabilities and/or also the actions that are explicitly included in the policy. In this case, the problem of interoperability becomes a matching problem: we need to understand if the actions that the agent would use are compatible with those intended by the interlocutor. We have already seen that it is possible to exploit semantic matchmaking techniques to this aim.

A third situation is when the agent that receives a policy would like to modify it, e.g. by composing it with some policy of its own (see Section 5) or modify it in other ways. For instance, the agent would like to conclude successfully an interaction, disclosing the minimal amount of information that is possible, and to this aim it will modify the received policy. Other situations are those in which the agent does not want to disclose certain information or in which it wants to adopt some interaction/negotiation strategies that it considers as being more profitable for itself. It is important in this context to have the possibility of verifying that by the modified policy it is still possible to interact with the other agent. This verification is to be done by the receiver of the policy itself, which is the only to know how the policy will be modified. The only

knowledge that our agent has about its counterpart, however, is the policy that it has received from it, which, as we have seen, is supposed to guarantee the interoperability. If we had also the policy that is executed by the interlocutor, we could apply a verification technique (e.g. using SPIN) for performing the test again. In general this knowledge is not given, a fortiori in this example where information is disclosed as little as possible. In general, the problem is then to verify if what the agent will execute obeys the operational behavior described by the received (original) policy. In this case the received policy represents a desired schema of interaction, which guarantees the interoperability. Therefore, it rises to the role of *protocol*. In other words, the interoperability problem becomes the problem of verifying that the policy that the agent will execute *conforms to* the desired policy received by the interlocutor. In this direction it is possible to apply the techniques described in [Baldoni et al., 2006a] and in

> *Matteo Baldoni, Cristina Baroglio, Alberto Martelli and Viviana Patti, Conformance and Interoperability in Open Enviroments, Proc. of WOA 2006: Dagli oggetti agli agenti, Catania, Italy, 2006.*

attached in the Appendix. In this work we propose and approach, inspired by (bi)simulation, in which a policy is compared to a role in a protocol, and we focus on those properties that are essential to the verification the interoperability of a set of services. The approach is based on message exchange and on *finite state automata*.

7 Conclusion

A very promising research direction that we mean to explore concerns the study of a coherent framework for a meaningful combination among declarative approaches on policy representation (like the one adopted in Protune) and formal techniques used for testing interoperability (e.g. automata-based approaches) as presented in Section 6. Obviously different representations of behaviors and protocols support different methods for modelling negotiation and for verifying fundamental properties of the interaction like interoperability, conformance, optimal negotiation, and so on. Which are the contexts where verification techniques based on an automata representation of the interaction can be fruitfully applied (and vice versa)? We have started to give some intuitions in the particular case of the application of conformance and interoperability test, but the matter deserves a deeper and more systematic investigation. In order to combine the advantages of both the approaches in a general framework the non-trivial problem of switching from a certain kind of representation to another is to be faced (what does it mean transforming a ruled-based declarative policy in a corresponding automaton?). Moreover, considering the increasing interest of modelling languages for describing the interaction, it would be interesting to include also this aspect in a study of the cross-transformations between different representation of the behavior of interactive peers. Notice that in the literature about web service it, some studies on this matter already exist, in particular for translating representations of WS-CDL choreophraphies in automata [Baldoni et al., 2005b, Foster et al., 2006].

Acknowledgement

This research has been co-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`).

References

[Alberti et al., 2006] Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., and Montali, M. (2006). An abductive framework for a-priori verification of web services. In *Principles and Practice of Declarative Programming, PPDP'06)*. ACM Press.

[Ancona and Mascardi, 2004] Ancona, D. and Mascardi, V. (2004). Coo-BDI: Extending the BDI Model with Cooperativity. In Leite, J. A., Omicini, A., Sterling, L., and Torroni, P., editors, *Proceedings of the First Declarative Agent Languages and Technologies Workshop (DALT'03), Revised Selected and Invited Papers*, pages 109–134. Springer-Verlag. LNAI 2990.

[Baldoni et al., 2006a] Baldoni, M., Baroglio, C., Martelli, A., and Patti (2006a). 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.

[Baldoni et al., 2006b] Baldoni, M., Baroglio, C., Martelli, A., and Patti, V. (2006b). Conformance and interoperability in open enviroments. In De Paoli, F., Omicini, A., and Santoro, C., editors, *Proc. of WOA 2006: Dagli oggetti agli agenti*, Catania, Italy.

[Baldoni et al., 2006c] Baldoni, M., Baroglio, C., Martelli, A., and Patti, V. (2006c). Reasoning about interaction protocols for customizing web service selection and composition. *J. of Logic and Algebraic Programming, special issue on Web Services and Formal Methods*. to appear.

[Baldoni et al., 2005a] Baldoni, M., Baroglio, C., Martelli, A., Patti, V., and Schifanella, C. (2005a). Verifying protocol conformance for logic-based communicating agents. In *Proc. of 5th Int. Workshop on Computational Logic in Multi-Agent Systems, CLIMA V*, number 3487 in LNCS, pages 192–212. Springer.

[Baldoni et al., 2006d] Baldoni, M., Baroglio, C., Martelli, A., Patti, V., and Schifanella, C. (2006d). Interaction protocols and capabilities: a preliminary report. In *Post-Proc. of the Fourth Workshop on Principles and Practice of Semantic Web Reasoning, PPSWR 2006*, LNCS. Springer.

[Baldoni et al., 2005b] Baldoni, M., Baroglio, C., Martelli, A., Patti, V., and Schifanella, C. (September, 2005b). 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.

[Baldoni et al., 2006e] Baldoni, M., Boella, G., and van der Torre, L. (2006e). Bridging Agent Theory and Object Orientation: Importing Social Roles in Object Oriented Languages. In Bordini, R. H., Dastani, M., Dix, J., and Seghrouchni, A., editors, *Post-Proc. of the International Workshop on Programming Multi-Agent Systems, ProMAS 2005*, volume 3862 of *Lecture Notes in Computer Science (LNCS)*, pages 57–75. Springer.

[Baldoni et al., 2006f] Baldoni, M., Boella, G., and van der Torre, L. (2006f). powerjava: Ontologically Founded Roles in Object Oriented Programming Languages. In Ancona, D. and Viroli, M., editors, *Proc. of 21st ACM Symposium on Applied Computing, SAC 2006, Special Track on Object-Oriented Programming Languages and Systems (OOPS 2006)*, Dijon, France. ACM.

[Bonatti and Olmedilla, 2005a] Bonatti, P. A. and Olmedilla, D. (2005a). Driving and monitoring provisional trust negotiation with metapolicies. In *6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, pages 14–23, Stockholm, Sweden. IEEE Computer Society.

[Bonatti and Olmedilla, 2005b] Bonatti, P. A. and Olmedilla, D. (2005b). Policy language specification. Technical Report IST506779/Naples/I2-D2/D/PU/b1, Reasoning on the Web with Rules and Semantics, REWERSE. Available at: http://rewerse.net/deliverables/m12/i2-d2.pdf.

[Bozzo et al., 2005] Bozzo, L., Mascardi, V., Ancona, D., and Busetta, P. (2005). CooWS: Adaptive BDI agents meet service-oriented computing. In *Proceedings of the Int. Conference on WWW/Internet*, pages 205–209.

[Busi et al., 2005] Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., and Zavattaro, G. (2005). Choreography and orchestration: a synergic approach for system design. In *Proc. of 4th International Conference on Service Oriented Computing (ICSOC 2005)*.

[Endriss et al., 2003] Endriss, U., Maudet, N., Sadri, F., and Toni, F. (2003). Protocol conformance for logic-based agents. In Gottlob, G. and Walsh, T., editors, *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003)*, pages 679–684. Morgan Kaufmann Publishers.

[Endriss et al., 2004] Endriss, U., Maudet, N., Sadri, F., and Toni, F. (2004). Logic-based agent communication protocols. In *Advances in agent communication languages*, volume 2922 of *LNAI*, pages 91–107. Springer-Verlag. invited contribution.

[Foster et al., 2006] Foster, H., Uchitel, S., Magee, J., and Kramer, J. (2006). Model-based analysis of obligations in web service choreography. In *Proc. of IEEE International Conference on Internet&Web Applications and Services 2006*.

[Giordano et al., 2004] Giordano, L., Martelli, A., and Schwind, C. (2004). Verifying communicating agents by model checking in a temporal action logic. In Alferes, J. and Leite, J., editors, *Proc. of the 9th European Conference on Logics in Artificial Intelligence (JELIA'04)*, volume 3229 of *Lecture Notes in Computer Science (LNCS)*, pages 57–69. Springer.

[Guerin and Pitt, 2003] Guerin, F. and Pitt, J. (2003). Verification and Compliance Testing. In Huget, H., editor, *Communication in Multiagent Systems*, volume 2650 of *LNAI*, pages 98–112. Springer.

[Jade, 2001] Jade (2001). http://jade.cselt.it/.

[Keller et al., 2004] Keller, U., Polleres, R. L. A., Toma, I., Kifer, M., and Fensel, D. (2004). D5.1 v0.1 wsmo web service discovery. Technical report, WSML deliverable.

[Mamdani and Pitt, 2000] Mamdani, A. and Pitt, J. (2000). Communication protocols in multiagent systems: A development method and reference architecture. In *Issues in Agent Communication*, volume 1916 of *LNCS*, pages 160–177. Springer.

[Padgham and Lambrix, 2000] Padgham, L. and Lambrix, P. (2000). Agent capabilities: Extending BDI theory. In *AAAI/IAAI*, pages 68–73.

[Padmanabhan et al., 2001] Padmanabhan, V., Governatori, G., and Sattar, A. (2001). Actions made explicit in bdi. In *Advances in Artificial Intelligence*, number 2256 in LNCS, pages 390–401. Springer.

[Paolucci et al., 2002] Paolucci, M., Kawmura, T., Payne, T., and Sycara, K. (2002). Semantic matching of web services capabilities. In *First International Semantic Web Conference*.

[S. Costantini, 2005] S. Costantini, A. T. (2005). Learning by knowledge exchange in logical agents. In Corradini, F., De Paoli, F., Merelli, E., and Omicini, A., editors, *Proc. of WOA 2005: Dagli oggetti agli agenti, simulazione e analisi formale di sistemi complessi*, Camerino, Italy. Pitagora Editrice Bologna.

[Zhao et al., 2006] Zhao, X., Yang, H., and Qui, Z. (2006). Towards the formal model and verification of web service choreography description language. In *Proc. of WS-FM 2006*.

A Appendix

The appendix includes five papers. One paper is in press on an international journal, three appeared or are in press in the series Lecture Notes on Computer Science, and one in a workshop:

1. *Laura Giordano, Alberto Martelli, and Camilla Schwind. Verifying communicating agents by model checking in a temporal action logic. In J. Alferes and J. Leite, editors, 9th European Conference on Logics in Artificial Intelligence (JELIA'04), volume 3229 of LNAI, pages 57-69, Lisbon, Portugal, Sept. 2004. Springer-Verlag.*

2. *Matteo Baldoni, Cristina Baroglio, Alberto Martelli, and Viviana Patti. Reasoning about interaction protocols for customizing web service selection and composition. Journal of Logic and Algebraic Programming, special issue on Web Services and Formal Methods, 2006. In press.*

3. *Matteo Baldoni, Cristina Baroglio, Alberto Martelli, Viviana Patti, and Claudio Schifanella. Interaction Protocols and Capabilities: A Preliminary Report. In J. J. Alferes, J. Bailey, W. May, and U. Schwertel, editors, Post-Proc. of the Fourth Workshop on Principles and Practice of Semantic Web Reasoning, PPSWR 2006, volume 4187 of LNCS, pages 63-77. Springer, 2006.*

4. *Matteo Baldoni, Cristina Baroglio, Alberto Martelli and Viviana Patti, Verification of protocol conformance and agent interoperability, Post-Proc. of CLIMA VI, Post-Proc. of Sixth International Workshop on Computational Logic in Multi-Agent Systems, CLIMA VI, 2006, vol. 3900 of LNCS State-of-the-Art Survey, 265–283, Springer.*

5. *Matteo Baldoni, Cristina Baroglio, Alberto Martelli and Viviana Patti, Conformance and Interoperability in Open Enviroments, Proc. of WOA 2006: Dagli oggetti agli agenti, Catania, Italy, 2006.*

Verifying Communicating Agents by Model Checking in a Temporal Action Logic *

Laura Giordano[1], Alberto Martelli[2], Camilla Schwind[3]

[1]Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria
[2]Dipartimento di Informatica, Università di Torino, Torino
[3]MAP, CNRS, Marseille, France

Abstract. In this paper we address the problem of specifying and verifying systems of communicating agents in a Dynamic Linear Time Temporal Logic (DLTL). This logic provides a simple formalization of the communicative actions in terms of their effects and preconditions. Furthermore it allows to specify interaction protocols by means of temporal constraints representing permissions and commitments. Agent programs, when known, can be formulated in DLTL as complex actions (regular programs). The paper addresses several kinds of verification problems including the problem of compliance of agents to the protocol, and describes how they can be solved by model checking in DLTL using automata.

1 Introduction

The specification and the verification of the behavior of interacting agents is one of the central issues in the area of multi-agent systems. In this paper we address the problem of specifying and verifying systems of communicating agents in a Dynamic Linear Time Temporal Logic (DLTL).

The extensive use of temporal logics in the specification and verification of distributed systems has led to the development of many techniques and tools for automating the verification task. Recently, temporal logics have gained attention in the area of reasoning about actions and planning [2, 10, 12, 17, 5], and they have also been used in the specification and in the verification of systems of communicating agents. In particular, in [21] agents are written in MABLE, an imperative programming language, and the formal claims about the system are expressed using a quantified linear time temporal BDI logic and can be automatically verified by making use of the SPIN model checker. Guerin in [13] defines an agent communication framework which gives agent communication a grounded declarative semantics. In such a framework, temporal logic is used for formalizing temporal properties of the system.

In this paper we present a theory for reasoning about communicative actions in a multiagent system which is based on the Dynamic Linear Time Temporal

* This research has been partially supported by the project PRIN 2003 "Logic-based development and verification of multi-agent systems", and by the European Commission within the 6th Framework Programme project REWERSE number 506779

Logic ($DLTL$) [15], which extends LTL by strengthening the *until* operator by indexing it with the regular programs of dynamic logic. As a difference with [21] we adopt a social approach to agent communication [1, 7, 19, 13], in which communicative actions affect the "social state" of the system, rather than the internal (mental) states of the agents. The social state records social facts, like the permissions and the commitments of the agents. The dynamics of the system emerges from the interactions of the agents, which must respect these permissions and commitments (if they are compliant with the protocol). The social approach allows a high level specification of the protocol, and does not require the rigid specification of the allowed action sequences. It is well suited for dealing with "open" multiagent systems, where the history of communications is observable, but the internal states of the single agents may not be observable.

Our proposal relies on the theory for reasoning about action developed in [10] which is based on DLTL and which allows reasoning with incomplete initial states and dealing with postdiction, ramifications as well as with nondeterministic actions. It allows a simple formalization of the communicative actions in terms of their effects and preconditions as well as the specification of an interaction protocol to constrain the behaviors of autonomous agents.

In [11] we have presented a proposal for reasoning about communicating agents in the Product Version of DLTL, which allows to describe the behavior of a network of sequential agents which coordinate their activities by performing common actions together. Here we focus on the non-product version of DLTL, which appears to be a simpler choice and also a more reasonable choice when a social approach is adopted. In fact, the Product Version of DLTL does not allow to describe global properties of a system of agents, as it keeps the local states of the agents separate. Instead, the "social state" of the system is inherently global and shared by all of the agents. Moreover, we will see that the verification tasks described in [11] can be conveniently represented in DLTL without requiring the product version. The verification of the compliance of an agent to the protocol, the verification of protocol properties, the verification that an agent is (is not) respecting its social facts (commitments and permissions) at runtime are all examples of tasks which can be formalized either as validity or as satisfiability problems in DLTL. Such verification tasks can be automated by making use of Büchi automata. In particular, we make use of the tableau-based algorithm presented in [9] for constructing a Büchi automaton from a DLTL formula. The construction of the automata can be done on-the-fly, while checking for the emptiness of the language accepted by the automaton. As for LTL, the number of states of the automata is, in the worst case, exponential in the size of the input formula.

2 Dynamic Linear Time Temporal Logic

In this section we shortly define the syntax and semantics of DLTL as introduced in [15]. In such a linear time temporal logic the next state modality is indexed

by actions. Moreover, (and this is the extension to LTL) the until operator is indexed by programs in Propositional Dynamic Logic (PDL).

Let Σ be a finite non-empty alphabet. The members of Σ are actions. Let Σ^* and Σ^ω be the set of finite and infinite words on Σ, where $\omega = \{0, 1, 2, \ldots\}$. Let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We denote by σ, σ' the words over Σ^ω and by τ, τ' the words over Σ^*. Moreover, we denote by $\leq$ the usual prefix ordering over Σ^* and, for $u \in \Sigma^\infty$, we denote by *prf(u)* the set of finite prefixes of u.

We define the set of programs (regular expressions) $Prg(\Sigma)$ generated by Σ as follows:

$$Prg(\Sigma) ::= a \mid \pi_1 + \pi_2 \mid \pi_1; \pi_2 \mid \pi^*$$

where $a \in \Sigma$ and π_1, π_2, π range over $Prg(\Sigma)$. A set of finite words is associated with each program by the mapping $[[]] : Prg(\Sigma) \to 2^{\Sigma^*}$, which is defined as follows:

- $[[a]] = \{a\}$;
- $[[\pi_1 + \pi_2]] = [[\pi_1]] \cup [[\pi_2]]$;
- $[[\pi_1; \pi_2]] = \{\tau_1\tau_2 \mid \tau_1 \in [[\pi_1]] \text{ and } \tau_2 \in [[\pi_2]]\}$;
- $[[\pi^*]] = \bigcup [[\pi^i]]$, where
  - $[[\pi^0]] = \{\varepsilon\}$
  - $[[\pi^{i+1}]] = \{\tau_1\tau_2 \mid \tau_1 \in [[\pi]] \text{ and } \tau_2 \in [[\pi^i]]\}$, for every $i \in \omega$.

Let $\mathcal{P} = \{p_1, p_2, \ldots\}$ be a countable set of atomic propositions. The set of formulas of DLTL(Σ) is defined as follows:

$$\text{DLTL}(\Sigma) ::= p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \mathcal{U}^\pi \beta$$

where $p \in \mathcal{P}$ and α, β range over DLTL(Σ).

A model of DLTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^\omega$ and $V : prf(\sigma) \to 2^{\mathcal{P}}$ is a valuation function. Given a model $M = (\sigma, V)$, a finite word $\tau \in prf(\sigma)$ and a formula α, the satisfiability of a formula α at τ in M, written $M, \tau \models \alpha$, is defined as follows:

- $M, \tau \models p$ iff $p \in V(\tau)$;
- $M, \tau \models \neg\alpha$ iff $M, \tau \not\models \alpha$;
- $M, \tau \models \alpha \vee \beta$ iff $M, \tau \models \alpha$ or $M, \tau \models \beta$;
- $M, \tau \models \alpha \mathcal{U}^\pi \beta$ iff there exists $\tau' \in [[\pi]]$ such that $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \beta$. Moreover, for every τ'' such that $\varepsilon \leq \tau'' < \tau'$[1], $M, \tau\tau'' \models \alpha$.

A formula α is satisfiable iff there is a model $M = (\sigma, V)$ and a finite word $\tau \in prf(\sigma)$ such that $M, \tau \models \alpha$.

The formula $\alpha \mathcal{U}^\pi \beta$ is true at τ if "α until β" is true on a finite stretch of behavior which is in the linear time behavior of the program π.

The derived modalities $\langle\pi\rangle$ and $[\pi]$ can be defined as follows: $\langle\pi\rangle\alpha \equiv \top \mathcal{U}^\pi \alpha$ and $[\pi]\alpha \equiv \neg\langle\pi\rangle\neg\alpha$.

[1] We define $\tau \leq \tau'$ iff $\exists\tau''$ such that $\tau\tau'' = \tau'$. Moreover, $\tau < \tau'$ iff $\tau \leq \tau'$ and $\tau \neq \tau'$.

Furthermore, if we let $\Sigma = \{a_1, \ldots, a_n\}$, the $\mathcal{U}$, O (next), $\Diamond$ and $\Box$ operators of LTL can be defined as follows: $O\alpha \equiv \bigvee_{a \in \Sigma} \langle a \rangle \alpha$, $\alpha \mathcal{U} \beta \equiv \alpha \mathcal{U}^{\Sigma^*} \beta$, $\Diamond \alpha \equiv \top \mathcal{U} \alpha$, $\Box \alpha \equiv \neg \Diamond \neg \alpha$, where, in $\mathcal{U}^{\Sigma^*}$, Σ is taken to be a shorthand for the program $a_1 + \ldots + a_n$. Hence both LTL(Σ) and PDL are fragments of DLTL(Σ). As shown in [15], DLTL(Σ) is strictly more expressive than LTL(Σ). In fact, DLTL has the full expressive power of the monadic second order theory of ω-sequences.

3 Action theories

In this section we recall the action theory developed in [10] that we use for specifying the interaction between communicating agents.

Let $\mathcal{P}$ be a set of atomic propositions, the *fluent names*. A *fluent literal* l is a fluent name f or its negation $\neg f$. Given a fluent literal l, such that $l = f$ or $l = \neg f$, we define $|l| = f$. We will denote by *Lit* the set of all fluent literals.

A *domain description* D is defined as a tuple $(\Pi, \mathcal{C})$, where Π is a set of *action laws* and *causal laws*, and $\mathcal{C}$ is a set of *constraints*.

Action laws in Π have the form: $\Box(\alpha \rightarrow [a]\beta)$, with $a \in \Sigma$ and α, β arbitrary formulas, meaning that executing action a in a state where precondition α holds causes the effect β to hold.

Causal laws in Π have the form: $\Box((\alpha \wedge \bigcirc \beta) \rightarrow \bigcirc \gamma)$, meaning that if α holds in a state and β holds in the next state, then γ also holds in the next state. Such laws are intended to expresses "causal" dependencies among fluents.

Constraints in $\mathcal{C}$ are arbitrary temporal formulas of $DLTL$. In particular, the set of constraints $\mathcal{C}$ contains all the temporal formulas which might be needed to constrain the behaviour of a protocol, including the value of fluents in the initial state. The set of constraints $\mathcal{C}$ also includes the precondition laws.

Precondition laws have the form: $\Box(\alpha \rightarrow [a]\bot)$, meaning that the execution of an action a is not possible if α holds (i.e. there is no resulting state following the execution of a if α holds). Observe that, when there is no precondition law for an action, the action is executable in all states.

Action laws and causal laws describe the changes to the state. All other fluents which are not changed by the actions are assumed to persist unaltered to the next state. To cope with the *frame problem*, the laws in Π, describing the (immediate and ramification) effects of actions, have to be distinguished from the constraints in $\mathcal{C}$ and given a special treatment. In [10], to deal with the frame problem, a completion construction is defined which, given a domain description, introduces frame axioms for all the frame fluents in the style of the successor state axioms introduced by Reiter [18] in the context of the situation calculus. The completion construction is applied only to the action laws and causal laws in Π and not to the constraints. In the following we call $Comp(\Pi)$ the completion of a set of laws Π and we refer to [10] for the details on the completion construction.

Test actions allow the choice among different behaviours to be controlled. As $DLTL$ does not include test actions, we introduce them in the language as atomic actions in the same way as done in [10]. More precisely, we introduce

an atomic action $\phi?$ for each proposition ϕ we want to test. The test action $\phi?$ is executable in any state in which ϕ holds and it has no effect on the state. Therefore, we introduce the following laws which rule the modality $[\phi?]$:

$\Box(\neg\phi \rightarrow [\phi?]\bot)$
$\Box(\langle\phi?\rangle\top \rightarrow (L \leftrightarrow [\phi?]L))$, for all fluent literals L.

The first law is a precondition law, saying that action $\phi?$ is only executable in a state in which ϕ holds. The second law describes the effects of the action on the state: the execution of the action $\phi?$ leaves the state unchanged. We assume that, for all test actions occurring in a domain description, the corresponding action laws are implicitly added.

As a difference from [10], in this paper we will use, besides boolean fluents, *functional fluents*, i.e. fluents which take a value in a (finite) set. We use the notation $f = V$ to say that fluent f has value V. It is clear however, that functional fluents can be easily represented by making use of multiple (and mutually exclusive) boolean fluents.

4 Contract Net Protocol

In the social approach [7, 13, 19, 22] an interaction protocol is specified by describing the effects of communicative actions on the social state, and by specifying the permissions and the commitments that arise as a result of the current conversation state. In our action theory the effects of communicative actions will be modelled by *action laws*. Permissions, which determine when an action can be taken by each agent, can be modelled by *precondition laws*. Commitment policies, which rule the dynamic of commitments, can be described by *causal laws* which establish the causal dependencies among fluents. The specification of a protocol can be further constrained through the addition of suitable *temporal formulas*, and also the agents' programs can be modelled, by making use of complex actions (regular programs).

As a running example we will use the Contract Net protocol [6].

Example 1. The Contract Net protocol begins with an agent (the manager) broadcasting a task announcement (call for proposals) to other agents viewed as potential contractors (the participants). Each participant can reply by sending either a proposal or a refusal. The manager must send an accept or reject message to all those who sent a proposal. When a contractor receives an acceptance it is committed to perform the task. For lack of space we will leave out the final step of the protocol.

Let us consider first the simplest case where we have only two agents: the manager (M) and the participant (P). The two agents share all the communicative actions, which are: *cfp(T)* (the manager issues a call for proposals for task T), *accept* and *reject* whose sender is the manager, and *refuse* and *propose* whose sender is the participant.

The social state will contain the following domain specific fluents: *task* (a functional fluent whose value is the task which has been announced, or *nil* if the task has not yet been announced), *replied* (the participant has replied), *proposal* (the participant has sent a proposal) and *acc_rej* (the manager has sent an accept or reject message). Such fluents describe observable facts concerning the execution of the protocol.

We also introduce special fluents to represent *base-level commitments* of the form $C(i, j, \alpha)$, meaning that agent i is committed to agent j to bring about α, where α is an arbitrary formula, or they can be *conditional commitments* of the form $CC(i, j, \beta, \alpha)$ (agent i is committed to agent j to bring about α, if the condition β is brought about)[2]. For modelling the Contract Net example we introduce the following commitments

$C(P, M, replied)$ and $C(M, P, acc\_rej)$

and conditional commitments

$CC(P, M, task \neq nil, replied)$ and $CC(M, P, proposal, acc\_rej)$.

Some reasoning rules have to be defined for cancelling commitments when they have been fulfilled and for dealing with conditional commitments. We introduce the following *causal laws*:

$\Box(\bigcirc\alpha \rightarrow \bigcirc\neg C(i, j, \alpha))$
$\Box(\bigcirc\alpha \rightarrow \bigcirc\neg CC(i, j, \beta, \alpha))$
$\Box((CC(i, j, \beta, \alpha) \wedge \bigcirc\beta) \rightarrow \bigcirc(C(i, j, \alpha) \wedge \neg CC(i, j, \beta, \alpha)))$

A commitment (or a conditional commitment) to bring about α is cancelled when α holds, and a conditional commitment $CC(i, j, \beta, \alpha)$ becomes a base-level commitment $C(i, j, \alpha)$ when β has been brought about.

Let us now describe the effects of communicative actions by the following *action laws*:

$\Box[cfp(T)]task = T$
$\Box[cfp(T)]CC(M, P, proposal, acc\_rej)$
$\Box[accept]acc\_rej$
$\Box[reject]acc\_rej$
$\Box[refuse]replied$
$\Box[propose](replied \wedge proposal)$

The laws for action $cfp(T)$ add to the social state the information that a call for proposal has been done for the task T, and that, if the manager receives a proposal, it is committed to accept or reject it.

The permissions to execute communicative actions in each state are determined by social facts. We represent them by precondition laws. Preconditions on the execution of action *accept* can be expressed as:

[2] The two kinds of base-level and conditional commitments we allow are essentially those introduced in [22]. Such choice is different from the one in [13] and in [11], where agents are committed to execute an action rather than to achieve a condition.

$$\Box(\neg proposal \vee acc\_rej \rightarrow [accept]\bot)$$

meaning that action *accept* cannot be executed only if a proposal has not been done, or if the manager has already replied. Similarly we can give the *precondition laws* for the other actions:

$$\Box(\neg proposal \vee acc\_rej \rightarrow [reject]\bot)$$
$$\Box(task = nil \vee replied \rightarrow [refuse]\bot)$$
$$\Box(task = nil \vee replied \rightarrow [propose]\bot)$$
$$\Box(task \neq nil \rightarrow [cfp(T)]\bot).$$

The precondition law for action *propose* ($refuse$) says that a proposal can only be done if $task \neq nil$, that is, if a task has already been announced and the participant has not already replied. The last law says that the manager cannot issue a new call for proposal if $task \neq nil$, that is, if a task has already been announced.

In the following we will denote $Perm_i$ (permissions of agent i) the set of all the precondition laws of the protocol pertaining to the actions of which agent i is the sender.

Assume now that we want the participant to be committed to reply to the task announcement. We can express it by adding the following conditional commitment to the initial state of the protocol: $CC(P, M, task \neq nil, replied)$. Furthermore the manager is committed initially to issue a call for proposal for a task. We can define the *initial state Init* of the protocol as follows:

$$\{task = nil, \neg replied, \neg proposal, CC(P, M, task \neq nil, replied),$$
$$C(M, P, task \neq nil)\}$$

In the following we will be interested in those execution of the protocol in which all commitments have been fulfilled. We can express the condition that the commitment $C(i, j, \alpha)$ will be fulfilled by the following constraint:

$$\Box(C(i, j, \alpha) \rightarrow \Diamond\alpha)$$

We will call Com_i the set of constraints of this kind for all commitments of agent i. Com_i states that agent i will fulfill all the commitments of which he is the debtor.

Given the above rules, the domain description $D = (\Pi, \mathcal{C})$ of a protocol is defined as follows: Π is the set of the action and causal laws given above, and $\mathcal{C} = Init \wedge \bigwedge_i (Perm_i \wedge Com_i)$ is the set containing the constraints on the initial state, the permissions $Perm_i$ and the commitments Com_i of all the agents (the agents P and M, in this example).

Given a domain description D, let the completed domain description $Comp(D)$ be the set of formulas $(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i))$. The runs of the system according the protocol are the linear models of $Comp(D)$. Observe that in these protocol runs all permissions and commitments have been fulfilled. However, if Com_j is not included for some agent j, the runs may contain commitments which have not been fulfilled by j.

5 Verification

Given the DLTL specification of a protocol by a domain description, we describe the different kinds of verification problems which can be addressed.

First, given an execution history describing the interactions of the agents, we want to verify the compliance of that execution to the protocol. This verification is carried out at runtime. We are given a history $\tau = a_1, \ldots, a_n$ of the communicative actions executed by the agents, and we want to verify that the history τ is the prefix of a run of the protocol, that is, it respects the permissions and commitments of the protocol. This problem can be formalized by requiring that the formula

$$(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i)) \wedge < a_1; a_2; \ldots; a_n > \top$$

(where i ranges on all the agents involved in the protocol) is satisfiable. In fact, the above formula is satisfiable if it is possible to find a run of the protocol starting with the action sequence $a_1, \ldots, a_n$.

A second problem is that of proving a property φ of a protocol. This can be formulated as the validity of the formula

$$(Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i)) \rightarrow \varphi. \tag{1}$$

Observe that, to prove the property φ, all the agents are assumed to be compliant with the protocol.

A further problem is to verify that an agent is compliant with the protocol, given the program executed by the agent itself. In our formalism we can specify the behavior of an agent by making use of complex actions (regular programs). Consider for instance the following program π_P for the participant:

$$[\neg done?; ((cfp(T); eval\_task; (\neg ok?; refuse; exit + ok?; propose)) + (reject; exit) + (accept; do\_task; exit))]^*; done?$$

The participant cycles and reacts to the messages received by the manager: for instance, if the manager has issued a call for proposal, the participant can either refuse or make a proposal according to his evaluation of the task; if the manager has accepted the proposal, the participant performs the task; and so on.

The state of the agent is obtained by adding to the fluents of the protocol the following local fluents: *done*, which is initially false and is made true by action *exit*, and *ok* which says if the agent must make a bid or not. The local actions are *eval_task*, which evaluates the task and sets the fluent *ok* to true or false, *do_task* and *exit*. Furthermore, *done*? and *ok*? are test actions.

The program of the contractor can be specified by a domain description $Prog_P = (\Pi_P, \mathcal{C}_P)$, where Π_P is a set of action laws describing the effects of the private actions of the contractor, for instance:

$\Box[exit]done$
$\Box(task = t1 \rightarrow [eval\_task]ok)$
$\Box(task = t2 \rightarrow [eval\_task]\neg ok)$

and, $\mathcal{C}_P = \{\langle\pi_P\rangle\top, \neg done, \neg ok\}$ contains the constraints on the initial values of fluents $(\neg done, \neg ok)$ as well as the formula $\langle\pi_P\rangle\top$ stating that the program of the participant is executable in the initial state.

We want now to prove that the participant is compliant with the protocol, i.e. that all executions of program π_P satisfy the specification of the protocol. This property cannot be proved by considering only the program π_P. In fact, it is easy to see that the correctness of the property depends on the behavior of the manager. For instance, if the manager begins with an *accept* action, the participant will execute the sequence of actions $accept; do\_task; exit$ and stop, which is not a correct execution of the protocol. Thus we have to take into account also the behavior of the manager. Since we don't know its internal behavior, we will assume that the manager respects its public behavior, i.e. that it respects its permissions and commitments in the protocol specification.

The verification that the participant is compliant with the protocol can be formalized as a validity check. Let $D = (\Pi, \mathcal{C})$ be the domain description describing the protocol, as defined above. The formula

$$(Comp(\Pi) \wedge Init \wedge Perm_M \wedge Com_M \wedge Comp(\Pi_P) \wedge \mathcal{C}_P) \rightarrow (Perm_P \wedge Com_P)$$

is valid if in all the behaviors of the system, in which the participant executes its program π_P and the manager (whose internal program is unknown) respects the protocol specification (in particular, its permissions and commitments), the permissions and commitment of the participant are also satisfied.

6 Contract Net with N participants

Let us assume now that we have N potential contractors. The above formulation of the protocol can be extended by introducing a fluent $replied(i)$, $proposal(i)$ and $acc\_rej(i)$ for each participant i, and similarly for the commitments. Furthermore we introduce the communicative actions $refuse(i)$, and $propose(i)$, which are sent from participant i to the manager, and $reject(i)$ and $accept(i)$, which are sent from the manager to participant i. We assume action $cfp(T)$ to be shared by all agents (broadcast by the manager).

The theory describing the new version of the protocol can be easily obtained from the one given above. For instance the precondition laws for $accept(i)$ and $reject(i)$ must be modified so that these actions will be executed only after all participants have replied to the manager, i.e.:

$$\Box((\neg proposal(i) \vee acc\_rej(i) \vee \bigvee_{j=1,N} \neg replied(j)) \rightarrow [accept(i)]\bot)$$

and the same for $reject(i)$.

The verification problems mentioned before can be formulated using the same approach. For instance, the verification that the protocol satisfies a given property φ can be expressed as the validity of the formula (1) above, where i ranges

over the N participants and the manager. To prove compliance of a participant with the protocol, we need to restrict the protocol to the actions and the fluents shared between the manager and this participant (i.e. we need to take the projection of the protocol on agent i and the agents with whom i interacts). Then the problem can be formulated as in the case of a single participant.

We have not considered here the formulation of the problem in which proposals must be submitted within a given deadline. This would require adding to the system a further agent *clock*.

7 Model checking

The above verification and satisfiability problems can be solved by extending the standard approach for verification and model-checking of Linear Time Temporal Logic, based on the use of Büchi automata. As described in [15], the satisfiability problem for DLTL can be solved in deterministic exponential time, as for LTL, by constructing for each formula $\alpha \in DLTL(\Sigma)$ a Büchi automaton $\mathcal{B}_\alpha$ such that the language of ω-words accepted by $\mathcal{B}_\alpha$ is non-empty if and only if α is satisfiable. Actually a stronger property holds, since there is a one to one correspondence between models of the formula and infinite words accepted by $\mathcal{B}_\alpha$. The size of the automaton can be exponential in the size of α, while emptiness can be detected in a time linear in the size of the automaton.

The validity of a formula α can be verified by constructing the Büchi automaton $\mathcal{B}_{\neg\alpha}$ for $\neg\alpha$: if the language accepted by $\mathcal{B}_{\neg\alpha}$ is empty, then α is valid, whereas any infinite word accepted by $\mathcal{B}_{\neg\alpha}$ provides a counterexample to the validity of α.

For instance, let CN be the completed domain description of the Contract Net protocol, that is $CN = (Comp(\Pi) \wedge Init \wedge \bigwedge_i (Perm_i \wedge Com_i))$. Then every infinite word accepted by $\mathcal{B}_{CN}$ corresponds to a possible execution of the protocol. To prove a property φ of the protocol, we can build the automaton $\mathcal{B}_{\neg\varphi}$ and check that the language accepted by the product of $\mathcal{B}_{CN}$ and $\mathcal{B}_{\neg\varphi}$ is empty.

The construction given in [15] is highly inefficient since it requires to build an automaton with an exponential number of states, most of which will not be reachable from the initial state. A more efficient approach for constructing a Büchi automaton from a DLTL formula makes use of a tableau-based algorithm [9]. The construction of the states of the automaton is similar to the standard construction for LTL [8], but the possibility of indexing until formulas with regular programs puts stronger constraints on the fulfillment of until formulas than in LTL, requiring more complex acceptance conditions. The construction of the automaton can be done on-the-fly, while checking for the emptiness of the language accepted by the automaton. Given a formula φ, the algorithm builds a graph $\mathcal{G}(\varphi)$ whose nodes are labelled by sets of formulas. States and transitions of the Büchi automaton correspond to nodes and arcs of the graph. The algorithm makes use of an auxiliary tableau-based function which expands the set of formulas at each node. As for LTL, the number of states of the automaton is,

in the worst case, exponential in the size if the input formula, but in practice it is much smaller. For instance, the automaton obtained from the Contract Net protocol has about 20 states.

LTL is widely used to prove properties of (possibly concurrent) programs by means of *model checking* techniques. The property is represented as an LTL formula φ, whereas the program generates a Kripke structure (the model), which directly corresponds to a Büchi automaton where all the states are accepting, and which describes all possible computations of the program. The property can be proved as before by taking the product of the model and of the automaton derived from $\neg\varphi$, and by checking for emptiness of the accepted language.

In principle, with DLTL we do not need to use model checking, because programs and domain descriptions can be represented in the logic itself, as we have shown in the previous section. However representing everything as a logical formula can be rather inefficient from a computational point of view. In particular all formulas of the domain description are universally quantified, and this means that our algorithm will have to propagate them from each state to the next one, and to expand them with the tableau procedure at each step.

Therefore we have adapted model checking to the proof of the formulas given in the previous section, as follows. Let us assume that the negation of a formula to be proved can be represented as $F \wedge \varphi$, where $F = Comp(\Pi) \wedge Init$ contains the completion of the action and causal laws in the domain description and the initial state, and φ the rest of the formula. For instance, in the verification of the compliance of the participant, the negation of the formula to be proved is $(Comp(\Pi) \wedge Init \wedge Perm_M \wedge Com_M \wedge Comp(\Pi_P) \wedge \neg(Perm_P \wedge Com_P))$ and thus $\varphi = (Perm_M \wedge Com_M \wedge Comp(\Pi_P) \wedge \neg(Perm_P \wedge Com_P))$. We can derive from F an automaton describing all possible computations, whose states are sets of fluents, which we consider as the model. In particular, we can obtain from the domain description a function $trans_a(S)$, for each action a, for transforming a state in the next one, and then build this automaton by repeatedly applying these functions starting from the initial state. We can then proceed by taking the product of the model and of the automaton derived from φ, and by checking for emptiness of the accepted language.

Note that, although this automaton has an exponential number of states, we can build it step by step by following the construction of the algorithm on-the-fly. The state of the product automaton will consist of two parts $< S_1, S_2 >$, where S_1 is a set of fluents representing a state of the model, and S_2 is a set of formulas. The initial state will be $< I, \varphi >$, where I is the initial set of fluents. A successor state through a transition a will be obtained as $< trans_a(S_1), S_2' >$ where S_2' is derived from S_2 by the on-the-fly algorithm. If the two parts of a state are inconsistent, the state is discarded.

8 Conclusions

We have shown that DLTL is a suitable formalism for specifying and verifying a system of communicating agents. Our approach provides a unified framework for

describing different aspects of multi-agent systems. Programs are expressed as regular expressions, (communicative) actions can be specified by means of action and precondition laws, properties of social facts can be specified by means of causal laws and constraints, and temporal properties can be expressed by means of the *until* operator. We have addressed several kinds of verification problems, including the problem of compliance of agents to the protocol, and described how they can be solved by developing automata-based model checking techniques for DLTL. A preliminary implementation of a model checker based on the algorithm in [9] is being tested in the verification of the properties of various protocols.

The issue of developing semantics for agent communication languages has been examined in [20], by considering in particular the problem of giving a *verifiable* semantics, i.e. a semantics *grounded* on the computational models. Guerin and Pitt [13, 14] define an agent communication framework which gives agent communication a grounded declarative semantics. The framework introduces different languages: a language for agent programming, a language for specifying agent communication and social facts, and a language for expressing temporal properties. Our approach instead provides a unified framework for describing multiagent systems using DLTL.

While in this paper we follow a social approach to the specification and verification of systems of communicating agents, [4, 3, 16, 21] have adopted a mentalistic approach. The goal of [3] is to extend model checking to make it applicable to multi-agent systems, where agents have BDI attitudes. This is achieved by using a new logic which is the composition of two logics, one formalizing temporal evolution and the other formalizing BDI attitudes. In [16, 21] agents are written in MABLE, an imperative programming language, and have a mental state. MABLE systems may be augmented by the addition of formal claims about the system, expressed using a quantified, linear time temporal BDI logic. Instead [4] deals with programs written in AgentSpeak(F), a variation of the BDI logic programming language AgentSpeak(L). Properties of MABLE or AgentSpeak programs can be verified by means of the SPIN model checker. These papers do not deal with the problem of proving properties of protocols.

Yolum and Singh [22] developed a social approach to protocol specification and execution. In this approach, commitments are formalized in a variant of event calculus. By using an event calculus planner it is possible to determine execution paths that respect the protocol specification. Alberti et al. address a similar problem, by expressing protocols in a logic-based formalism based on Social Integrity Constraints. In [1] they present a system that, during the evolution of a society of agents, verifies the compliance of the agents' behavior to the protocol.

References

1. M. Alberti, D. Daolio and P. Torroni. Specification and Verification of Agent Interaction Protocols in a Logic-based System. *SAC'04*, March 2004.
2. F. Bacchus and F. Kabanza. Planning for temporally extended goals. in *Annals of Mathematics and AI*, 22:5–27, 1998.

3. M. Benerecetti, F. Giunchiglia and L. Serafini. Model Checking Multiagent Systems. *Journal of Logic and Computation*. Special Issue on Computational Aspects of Multi-Agent Systems, 8(3):401-423. 1998.
4. R. Bordini, M. Fisher, C. Pardavila and M. Wooldridge. Model Checking AgentSpeak. *AAMAS 2003*, pp. 409–416, 2003.
5. D. Calvanese, G. De Giacomo and M.Y.Vardi. Reasoning about Actions and Planning in LTL Action Theories. In Proc. *KR'02*, 2002.
6. FIPA Contract Net Interaction Protocol Specification, 2002. Available at `http://www.fipa.org`.
7. N. Fornara and M. Colombetti. Defining Interaction Protocols using a Commitment-based Agent Communication Language. *Proc. AAMAS'03*, Melbourne, pp. 520–527, 2003.
8. R. Gerth, D. Peled, M.Y.Vardi and P. Wolper. Simple On-the-fly Automatic verification of Linear Temporal Logic. In *Proc. 15th Work. Protocol Specification, Testing and Verification*, Warsaw, June 1995, North Holland.
9. L. Giordano and A. Martelli. On-the-fly Automata Construction for Dynamic Linear Time Temporal Logic. *TIME 04*, June 2004.
10. L. Giordano, A. Martelli, and C. Schwind. Reasoning About Actions in Dynamic Linear Time Temporal Logic. In FAPR'00 - Int. Conf. on Pure and Applied Practical Reasoning, London, September 2000. Also in *The Logic Journal of the IGPL*, Vol. 9, No. 2, pp. 289-303, March 2001.
11. L. Giordano, A. Martelli, and C. Schwind. Specifying and Verifying Systems of Communicating Agents in a Temporal Action Logic. In *Proc. AI*IA'03*, Pisa, pp. 262–274, Springer LNAI 2829, September 2003.
12. F. Giunchiglia and P. Traverso. Planning as Model Checking. In *Proc. The 5th European Conf. on Planning (ECP'99)*, pp.1–20, Durham (UK), 1999.
13. F. Guerin. Specifying Agent Communication Languages. PhD Thesis, Imperial College, London, April 2002.
14. F. Guerin and J. Pitt. Verification and Compliance Testing. *Communications in Multiagent Systems*, Springer LNAI 2650, pp. 98–112, 2003.
15. J.G. Henriksen and P.S. Thiagarajan. Dynamic Linear Time Temporal Logic. in *Annals of Pure and Applied logic*, vol.96, n.1-3, pp.187–207, 1999
16. M.P. Huget and M. Wooldridge. Model Checking for ACL Compliance Verification. *ACL 2003*, Springer LNCS 2922, pp. 75–90, 2003.
17. M.Pistore and P.Traverso. Planning as Model Checking for Extended Goals in Non-deterministic Domains. Proc. IJCAI'01, Seattle, pp.479-484, 2001.
18. R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, V. Lifschitz, ed.,pages 359–380, Academic Press, 1991.
19. M. P. Singh. A social semantics for Agent Communication Languages. In *IJCAI-98 Workshop on Agent Communication Languages*, Springer, Berlin, 2000.
20. M. Wooldridge. Semantic Issues in the Verification of Agent Communication Languages. *Autonomous Agents and Multi-Agent Systems*, vol. 3, pp. 9-31, 2000.
21. M. Wooldridge, M. Fisher, M.P. Huget and S. Parsons. Model Checking Multi-Agent Systems with MABLE. In *AAMAS'02*, pp. 952–959, Bologna, Italy, 2002.
22. P. Yolum and M.P. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. In *AAMAS'02*, pp. 527–534, Bologna, Italy, 2002.

Reasoning about interaction protocols for customizing web service selection and composition [1]

Matteo Baldoni, Cristina Baroglio, Alberto Martelli, Viviana Patti

Dipartimento di Informatica — Università degli Studi di Torino,
corso Svizzera 185, 10149, Torino, Italy
E-mail: {baldoni,baroglio,mrt,patti}@di.unito.it

Abstract

This work faces the problem of automatic selection and composition of web services, discussing the advantages that derive from the inclusion, in a web service declarative description, of the high-level communication protocol, that is used by the service for interacting with its partners, allowing a rational inspection of it. The approach we propose is set in the Semantic Web field of research and inherits from research in the field of multi-agent systems. Web services are viewed as software agents, communicating by predefined sharable interaction protocols. A logic programming framework based on modal logic is proposed, where the protocol-based interactions of web services are formalized and the use of reasoning about actions and change techniques (planning) for performing the tasks of selection and composition of web services in a way that is personalized w.r.t. the user request is enabled. We claim that applying reasoning techniques on a declarative specification of the service interactions allows to gain flexibility in fulfilling the user preference in the context of a web service matchmaking process.

Key words: Semantic web-services, reasoning about actions, interaction protocols, personalization, agent logic programming

1 Introduction

Web services are an emergent paradigm for implementing business collaborations, across and within corporation boundaries [1]. Workflow research and technology

[1] 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.

have found in web services a natural field of application which opens interesting and challenging perspectives. Web services, however, also raised the attention of other research communities, in particular the two respectively studying the Semantic Web and multi-agent systems (MAS for short). The work presented in this paper is set transversely across these three fields. The aim is, basically, to show with a practical example the possibility and the benefits of cross-fertilization of these three areas, which indeed show interesting convergence points.

Concerning web services, this paper focuses on a central issue: studying declarative descriptions aimed at allowing forms of automated interoperation that include, on the one hand, the automation of tasks like matchmaking and execution of web services, on the other, the automation of service *selection and composition* in a way that is customized w.r.t. the *user's goals* and *needs*, a task that can be considered as a form of *personalization* [2]. Indeed, selection and composition not always are to be performed on the sole basis of general properties of the services themselves and of their interactive behavior, such as the category of the service or the functional compositionality of a set of services, but they should also take into account the user's intentions (and purposes) which both motivate and constrain the search or the composition. As a quick example, consider a web service that allows buying products, alternatively paying cash or by credit card: a user might have preferences on the form of payment to enact. In order to decide whether or not buying at this shop, it is necessary to single out the specific course of interaction that allows buying cash. This form of personalization can be obtained by applying *reasoning techniques* on a description of the service process. Such a description must have a well-defined meaning for all the parties involved. In this issue it is possible to distinguish three necessary components: first, web services capabilities must be represented according to some declarative formalism with a well-defined semantics, as also recently observed by van der Aalst [1]; second, automated tools for reasoning about such a description and performing tasks of interest must be developed; third in order to gain flexibility in fulfilling the user's request, reasoning tools should represent such requests as *abstract goals*.

The approach that we propose is to exploit results achieved by the community that studies *logic for agent systems* and, in particular, *reasoning about actions and change*. Indeed, the availability of semantic information about web resources enables the application of reasoning techniques, such as ontology reasoning constraint reasoning, non-monotonic reasoning, and temporal reasoning [3], whose use would allow the design of systems that, being able of autonomous decisions, can adapt to different users and are open to interact with one another. In particular, we propose to use techniques for *reasoning about actions* for performing the automatic selection and composition of web services, in a way that is customized w.r.t. the users's request, by reasoning on the *communicative behavior* of the services. Communication can, in fact, be considered as the behavior resulting from the application of a special kind of actions: *communication actions*. The reasoning problem that this proposal faces can intuitively be described as looking for a an answer to the ques-

tion “Is it possible to make a deal with this service respecting the user’s goals?”. Given a logic-based representation of the service policies and a representation of the customer’s needs as abstract goals, expressed by a logic formula, logic programming reasoning techniques are used for understanding if the constraints of the customer fit in with the policy of the service.

This proposal inherits from the experience of the research community that studies MAS and, in particular, logic-based formalizations of interaction aspects. Indeed, communication has intensively been studied in the context of formal theories of agency [4,5] and a great deal of attention has been devoted to the definition of standard agent communication languages (ACL), e.g. FIPA [6] and KQML [7]. Recently, most of the efforts have been devoted to the definition of formal models of interaction among agents, that use *conversation protocols*. The interest for protocols is due to the fact that they improve the interoperability of the various components (often separately developed) and allow the verification of compliance to the desired standards. Given the abstraction of web services as entities, that communicate by following predefined, public and sharable interaction protocols, we have studied the possible benefits provided by a *declarative description* of their communicative behavior, in terms of personalization of the service selection and composition. The approach models the interaction protocols provided by web services by a set of logic clauses, thus at high (not at network) level. A description by policies is definitely richer than the list of input and output, precondition and effect properties usually taken into account for the matchmaking (see Section 5). Moreover having a logic specification of the protocol, it is possible to reason about the effects of engaging specific conversations, and, on this basis, to perform many tasks in an automatic way. Actually, the proposed approach can be considered as a *second step* in the matchmaking process, which narrows a set of already selected services and performs a *customization* of the interaction with them.

For facing the problem of describing and reasoning about conversation protocols, we have extended the agent language DyLOG [8] by introducing a communication kit, that is presented in this article. DyLOG is an agent programming language, based on a modal logic for reasoning about actions and beliefs, which has already been used in the development of adaptive web applications [9]. It allows the specification of the behavior of rational agents, and it supplies mechanisms for reasoning about it. In Section 3 we presents an *extension* for dealing with communication. This extension is based on an agent theory, in which agents have *local* beliefs about the world and about the mental states of the other agents, and where communications are modelled as actions that operate on such beliefs. This account of communication aims at coping with two main aspects: the *change in the state* that encodes an agent’s beliefs, caused by a communicative act, and the *decision strategy* used by an agent for answering to a received communication. To these aims, the semantics of primitive speech acts is described in terms of effects on the mental state, both in the case in which the agent is the sender and in the case in which it is the recipient, and, in the line of [10], conversation protocols are used as

decision procedures. Each agent has a *subjective perception* of the on-going conversations and, guided by the protocol, it makes hypothetical assumptions on the other agents' answers. In the web service application context we exploit such a feature by including in the knowledge base of an agent (the requester) a description of the potentially interesting services. This description is given by logic rules expressing their communicative policies from the point of view of the requester. The language provides a goal-directed proof procedure that supports reasoning on communication and allows an agent to reason about the interaction that it is going to enact *before* it actually occurs, with the aim of proving properties of the possible executions.

In Section 2 we will locate our proposal in the context of current research on web services, while in Sections 2.1 and 4 we will show by means of examples how it is possible to use the new tools offered by the language for representing and reasoning on the conversation policies of the web services for personalizing the retrieval, the composition of, and the interaction with services. In this perspective, this article integrates and extends the work in [11,12].

2 Context and perspectives

In the last years distributed applications over the World-Wide Web have obtained wide popularity and uniform mechanisms have been developed for handling computing problems which involve a large number of heterogeneous components, that are physically distributed and that interoperate. These developments have begun to coalesce around the web service paradigm, where a service can be seen as a component available over the web. Each service has an interface that is accessible through standard protocols and that describes the *interaction capabilities* of the service. It is possible to develop new applications over the web by *combining and integrating* existing web services.

In this scenario, two needs have inspired recent research [13]: the first is the need of developing *programming languages* for implementing the behavior of the single participants involved in an interaction, the other is the need of developing *modelling languages* for describing processes and their *interaction protocols*, abstracting from details concerning the internal behavior of the single components. At the moment of writing this article, the language BPEL4WS (BPEL for short [14]) has emerged as the standard for specifying the business processes of single services and it allows writing a *local* view of the interaction that should take place, i.e. the interaction from the point of view of the process. Its authors envisioned the use of the language both as an *execution language*, for specifying the actual behavior of a participant in a business interaction, and as a modelling language, for specifying the interaction at an abstract level, once again from the perspective of the service being described. If, on a hand, BPEL as an execution language is extremely suc-

cessful, BPEL as a language for modelling abstract interactions substantially did not succeed [1]. Its limit is that it does not allow to perform the analysis of the described process, but the capability of performing this analysis is fundamental to the real implementation of those sophisticate forms of flexibility and composition that one expects in the context of the web. For achieving such a flexibility and enable automatic devices to use a web resource, the latter must bear some public information about itself, its structure, the way in which it is supposed to be used, and so forth. This information should be represented according to some conventional formalism which relies on *well-founded models*, upon which it is possible to define access and usage mechanisms. To meet these requirements, the attention of part of the community has focussed on capturing the behavior of BPEL in a formal way, and many translations of BPEL into models supporting analysis and verification (process algebras, petri nets, finite state machines) are currently under investigation [15,16]. For what concerns the *specification of interaction protocols*, instead, there is a growing agreement on the fact that the local point of view is not sufficient and that a *global* view of the interaction to occur should be expressed. This level of representation is captured by the concept of *choreography*. Choreographies are expressed by using specific languages, like the W3C proposal WS-CDL [17]. We will discuss about choreographies further below in the section.

In parallel, the World-Wide Web Consortium (W3C), has given birth to the *Semantic Web* initiative [18] (see also [19,20]) which is centered on providing a common framework, that allows resources to be shared and reused across application, enterprise, and community boundaries. In order to be *machine-processable*, information must be given a well-defined meaning; one underling assumption is that its representation has a *declarative format*. Research in the Semantic Web area is giving very good results for what concerns the intelligent retrieval and use of documents, nevertheless, research on semantic web services is still at its beginning. The current W3C proposal for describing semantic web services is the language OWL-S. Like BPEL, OWL-S allows the description of possibly composite processes from a *local* perspective. Noticeably, the sets of operators that the two languages supply, and by which the processes can be composed, though not fully identical, show a consistent intersection. The main difference between the two languages seems to be that BPEL is more focussed on the *message exchange* between the service and its parties, while OWL-S is more focussed on the *process advertisement* and the *process structure*. Another similarity is that both languages allow the definition of executable specifications. For the sake of completeness, OWL-S is but a proposal and alternative initiatives are being conducted, like WSMO [21]. It is also interesting to notice that in the Semantic Web context, there is currently no proposal of a concept close to that of “choreography”, the questions of service effective executability and interoperability are open, and the proposed matchmaking techniques are still simple and quite far from fully exploiting the power of shareable semantics.

We have observed two interesting convergence points, one between the vision reported in [13] and results of research in multi-agent systems, the other between

research in multi-agent systems and that in the Semantic Web. The former is that also in the case of agent systems, research has identified in message exchange the key to interoperation, even in the case in which agents are not rational. In particular, in an agent system it is possible to distinguish the *global interaction schema* that the group of agents should realize, the interaction protocol (analogous to a choreography), from a declarative representation of the interactive *behavior of the single* agents, their interaction policies (executable specifications). Protocols are public and shared by the agents; policies can be made public. Similarly to what postulated by the Semantic Web, both descriptions are supposed as being declarative, with a well-defined meaning and a well-founded semantics. Moreover, agent systems are commonly supposed as being open and constituted by heterogeneous members. In this context, besides giving a global view of communication, protocols can also be considered as a specification, in an interlanguage (koine), that can be used –even run-time– for selecting agents that should interact in some task. Partners which agree on using a global interaction protocol, can cooperate even if their local interactive behaviors are implemented by using different programming languages. The respect of the global rules guarantees the interoperability of the parties (i.e. their capability of *actually* producing an interaction), and that the interactions will satisfy expected requirements. Of course in this context verifying that the local implementations conform to the global specification becomes crucial.

On the other hand, having a declarative representation with a strong formal basis makes it possible to apply *reasoning techniques* for achieving goals of interest or prove properties of interest, the formalization basically depending on the reasoning tasks to be tackled. In particular, in order to introduce that degree of flexibility that is necessary for personalizing the selection and composition of web services in an open environment, in such a way that the user's goals and needs are satisfied, it is necessary to reason on the interactive behavior that the service can enact by using a declarative language that both supports the *explicit coding* of interaction policies, and that refers to a formal model for reasoning on *knowledge attitudes* like goals and beliefs. As we will show in the following examples, this kind of formalization allows the selection of specific courses of interaction among sets of possible alternatives, based on the user's likings and constraints, a selection process that can be extended to the case of service composition. To our knowledge, none of the most widely diffused languages for representing (semantic) web services has all these characteristics, so for proposing these advanced forms of selection it is necessary to borrow from the neighboring area of multi-agent systems. We have chosen the DyLOG language for writing the abstract specifications of the service behaviors because the language shows all these characteristics.

2.1 A scenario

Let us now introduce a simple scenario that will help us in showing how, by reasoning about the communicative behavior of web services, it is possible to personalize their fruition. The scenario encompasses two kinds of web services for taking reservation: services to book tables at restaurants and services to book seats at cinemas. In this framework it is quite easy to invent queries of various complexity. Not all of them can be answered by the basic keyword-based retrieval mechanisms (see related works in Section 5) with sufficient precision. To the aims of this example it is sufficient to suppose that there are only two restaurant booking services (we will simply call them *restaurant1* and *restaurant2*) and two cinema booking services (*cinema1* and *cinema2*). On a keyword basis, the two cinemas have the same description as well as the two restaurants but all the services have a different interactive behavior with their clients. Figure 1 and Figure 2 report their interaction protocols, represented as AUML [22,23] sequence diagrams. In particular, *restaurant1* takes part to a promotion campaign, by which each customer, who made a reservation by the internet service, receives a free ticket for a movie (Figure 1 (i)). *Restaurant2* does not take part to this initiative (Figure 1 (ii)). On the side of cinemas, *cinema2* accepts reservations to be paid cash but no free tickets (Figure 2 (iv) whereas *cinema1* accepts the promotional tickets and, as an alternative, it also takes reservations by credit card (Figure 2 (iii)).

One of the simplest tasks that can be imagined is web service retrieval, in this case the query is a description of the desired service (e.g. “cinema booking service”). However, what to do when the user is not simply interested in a generic kind of service but s/he would like to find services that besides being of that kind also show other characteristics? For instance to find a cinema booking service that does not require to confirm the reservation by sending a credit card number. While in the former case a keyword-based description would have been sufficient, and the answer would have included both *cinema1* and *cinema2*, in the latter case more information about the service is necessary. How to consider *cinema1* that requires either the credit card or a promotional free ticket? Even more interesting is the case in which the answer to the user’s goal requires to combine the executions of two (or more) independent web services. For instance, let us suppose that the user would like to organize an evening out by, first, having dinner at some nice restaurant and, then, watching a movie; moreover, s/he heard that in some restaurants it is possible to gain a free ticket for a movie and s/he would like to profit of this possibility but only if it is not necessary to use the credit card because s/he does not trust internet connections very much. If, on one hand, searching for a cinema or a restaurant reservation service is a task that can be accomplished by any matchmaking approach, the conditions “look for promotions” and “do not use my credit card” can be verified only by a rational inspection of the communication with the services. This verification should be done before the actual interaction. Obviously, the only combination that satisfies all the user’s requests is to reserve a table at *restaurant1*,

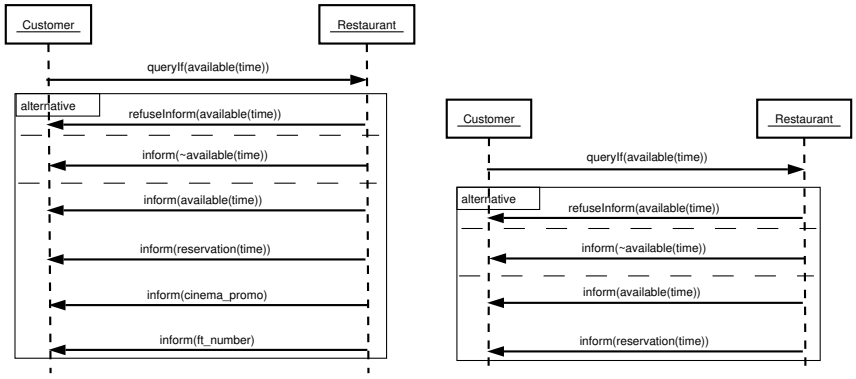
and then make a reservation at *cinema1*. This solution can be found only enacting a reasoning mechanism that, based on the description of the interaction protocols, selects *restaurant1* and *cinema1* and, in particular, it selects the course of action that makes use of the free ticket.

In order to perform this kind of tasks, in this work we set the web service selection and composition problems in a multi-agent perspective. The idea is to interpret web services as agents, each bearing a public communication protocol (described in a declarative format), and to delegate the task of selection (or composition) to a *personalization agent*. Such an agent receives a description of the user's desires and goals, interacts with other systems that perform a less informed retrieval of services, and, in order to find a suitable solution to the user's request, applies a reasoning mechanism to the communication protocols of the services. The protocols of such services, that have been identified by the less informed retrieval systems, are supposed to be represented in a specification language, such as AUML [22,23]. We assume that this representations are translated in DyLOG procedures that are included in the knowledge base of the personalization agent. Given this representation, in order to accomplish the reasoning tasks that have been described, we propose the use of procedural planning techniques, in which the possible executions of a sequence of protocols are evaluated, with the aim of deciding if they meet or not the requirements or under which assumptions. In particular, we will exploit the planning capabilities offered by the DyLOG agent programming language, which is described in Section 3.

Of course, the agent cannot be sure that once the plan is executed it will not fail because the planning process is performed before the interaction; so, for instance, it cannot know if there will actually be a free table left at the restaurant. No currently existing selection mechanism can guarantee this but it would be interesting to study the relations between replanning techniques and the compensation techniques developed for long-time transactions (see the Conclusions for further discussion). The advantage of the proposed approach is that it leaves out services, that would in no case allow the achievement of the user's goals respecting all the requirements, as well as it allows the exact identification of those permitted courses of interaction that satisfy them.

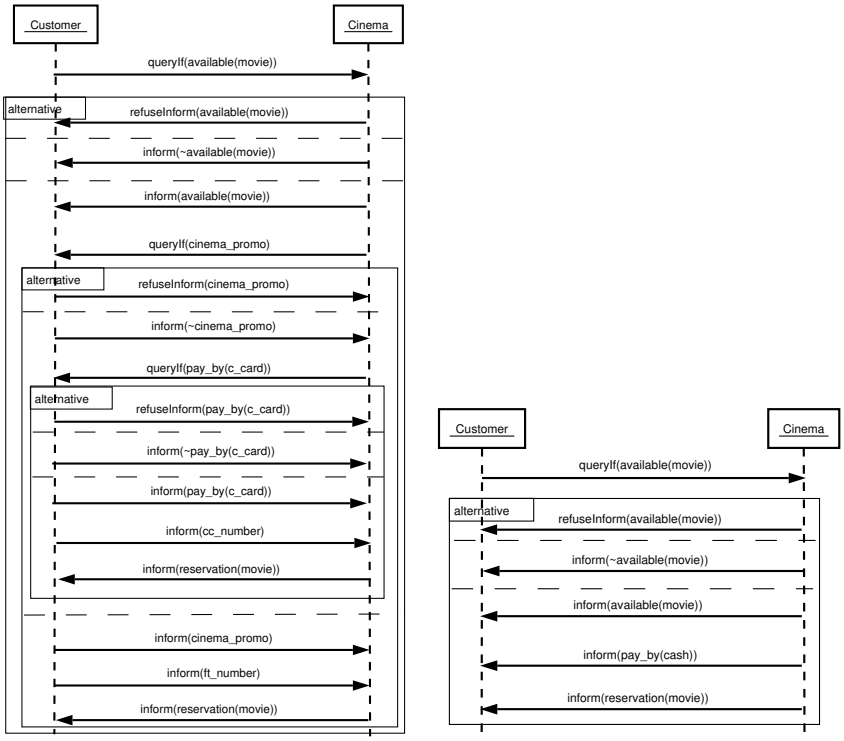
Present web service technology is quite primitive w.r.t. the framework we propose, and the realization of the picture sketched above would require an integration, in the current tools for web service representation and handling, of knowledge representation languages –in the line of those developed in the Semantic Web area– and of techniques for reasoning and dealing with communication, inspired by those studied in the area of MAS. Even if a real integration is still far from being real for the sake of concreteness, let us describe our *vision* of the steps to be taken toward the realization.

In our view, public descriptions of interaction protocols in the scenario above (rep-



(i) (ii)

Fig. 1. The AUML sequence diagrams represent the interactions occurring between the customer and each of the restaurant web services: (i) is followed by *restaurant1*, (ii) by *restaurant2*.



(iii) (iv)

Fig. 2. In this case, the AUML sequence diagrams represent the possible interactions between the customer and each cinema web service: (iii) represents the protocol followed by *cinema1*; (iv) is followed by *cinema2*.

resented by AUML diagrams) can be mapped in public descriptions of choreographies (e.g. WS-CDL-like descriptions). A choreography defines a global view of

the protocol followed by a certain service, e.g. the *cinema service*, for accomplishing the cooperative task of booking a cinema ticket. A *costumer* service, that in principle is interested to participate to the cooperation for booking a ticket for its user, *translates* such a description in the declarative language DyLOG [2] and uses reasoning techniques, supported by the language, plus its local knowledge on the user's preferences for checking whether the contract, defined by the choreography, satisfies its user. Such reasoning *conditions* the costumer's decision of selecting the cinema service as a partner in a real interaction for accomplishing a task. Of course the outcome of the reasoning is meaningful under the following assumption: the implementation of the cinema service behavior (that could be written in an execution language like BPEL) must be *conformant* w.r.t. the choreography specification that is used as input of the reasoning. Verifying the conformance and the interoperability of web services to a global protocol definition (to be provided at the choreography level) is definitely one of the hot topics at the moment. In [24] we started to attack the problem by proposing a framework, based on the theory of *formal languages*, where both the global protocol and the web service behavior are expressed by using finite state automata. Instead in [25] choreography and orchestration are formalized by using two process algebras and conformance takes the form of a bisimulation-like relation.

3 A declarative agent language for reasoning about communication

Logic-based, executable languages for agent specification have been deeply investigated in the last years [26–28,8,29]. In this section we introduce the main features of DyLOG, our agent language for reasoning about actions and changes, and, in particular, we present the extension for dealing with communication (the CKit, Section 3.2), which will be explained with more details and with examples. Section 3.3 presents the semantics of the CKit based on a non-monotonic solution to deal with the persistency problem, while Section 3.4 presents the reasoning mechanisms that will be used in Section 4. The proof theory is reported in Appendix A.

DyLOG is a logic programming language for modeling rational agents, based on a modal theory of actions and mental attitudes. In this language *modalities* are used for representing *actions* while *beliefs* model the agent's internal state. The language refers to a *mentalistic* approach, which is also adopted by the standard FIPA-ACL [6], where communicative actions affect the internal mental state of the agent. The mental state of an agent is described in terms of a consistent set of *belief formulas*. The modal operator $\mathbf{B}^{ag_i}$ models the beliefs of the agent ag_i. The modal operator $\mathbf{M}^{ag_i}$ is defined as the dual of $\mathbf{B}^{ag_i}$ ($\mathbf{M}^{ag_i}\varphi \equiv \neg\mathbf{B}^{ag_i}\neg\varphi$); intuitively it represents the fact that agent ag_i considers φ possible. The language allows also dealing with *nested beliefs*, which allow the representation of what an agent thinks about the

[2] This process requires the selection of a proper ontology, see Section 5.

other agents' beliefs, and make reasoning on how they can be affected by communicative actions possible. DyLOG accounts both for *atomic* and *complex actions*, or procedures. Atomic actions are either *world* actions, that is actions which affect the world, or *mental* actions, i.e. sensing or communicative actions which only modify the agent's beliefs. For each world action and for each agent the modalities $[a^{ag_i}]$ and $\langle a^{ag_i}\rangle$ are defined: $[a^{ag_i}]\varphi$ denotes the fact that the formula φ holds after *every* execution of a performed by agent ag_i, while $\langle a^{ag_i}\rangle\varphi$, represents the *possibility* that φ holds after the action has been executed by the agent. A modality $Done(a^{ag_i})$ is also introduced for expressing that a (a communicative act or a world action) has been executed. Last but not least, the modality $\Box$ (box) denotes formulas that hold in all the possible agent mental states. The formalization of *complex actions* draws considerably from dynamic logic for the definition of action operators like sequence (;), ruled by $\langle a; b\rangle\varphi \equiv \langle a\rangle\varphi$, test (?), $\langle\psi?\rangle\varphi \equiv \psi \wedge \varphi$ and non-deterministic choice ($\cup$), $\langle a \cup b\rangle\varphi \equiv \langle a\rangle\varphi \vee \langle b\rangle\varphi$ but, differently than [30], DyLOG refers to a *Prolog-like* paradigm and procedures are defined as recursive Prolog-like clauses. Analogously to what done in the case of atomic actions, for each procedure p, the language contains also the *universal* and *existential modalities* $[p]$ and $\langle p\rangle$. All the modalities of the language are normal; $\Box$ is reflexive and transitive and its interaction with action modalities is ruled by the axiom $\Box\varphi \supset [a^{ag_i}]\varphi$, that is, in ag_i's mental state φ will hold after every execution of any action performed by the agent. The epistemic modality $\mathbf{B}^{ag_i}$ is serial, transitive and Euclidean. The interaction of $Done(a^{ag_i})$ with other modalities is ruled by the axioms $\varphi \supset [a^{ag_i}]Done(a^{ag_i})\varphi$ and $Done(a^{ag_j})\varphi \supset \mathbf{B}^{ag_i}Done(a^{ag_j})\varphi$ (*awareness*), with $ag_i = ag_j$ when a^{ag_i} is not a communicative action.

3.1 The agent theory

DyLOG agents are considered as individuals, each with its *subjective* view of a dynamic domain. The framework does not model the real world but only the internal dynamics of each agent in relation to the changes caused by actions. An agent's *behavior* is specified by a *domain description* that includes:

(1) the agent's *belief state*;
(2) *action* and *precondition laws* that describe the effects and the preconditions of atomic world actions on the executor's mental state;
(3) *sensing axioms* for describing atomic sensing actions;
(4) *procedure axioms* for describing complex behaviors.

Let us denote by the term *belief fluent* F, a belief formula $\mathbf{B}^{ag_i}L$ or its negation, where L, the *belief argument*, is a *fluent literal* (f or $\neg f$), a *done fluent* ($Done(a^{ag_i})\top$ or its negation), or a belief fluent of *rank 1* ($\mathbf{B}l$ or $\neg\mathbf{B}l$). In this latter case the symbol l is an *attitude-free* fluent, that is a fluent literal or a done fluent.

Intuitively, the *belief state* contains what an agent (dis)believes about the world and about the other agents. It is a complete and consistent set of *rank 1* and 2 belief fluents. A belief state provides, for each agent, a three-valued interpretation of all the possible belief arguments L, that can either be *true*, *false*, or *undefined* when both $\neg\mathbf{B}^{ag_i}L$ and $\neg\mathbf{B}^{ag_i}\neg L$ hold. $\mathbf{U}^{ag_i}L$ expresses the ignorance of ag_i about L.

World actions are described by their preconditions and effects on the *actor*'s mental state; they trigger a revision process on the actor's beliefs. Formally, *action laws* describe the conditional effects on ag_i's belief state of an atomic action a, executed by ag_i itself. They have the form:

$$\Box(\mathbf{B}^{ag_i}L_1 \wedge \ldots \wedge \mathbf{B}^{ag_i}L_n \supset [a^{ag_i}]\mathbf{B}^{ag_i}L_0) \quad (1)$$
$$\Box(\mathbf{M}^{ag_i}L_1 \wedge \ldots \wedge \mathbf{M}^{ag_i}L_n \supset [a^{ag_i}]\mathbf{M}^{ag_i}L_0) \quad (2)$$

Law (1) states that if ag_i believes the preconditions to an action a in a certain epistemic state, after a execution, ag_i will also believe the action's effects. (2) states that when the preconditions of a are unknown to ag_i, after the execution of a, ag_i will consider unknown also its effects [3]. *Precondition laws*, instead, specify mental conditions that make an atomic action executable in a state. An agent ag_i can execute a when the precondition fluents of a are in its belief state. More formally:

$$\Box(\mathbf{B}^{ag_i}L_1 \wedge \ldots \wedge \mathbf{B}^{ag_i}L_n \supset \langle a^{ag_i}\rangle\top) \quad (3)$$

Sensing Actions produce knowledge about fluents; they are defined as non-deterministic actions, with unpredictable outcome, formally modelled by a set of *sensing axioms*. Each sensing action s has associated a set $dom(s)$ of literals (its *domain*). When ag_i executes s, it will know which of such literals is true.

$$[s]\varphi \equiv [\bigcup_{l\in dom(s)} s^{\mathbf{B}^{ag_i}l}]\varphi \quad (4)$$

$\cup$ is the choice operator of dynamic logic and $s^{\mathbf{B}^{ag_i}l}$, for each $l \in dom(s)$, is an *ad hoc* primitive action, that probes one of the possible outcomes of the sensing. Such primitive actions are ruled by the simple action clauses:

$$\Box(\mathbf{B}^{ag_i}l_1 \wedge \ldots \wedge \mathbf{B}^{ag_i}l_n \supset \langle s^{\mathbf{B}^{ag_i}l}\rangle\top) \quad (5)$$
$$\Box(\top \supset [s^{\mathbf{B}^{ag_i}l}]\mathbf{B}^{ag_i}l) \quad (6)$$
$$\Box(\top \supset [s^{\mathbf{B}^{ag_i}l}]\mathbf{B}^{ag_i}\neg l') \quad (7)$$

for each $l' \in dom(s)$, $l \neq l'$. Clause (5) means that after any sequence of world actions, if the set of literals $\mathbf{B}^{ag_i}l_1 \wedge \ldots \wedge \mathbf{B}^{ag_i}l_n$ holds, then the action $s^{\mathbf{B}^{ag_i}l}$ can

[3] Laws of form (2) allow actions with non-deterministic effects, that may cause a *loss* of knowledge, to be specified.

be executed. The other ones describe the effects of $s^{\mathbf{B}^{ag_i}l}$: in any state, after the execution of $s^{\mathbf{B}^{ag_i}l}$, l is believed (6), while all the other fluents belonging to $dom(s)$ are believed to be false (7). Note that the binary sensing action on a fluent l, is a special case of sensing where the associated finite set is $\{l, \neg l\}$.

Complex actions specify complex behaviors by means of *procedure definitions*, built upon other actions. Formally, a complex action is a collection of *inclusion axiom schema* of the modal logic, of form:

$$\langle p_0 \rangle \varphi \subset \langle p_1; p_2; \ldots; p_m \rangle \varphi \tag{8}$$

p_0 is a procedure name, ";" is the *sequencing operator* of dynamic logic, and the p_i's, $i \in [1, m]$, are procedure names, atomic actions, or test actions. Procedure definitions may be *recursive* and procedure clauses can be executed in a *goal-directed* way, similarly to standard logic programs.

3.2 Communication

A *communication theory* has been integrated in the general agent theory by adding further axioms and laws to the agents' domain description. It consists of *speech acts*, *get-message actions* and *conversation policies*.

Speech acts are atomic actions of the form $\mathsf{speech\_act}(sender, receiver, l)$ where $sender$ and $receiver$ are agents and l is either a fluent literal or a done fluent. Since agents have a personal view of the world, the way in which an agent's beliefs are modified by the execution of a speech act depends on the *role* that it plays. For this reason, speech act specification is twofold: one definition holds when the agent is the *sender*, the other when it is the *receiver*. Speech acts are modelled by generalizing the action and precondition laws of world actions, so to enable the representation of the effects of communications that are performed by other agents. When the agent is the sender, the precondition laws contain some *sincerity condition* that must hold in the agent's mental state. When ag_i is the receiver, the action is supposed as *always* executable (ag_i has no control over a communication performed by another agent). This representation allows agents to *reason about* conversation effects. Hereafter are a few examples of speech act, as defined in DyLOG: they are the specification of the *inform*, *queryIf*, and *refuseInform* FIPA-ACL primitive speech acts.

$\mathsf{inform}(Self, Other, l)$
a) $\Box(\mathbf{B}^{Self} l \wedge \mathbf{B}^{Self}\mathbf{U}^{Other} l \supset \langle \mathsf{inform}(Self, Other, l)\rangle \top)$
b) $\Box([\mathsf{inform}(Self, Other, l)]\mathbf{M}^{Self}\mathbf{B}^{Other} l)$
c) $\Box(\mathbf{B}^{Self}\mathbf{B}^{Other} authority(Self, l) \supset [\mathsf{inform}(Self, Other, l)]\mathbf{B}^{Self}\mathbf{B}^{Other} l)$
d) $\Box(\top \supset \langle \mathsf{inform}(Other, Self, l)\rangle \top)$

e) $\Box([\mathsf{inform}(Other, Self, l)]\mathbf{B}^{Self}\mathbf{B}^{Other}l)$
f) $\Box(\mathbf{B}^{Self}authority(Other, l) \supset [\mathsf{inform}(Other, Self, l)]\mathbf{B}^{Self}l)$
g) $\Box(\mathbf{M}^{Self}authority(Other, l) \supset [\mathsf{inform}(Other, Self, l)]\mathbf{M}^{Self}l)$

Clause (a) represents the *executability preconditions* for the action $\mathsf{inform}(Self, Other, l)$: it specifies those mental conditions that make this action executable in a state. Intuitively, it states that $Self$ can execute an inform act only if it believes l ($\mathbf{B}^{Self}$ models the beliefs of agent $Self$) and it believes that the receiver ($Other$) does not know l. According to clause (b), the agent also considers possible that the receiver will adopt its belief (the modal operator $\mathbf{M}^{Self}$ is the dual of $\mathbf{B}^{Self}$ by definition), although it cannot be sure that this will happen by the *autonomy assumption*. Nevertheless, if agent $Self$ thinks to be considered by the receiver a *trusted authority* about l, it is also confident that $Other$ will adopt its belief, clause (c). Since executability preconditions can be tested only on the mental state of $Self$, when $Self$ is the receiver the action of informing is considered as *always* executable, clause (d). When $Self$ is the receiver, the effect of an inform act is that $Self$ will believe that l is believed by the sender ($Other$), clause (e), but $Self$ will adopt l as an own belief only if it thinks that $Other$ is a trusted authority, clause (f).

$\mathsf{querylf}(Self, Other, l)$
a) $\Box(\mathbf{U}^{Self}l \wedge \neg\mathbf{B}^{Self}\mathbf{U}^{Other}l \supset \langle\mathsf{querylf}(Self, Other, l)\rangle\top)$
b) $\Box(\top \supset \langle\mathsf{querylf}(Other, Self, l)\rangle\top)$
c) $\Box([\mathsf{querylf}(Other, Self, l)]\mathbf{B}^{Self}\mathbf{U}^{Other}l)$

By querylf an agent asks another agent if it believes that l is true. To perform a querylf act, $Self$ must ignore l and it must believe that the receiver does not ignore l, clause (a). After a querylf act, the receiver will believe that the sender ignores l.

$\mathsf{refuseInform}(Self, Other, l)$
a) $\Box(\mathbf{U}^{Self}l \wedge \mathbf{B}^{Self}Done(\mathsf{querylf}(Other, Self, l))\top$
$\quad\quad \supset \langle\mathsf{refuseInform}(Self, Other, l)\rangle\top)$
b) $\Box(\top \supset \langle\mathsf{refuseInform}(Other, Self, l)\rangle\top)$
c) $\Box([\mathsf{refuseInform}(Other, Self, l)]\mathbf{B}^{Self}\mathbf{U}^{Other}l)$

By refuseInform an agent refuses to give an information it was asked for. The refusal can be executed only if the sender ignores l and it believes that the receiver previously queried it about l, clause (a). A refusal by some other agent is considered as always possible, clause (b). After a refusal the receiver believes that the sender ignores l.

Get-message actions are used for *receiving* messages from other agents. Since, from the agent perspective, they correspond to *queries* for an external input, they are modeled as a special kind of sensing actions, whose outcome is unpredictable. The main difference w.r.t. normal sensing actions is that they are defined by means of speech acts performed by the *interlocutor*. Formally, get_message actions are

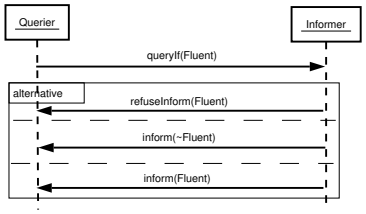


Fig. 3. The AUML sequence diagram represents the communicative interactions occurring between the *querier* and the *informer* in the yes_no_query protocol.

defined by axiom schemas of the form:

$$[\mathsf{get\_message}(ag_i, ag_j, l)]\varphi \equiv [\bigcup_{\mathsf{speech\_act} \in \mathbf{C}_{\mathsf{get\_message}}} \mathsf{speech\_act}(ag_j, ag_i, l)]\varphi \quad (9)$$

$\mathbf{C}_{\mathsf{get\_message}}$ is a finite set of speech acts, which are all the possible communications that agent ag_i expects from agent ag_j in the context of a given conversation. A get_message action does not have a domain of mental fluents associated to it, the information is obtained by looking at the effects of such speech acts on ag_i's mental state.

Conversation protocols specify patterns of communication; they define the context in which speech acts are executed [10] and are modelled by means of *procedure axioms* having the form (8). Since agents have a subjective perception of the communication, each protocol has as many procedural representations as the possible *roles* in the conversation. We will call *policy* each such role-centric implementation.

Let us consider, for instance the yes_no_query protocol reported in Fig. 3, a simplified version of the FIPA Query Interaction Protocol [31]. The protocol has two complementary views, one to be followed for performing a query ($\mathsf{yes\_no\_query}_Q$) and one for responding ($\mathsf{yes\_no\_query}_I$). Let us show how it would be possible to implement it in DyLOG.

(a) $\langle \mathsf{yes\_no\_query}_Q(Self, Other, Fluent)\rangle\varphi \subset$
$\langle \mathsf{queryIf}(Self, Other, Fluent); \mathsf{get\_answer}(Self, Other, Fluent)\rangle\varphi$

(b) $[\mathsf{get\_answer}(Self, Other, Fluent)]\varphi \equiv$
$[\mathsf{inform}(Other, Self, Fluent) \cup \mathsf{inform}(Other, Self, \neg Fluent) \cup$
$\mathsf{refuseInform}(Other, Self, Fluent)]\varphi$

Trivially, in $\mathsf{yes\_no\_query}_Q$ agent $Self$ performs a queryIf speech act then it waits for the answer of agent $Other$. The definitions of get_answer and get_start (the latter is reported hereafter, axiom (f)) are instances of the get_message axiom. Intuitively, the right hand side of get_answer represents all the possible answers expected by agent $Self$ from agent $Other$ about $Fluent$, in the context of a conversation ruled by the $\mathsf{yes\_no\_query}_Q$ conversation policy.

(c) $\langle \mathsf{yes\_no\_query}_I(Self, Other, Fluent)\rangle\varphi \subset$
$\langle \mathsf{get\_start}(Self, Other, Fluent);$
$\mathbf{B}^{Self} Fluent?; \mathsf{inform}(Self, Other, Fluent)\rangle\varphi$

(d) $\langle \mathsf{yes\_no\_query}_I(Self, Other, Fluent)\rangle\varphi \subset$
$\langle \mathsf{get\_start}(Self, Other, Fluent);$
$\mathbf{B}^{Self} \neg Fluent?; \mathsf{inform}(Self, Other, \neg Fluent)\rangle\varphi$

(e) $\langle \mathsf{yes\_no\_query}_I(Self, Other, Fluent)\rangle\varphi \subset$
$\langle \mathsf{get\_start}(Self, Other, Fluent);$
$\mathbf{U}^{Self} Fluent?; \mathsf{refuseInform}(Self, Other, Fluent)\rangle\varphi$

The $\mathsf{yes\_no\_query}_I$ protocol specifies the behavior of the agent $Self$, when it plays the role of the informer, waiting for a query from $Other$ and, then, replying in accordance to its beliefs on the query subject. Last but not least, rule (f) reports the axiom by which get_start is defined:

(f) $[\mathsf{get\_start}(Self, Other, Fluent)]\varphi \equiv [\mathsf{queryIf}(Other, Self, Fluent)]\varphi$

It is a renaming of queryIf.

We are now in condition to define the *communication kit*, denoted by CKit^{ag_i}, of an agent ag_i as the triple $(\Pi_{\mathbf{C}}, \Pi_{\mathbf{CP}}, \Pi_{\mathbf{S}get})$, where $\Pi_{\mathbf{C}}$ is the set of simple action laws defining ag_i's speech acts, $\Pi_{\mathbf{S}get}$ is the set of axioms that specify ag_i's get_message actions and $\Pi_{\mathbf{CP}}$ is the set of procedure axioms specifying its conversation protocols.

A *Domain Description* defining an agent ag_i is, then, a triple $(\Pi, \mathsf{CKit}^{ag_i}, S_0)$, where CKit^{ag_i} is the agent's communication kit, S_0 is ag_i's initial set of belief fluents, and Π is a specification of the agent's non-communicative behavior. It is a triple $(\Pi_{\mathbf{A}}, \Pi_{\mathbf{S}}, \Pi_{\mathbf{P}})$, where $\Pi_{\mathbf{A}}$ is the set of ag_i's world action and precondition laws, $\Pi_{\mathbf{S}}$ is the specification of a set of sensing action, $\Pi_{\mathbf{P}}$ a set of axioms that define complex actions.

3.3 Dealing with persistency

In the DyLOG framework, a non-monotonic solution is adopted to deal with the persistency problem. More precisely, an abductive semantics is proposed for the language, in which abductive assumptions are used to model the persistency of beliefs fluents, from a state to the following one, when an action is performed. The solution is a generalization of [8], that allows one to deal also with nested beliefs and communicative actions, and consists in maximizing persistency assumptions about epistemic fluents after the execution of action sequences. In particular any belief fluent F which holds in a given state is assumed to persist through an action, unless it is inconsistent to assume so, i.e. unless $\neg F$ holds after the action execution.

Notice that belief states are *inconsistent* when they contain either a belief $\mathbf{B}^{ag_i} l$ and its negation, or the belief formulas $\mathbf{B}^{ag_j}\mathbf{B}^{ag_i} l$ and $\mathbf{B}^{ag_j}\mathbf{B}^{ag_i}\neg l$, or the belief formulas $\mathbf{B}^{ag_j}\mathbf{B}^{ag_i} l$ and $\mathbf{B}^{ag_j}\neg\mathbf{B}^{ag_i} l$. However, from the *seriality* of the $\mathbf{B}^{ag_i}$ operators, the following general formula schema for the *rank 2* beliefs holds in the defined logic for any two agents ag_i and ag_j (actually, the general schema for any rank of nesting holds):

$$\mathbf{B}^{ag_i}\mathbf{B}^{ag_j}\neg\varphi \supset \neg\mathbf{B}^{ag_i}\mathbf{B}^{ag_j}\varphi \tag{10}$$

This property guarantees that when an inconsistency arises "locally" in the beliefs ascribed from ag_i to some other agent, the beliefs of ag_i itself will be inconsistent. Therefore, in case of a nested epistemic fluent $\mathbf{B}^{ag_i}\mathbf{B}^{ag_j} l$, the persistency is *correctly blocked* when a locally inconsistent fluent $\mathbf{B}^{ag_i}\mathbf{B}^{ag_j}\neg l$ becomes true after an action execution, because $\neg\mathbf{B}^{ag_i}\mathbf{B}^{ag_j} l$ can be derived from (10).

Given these considerations, the semantics is defined according to the method used by Eshghi and Kowalski in the definition of the abductive semantics for negation as failure [32]. A new set of atomic propositions of the form $\mathbb{M}[a_1]\ldots[a_m]F$ are defined as *abducibles*. [4] Their meaning is that the fluent expression F can be assumed to hold in the state obtained by the execution of the primitive actions $a_1,\ldots,a_m$. Each abducible can be assumed to hold, if it is consistent with the domain description $(\Pi, \mathsf{CKit}^{ag_i}, S_0)$ and with the other assumed abducibles. Then, we add to the axiom system, that characterizes the logic defined by the domain description, the *persistency axiom schema*:

$$[a_1]\ldots[a_{m-1}]F \wedge \mathbb{M}[a_1]\ldots[a_{m-1}][a_m]F \supset [a_1]\ldots[a_{m-1}][a_m]F$$

where $a_1,\ldots,a_m$ are primitive actions and F is a belief fluent. It means that if F holds after $a_1,\ldots,a_{m-1}$, and it can be assumed to persist after action a_m (i.e., it is consistent to assume $\mathbb{M}[a_1]\ldots[a_m]F$), one can conclude that F holds after the sequence of actions $a_1,\ldots,a_m$.

3.4 Reasoning about conversations

Given a domain description, we can reason about it and formalize the *temporal projection* and the *planning* problem by means of existential queries of form:

$$\langle p_1\rangle\langle p_2\rangle\ldots\langle p_m\rangle Fs \tag{11}$$

where each p_k, $k = 1,\ldots,m$ may be an (atomic or complex) action executed by ag_i or an external speech act, that belongs to CKit^{ag_i}. By the word *external* we denote

[4] Notice that here $\mathbb{M}$ is not a modality. $\mathbb{M}\alpha$ denotes a new atomic proposition. $\mathbb{M}\alpha$ means "α is consistent", analogously to default logic.

a speech act in which our agent plays the role of the receiver. Checking if a query of form (11) succeeds corresponds to answering the question "Is there an execution trace of the sequence $p_1, \ldots, p_m$ that leads to a state where the conjunction of belief fluents Fs holds for agent ag_i?". In case all the p_k's are atomic actions, it amounts to predict if the condition of interest will be true after their execution. In case complex actions are involved, the execution trace that is returned in the end is a *plan* to bring about Fs. The procedure definition constrains the search space.

A special case is when the procedure is a *conversation protocol*. In this case and by applying these same reasoning techniques, the agent will be able to predict how a conversation can affect its mental state and also to produce a conversation that will allow it achieve a communicative goal of interest. In this process get_message actions are treated as sensing actions, whose outcome is not known at planning time – agents cannot read each other's mind, so they cannot know in advance the answers that they will receive –. For this reason all of the possible alternatives are to be taken into account. This can be done because of the existence of the protocol. The extracted plan will, then, be *conditional*, in the sense that for each get_message action and for each sensing action it will contain as many branches as possible action outcomes. Each path in the resulting tree is a *linear plan* that brings about the desired condition Fs. More formally:

(1) an action sequence $\sigma = a_1; \ldots; a_m$, with $m \geq 0$, is a conditional plan;
(2) if $a_1; \ldots; a_m$ ($m \geq 0$) is an action sequence, $s \in \mathbf{S}$ is a sensing action, and $\sigma_1, \ldots, \sigma_t$ are conditional plans, then $\sigma = a_1; \ldots; a_m; s; ((\mathbf{B}^{ag_i} l_1?); \sigma_1 \cup \ldots \cup (\mathbf{B}^{ag_i} l_t?); \sigma_t)$ where $l_1, \ldots, l_t \in dom(s)$, is a conditional plan;
(3) if $a_1; \ldots; a_m$ ($m \geq 0$) is an action sequence, $g \in \mathbf{S}$ is a get_message action, and $\sigma_1, \ldots, \sigma_t$ are conditional plans, then $\sigma = a_1; \ldots; a_k; g; ((\mathbf{B}^{ag_i} Done(c_1) \top?); \sigma_1 \cup \ldots \cup (\mathbf{B}^{ag_i} Done(c_t) \top?); \sigma_t)$ where $c_1, \ldots, c_t \in \mathbf{C}_g$, is a conditional plan.

In some applications it is actually possible to extract a conditional plan, that leads to the goal *independently* from the answers of the interlocutor. This mechanism will be used for web service selection in Section 4.2. An alternative is to look for a linear plan that leads to the goal, given some *assumptions* on the received answers. This approach does not guarantee that at *execution time* the services will stick to the planned conversation, but it allows finding a feasible solution when a conditional plan cannot be found. This is actually the case of web service composition (Section 4.3). If we compose *restaurant1* and *cinema1*, it is possible to find a conversation after which the user's desires about the credit card and about the use of promotional tickets are satisfied. However, the success of the plan depends on information that is known only at execution time (availability of seats) and that we *assumed* during planning. In fact, if no seat is available the goal of making a reservation will fail. The advantage of reasoning about protocols, in this latter situation, is that the information contained in the protocol is sufficient to exclude *a priori* a number of compositions that will never satisfy the goal. For instance, *restaurant1* plus *cinema2* does not permit to exploit a promotion independently

from the availability of seats. The proof procedure that allows to reason about conversation protocols is a natural evolution of [8] and is described in Appendix A; it is goal-directed and based on negation as failure (NAF). NAF is used to deal with the persistency problem for verifying that the complement of a mental fluent is not true in the state resulting from an action execution, while in the modal theory we adopted an abductive characterization. The proof procedure allows agents to find *linear* plans for reaching a goal from an incompletely specified initial state. The soundness can be proved under the assumption of e-consistency, i.e. for any action the set of its effects is consistent [33]. The extracted plans always lead to a state in which the goal condition Fs holds.

4 Reasoning about conversations for web service selection and composition

This section reports a few examples aimed at showing the utility of a declarative representation of conversation protocols and policies in a Semantic Web framework. The scenario we refer to is the one introduced in Section 2.1. The web services that are involved can be classified in two categories, depending on their function: restaurant web services and cinema web services. The former allow a user to book a table at a given restaurant, the latter to book a seat at a given cinema. The interaction, however, is carried on in different ways. In particular, the services accept different forms of payment and only part of them allow users to benefit of promotions. The section is structured in the following way. First of all, we focus on knowledge representation and present the protocols used by the web services involved in the scenario. Afterwards, the two tasks of web service selection and web service composition will be tackled, showing how personalization plays an important role in both cases.

4.1 Representing conversation protocols in DyLOG

Let us begin with describing the *protocols*, that are followed by the web services. Such protocols allow the interaction of two agents (the service and the customer), so each of them encompasses two complementary views: the view of the web service and the view of the customer. Each view corresponds to an agent conversation policy, which is represented as a DyLOG procedure but for the sake of brevity, we report only the view of the customer. It is easy to see how the structures of the procedure clauses correspond to the sequence of AUML operators in the sequence diagrams. The subscripts next to the protocol names are a writing convention for representing the role that the agent plays; so, for instance, Q stands for *querier*, and C for *customer*. The customer view of the restaurant protocols is the following:

(a) $\langle \mathsf{reserv\_rest\_1}_C(Self, Service, Time)\rangle\varphi \subset$
$\langle \mathsf{yes\_no\_query}_Q(Self, Service, available(Time))$;

$\mathbf{B}^{Self} available(Time)?$;
$\mathsf{get\_info}(Self, Service, reservation(Time))$;
$\mathsf{get\_info}(Self, Service, cinema\_promo)$;
$\mathsf{get\_info}(Self, Service, ft\_number)\rangle\varphi$

(b) $\langle \mathsf{reserv\_rest\_2}_C(Self, WebS, Time)\rangle\varphi \subset$
$\langle \mathsf{yes\_no\_query}_Q(Self, WebS, available(Time))$;
$\mathbf{B}^{Self} available(Time)?$;
$\mathsf{get\_info}(Self, WebS, reservation(Time))\rangle\varphi$

(c) $[\mathsf{get\_info}(Self, Service, Fluent)]\varphi \subset [\mathsf{inform}(Service, Self, Fluent)]\varphi$

Procedure (a) is the protocol procedure that describes the communicative behavior of the first restaurant: the customer asks if a table is available at a certain time, if so, the restaurant informs it that a reservation has been taken and that it gained a promotional free ticket for a cinema ($cinema\_promo$), whose code number (ft_number) is returned. The get_message action get_info and the protocol $\mathsf{yes\_no\_query}_Q$ have already been explained in Section 3. Procedure (b), instead, describes the communicative behavior of the second restaurant: the interaction is similar to the previous case but the restaurant does not take part to the promotion so the customer does not get any free ticket for the cinema. Clause (c) shows how get_info can be implemented as an inform act executed by the service and having as recipient the customer. The question mark amounts to check the value of a fluent in the current state; the semicolon is the sequencing operator of two actions. On the other hand, the cinema protocols are as follows.

(c) $\langle \mathsf{reserv\_cinema\_1}_C(Self, Service, Movie)\rangle\varphi \subset$
$\langle \mathsf{yes\_no\_query}_Q(Self, Service, available(Movie))$;
$\mathbf{B}^{Self} available(Movie)?$;
$\mathsf{yes\_no\_query}_I(Self, Service, cinema\_promo)$;
$\neg\mathbf{B}^{Self} cinema\_promo?$;
$\mathsf{yes\_no\_query}_I(Self, Service, pay\_by(c\_card))$;
$\mathbf{B}^{Self} pay\_by(c\_card)?$;
$\mathsf{inform}(Self, Service, cc\_number)$;
$\mathsf{get\_info}(Self, Service, reservation(Movie))\rangle\varphi$

(d) $\langle \mathsf{reserv\_cinema\_1}_C(Self, Service, Movie)\rangle\varphi \subset$
$\langle \mathsf{yes\_no\_query}_Q(Self, Service, available(Movie))$;
$\mathbf{B}^{Self} available(Movie)?$;
$\mathsf{yes\_no\_query}_I(Service, Self, cinema\_promo)$;
$\mathbf{B}^{Self} cinema\_promo?$;
$\mathsf{inform}(Self, Service, ft\_number)$;
$\mathsf{get\_info}(Self, Service, reservation(Movie))\rangle\varphi$

(e) $\langle \mathsf{reserv\_cinema\_2}_C(Self, WebS, Movie)\rangle\varphi \subset$
$\langle \mathsf{yes\_no\_query}_Q(Self, WebS, available(Movie))$;
$\mathbf{B}^{Self} available(Movie)?$;
$\mathsf{get\_info}(Self, WebS, pay\_by(cash))$;

$\mathsf{get\_info}(Self, WebS, reservation(Movie))\rangle\varphi$

Supposing that the desired movie is available, the first cinema alternatively accepts credit card payments, clause (c), or promotional tickets, clause (d). The second cinema, instead, accepts only cash payments, clause (e).

In the following, the selection and composition tasks, that the personalization agent pa can accomplish, are described. As explained in Section 2.1, these tasks are aimed at refining a previous selection, performed by a retriever. In the example that we are using, when pa will start the personalization task the following list of candidate services will already be available in the DyLOG knowledge base as well as the protocols that they follow.

$\mathbf{B}^{pa} service(restaurant, restaurant1, reserv\_rest\_1_C)$
$\mathbf{B}^{pa} service(restaurant, restaurant2, reserv\_rest\_2_C)$
$\mathbf{B}^{pa} service(cinema, cinema1, reserv\_cinema\_1_C)$
$\mathbf{B}^{pa} service(cinema, cinema2, reserv\_cinema\_2_C)$

The agent will personalize the interaction with the services, according to the requests of the user, dismissing services that do not fit. To this aim, it will reason about the procedures $select\_service$ or $comp\_services$, reported hereafter.

4.2 Web service selection by reasoning about interaction

Web service selection by means of reasoning about a service conversation policy, amounts to answering to the query "Is there a possible conversation among those allowed by the service protocol, after which a condition of interest holds?". In the scenario depicted in Section 2.1, an example is the desire of the user of avoiding credit card payments over the web (for instance when booking a cinema ticket for the movie $akira$). Let us suppose that a preliminary selection has identified *cinema1* and *cinema2*, whose conversation policies are described in the previous section, clauses (c), (d), and (e), as cinemas that show $akira$. A further selection based on *reasoning* can begin by reasoning about the (Prolog-like) procedure $select\_service$:

$\langle select\_service(TypeService, Name, Data)\rangle\varphi \subset$
$\langle \mathbf{B}^{pa} service(TypeService, Name, Protocol)?\ ;\ Protocol(pa, Name, Data)\ \rangle\varphi$

Let us consider the query expressing the fact that the personalization agent wants to select a cinema booking service, that allows a dialogue by which the credit card number of the user is not communicated to the service, that is, the condition that must hold in pa's mental state after the procedure's execution is $\mathbf{B}^{pa}\neg\mathbf{B}^{C} cc\_number$. This condition is a nested belief, that is, a belief about the knowledge of another agent (the cinema service). This is the query:

$\langle select\_service(cinema, C, akira)\rangle\mathbf{B}^{pa}\neg\mathbf{B}^{C} cc\_number)$

C is a variable that ranges over the set of cinema names in the knowledge base. This query, as we will show, succeeds with answer C equal to *cinema2*.

Let us begin with considering *cinema1* ($Protocol$ will be equal to reserv_cinema_1$_C$). This service will be selected if it is possible to answer the query $\langle$reserv_cinema_1$_C(pa,$ $cinema1,\ akira)\rangle \mathbf{B}^{pa}\neg\mathbf{B}^{cinema1} cc\_number$. It is easy to see from the protocol specification that such an interaction is possible, given that the user owns a free ticket. However, let us suppose that this is not the case and that the initial mental state of the agent contains the beliefs: $\mathbf{B}^{pa} cc\_number$ i.e. the agent knows the user's credit card number which is not to be used, $\neg\mathbf{B}^{pa} cinema\_promo$, the user does not have a free ticket, $\neg\mathbf{B}^{pa} pay\_by(c\_card)$, the agent is an authority about the method of payment. Moreover, suppose that pa also has some hypothesis about the knowledge of the cinema services, like $\mathbf{B}^{pa}\neg\mathbf{B}^{C} cc\_number$ (with C varying on $\{cinema1, cinema2\}$), the services do not already know the credit card number. Of course, initially the agent will also believe that no ticket for $akira$ has been booked yet ($\mathbf{B}^{pa}\neg booked(akira)$).

When the agent reasons about the protocol execution, the test $\mathbf{B}^{pa} cinema\_promo?$ fails as well as the subsequent test $\mathbf{B}^{pa} pay\_by(c\_card)?$ fails. As a result, *cinema1* is not selected, and $select\_service$ considers the other option, i.e. *cinema2*. In this case the analogous query succeeds and an execution trace of the protocol reserv_cinema_2$_C$ is returned as a result. It is the following *conditional dialogue plan* between pa and the *cinema2*:

queryIf$(pa, cinema2, available(akira))$;
((**B**$^{pa} Done$(inform$(cinema2, pa, akira))\top?$);
get_info$(pa, cinema2, pay_by(cash))$;
(**B**$^{pa} Done$(inform$(cinema2, pa, pay_by(cash)))\top?$);
get_info$(pa, cinema2, reservation(akira))$;
(**B**$^{pa} Done$(inform$(cinema2, pa, reservation(akira)))\top?) \cup$
(**B**$^{pa} Done$(inform$(cinema2, pa, \neg akira))\top?) \cup$
(**B**$^{pa} Done$(refuseInform$(cinema2, pa, akira))\top?))$

Notice that no communication involves the belief $\mathbf{B}^{pa}\neg\mathbf{B}^{cinema2} cc\_number$, which persists from the initial state; thus, at the end of each execution branch the user's desire of keeping the credit card number secret is satisfied.

4.3 Web service composition by reasoning about interaction

The other task of interest is web service composition. Again, suppose that a preliminary selection has already been accomplished, resulting in the a set of candidate services listed at the end of Section 4.1. In this case, the user wants to book a table at a restaurant plus a cinema entrance profiting of the promotion. For accomplishing this task the personalization agent reasons about the procedure $comp\_services$ (a possible implementation is reported below), that sketches the general composition-

by-sequencing of a set of services, based on their interaction protocols.

$$\langle comp\_services([\,])\rangle\varphi \subset true$$
$$\langle comp\_services([[TypeService, Name, Data]|Services])\rangle\varphi \subset$$
$$\quad \langle \mathbf{B}^{pa} service(TypeService, Name, Protocol)? \;;$$
$$\quad Protocol(pa, Name, Data) \;; comp\_services(Services)\rangle\varphi$$

Intuitively, $comp\_services$ builds a sequence of protocols so that it will be possible to reason about them as a whole. Let us now consider the query:

$$\langle comp\_services([[restaurant, R, dinner], [cinema, C, akira]])\rangle$$
$$\quad (\mathbf{B}^{pa} cinema\_promo \wedge \mathbf{B}^{pa} reservation(dinner) \wedge$$
$$\quad \mathbf{B}^{pa} reservation(akira) \wedge \mathbf{B}^{pa} \neg \mathbf{B}^{C} cc\_number \wedge \mathbf{B}^{pa} \mathbf{B}^{C} ft\_number)$$

that amounts to determine if it is possible to compose the interaction with a restaurant and a cinema web services, reserving a table for dinner ($\mathbf{B}^{pa} reservation(dinner)$) and booking a ticket for the movie $akira$ ($\mathbf{B}^{pa} reservation(akira)$), exploiting a promotion ($\mathbf{B}^{pa} cinema\_promo$). Credit card is not to be used ($\mathbf{B}^{pa} \neg \mathbf{B}^{C} cc\_number$), the free ticket is to be spent ($\mathbf{B}^{pa} \mathbf{B}^{C} ft\_number$). The agent initial mental state contains beliefs about the user's credit card number ($\mathbf{B}^{pa} cc\_number$), the desire to avoid using it ($\neg \mathbf{B}^{pa} pay\_by(credit\_card)$), and the fact that the agent is an authority about the form of payment. Further assumptions are that no reservation for dinner nor for the movie has been taken yet, $\mathbf{B}^{pa} \neg reservation(dinner)$ and $\mathbf{B}^{pa} \neg reservation(akira)$, that *pa* does not have a free ticket for the cinema, $\neg \mathbf{B}^{pa} cinema\_promo$, and some hypothesis on the interlocutor's mental state, e.g. the belief fluent $\mathbf{B}^{pa} \neg \mathbf{B}^{C} cc\_number$ (with C in $\{cinema1, cinema2\}$), meaning that the cinema booking services do not already know the credit card number. In this context, the query succeeds with answer R equal to *restaurant1* and C equal to *cinema1*, and the agent builds the following execution trace of $comp\_services$ ($[[restaurant, restaurant1, dinner], [cinema, cinema1, akira]]$):

$\mathsf{queryIf}(pa, restaurant1, available(dinner))$;
$\boxed{\mathsf{inform}(restaurant1, pa, available(dinner)) \;;}$
$\mathsf{inform}(restaurant1, pa, reservation(dinner))$;
$\mathsf{inform}(restaurant1, pa, cinema\_promo)$;
$\mathsf{inform}(restaurant1, pa, ft\_number)$;
$\mathsf{queryIf}(pa, cinema1, available(akira))$;
$\boxed{\mathsf{inform}(cinema1, pa, available(akira)) \;;}$
$\mathsf{queryIf}(cinema1, pa, cinema\_promo)$;
$\mathsf{inform}(pa, cinema1, cinema\_promo)$;
$\mathsf{inform}(pa, cinema1, ft\_number)$;
$\mathsf{inform}(cinema1, pa, reservation(akira))$

This is a dialogue plan that is made of a conversation between pa and $restaurant1$ followed by one between pa and $cinema1$, instances of the respective conversation protocols, after which the desired condition holds. The linear plan, will, lead to the

desired goal given that some *assumptions* (above outlined with a box) about the provider's answers hold. For instance, that there is a free seat at the cinema, a fact that can be known only at execution time. Assumptions occur when the interlocutor can respond in different ways depending on its internal state. It is not possible to know in this phase which the answer will be, but since the set of the possible answers is given by the protocol, it is possible to identify the subset that leads to the goal. In the example they are answers foreseen by a yes_no_query protocol. Returning such assumptions to the designer is also very important to understand the correctness of the implementation w.r.t. the chosen speech act ontology.

5 Conclusions and related works

The work presented in this article is set in the Semantic Web field of research and faces some issues related to web service selection and composition. The basic idea is to consider a service as a software agent and the problem of composing a set of web services as the problem of making a set of software agents cooperate within a multiagent system (or MAS). This interpretation is, actually, quite natural, although somewhat new to the web service community, with a few exceptions [34,35]. In particular, we have studied the possible benefits provided by the introduction of an explicit (and declarative) description of the communicative behavior of the web services in terms of *personalization* of the service fruition and of the *composition* of a set of services. Indeed, a web service must follow some possibly non-deterministic procedure aimed at getting/supplying all the necessary information. So far, however, standard languages for *semantic* web service description do not envision the possibility of separating the communicative behavior from the rest of the description. On the contrary, communication plays a central role in languages for describing workflows and business processes, like BPEL. Unfortunately, such languages do not have a formalization that enables the automation, based on reasoning, of the tasks that are the focus of this work [36,37].

The idea of setting web service selection and composition in the Semantic Web rather than in the WWW (that is of representing web services according to a formal model that enables automated forms of reasoning), is motivated by a growing agreement, in the web service research community, on the need of finding representation models that abstract away from the specific languages (like BPEL) used to specify business processes, and which allow the analysis of properties of the interactions that are executed [16]. The introduction of choreographies as global schemas of interaction (and of languages like WS-CDL [17]) as well as the distinction between this global view from the local perspective of the single services are a first consequence of such needs. At the moment of writing, in the Semantic Web area no concept equivalent (or at least close to) choreographies has been defined yet, nevertheless, the languages proposed for web service representation share with BPEL the characteristic of representing the local view of the interaction, that each

single process should have. The interesting point is that such languages have been designed expressly with the aim of supplying abstract, formal representations. Another general agreement is on the principle that the richer the semantic information that is used, the higher the precision of the answer to a query, where *precision* is the extent to which only the items "really of interest" are returned [38]. Depending on the kind of resources and of tasks of interest it is necessary to identify the right piece of information to use. In the approach that we have proposed the user's goals and needs have this role, and to be matched, they require the rational inspection of the interactive behavior of the services, which consequently is to be represented in an explicit way.

The approach, proposed in this paper for performing the inspection, is based on techniques for reasoning about actions and change, in which the communicative behavior of the services is modelled as a complex action (a procedure) based on speech acts. In this framework the selection and composition tasks can be interpreted as the process of answering the questions "Is it possible to interact with this service in such a way that this condition will hold afterwards?" and "Is it possible to execute these services in sequence in such a way that this condition will hold afterwards?". Of course the effective achievement of the goal might depend on conditions, that are necessary to the completion of the interaction but that will be known only at execution time. For instance, in order to book a table at a restaurant it is necessary that there is at least one free table. We have not tackled this problem yet, although it would be very interesting to integrate in the approach that we have proposed a mechanism for dealing with *failure* (at execution time) and *replanning*. This form of reasoning is necessary to arrive to real applications and it could take into account also degrees of preference explicitly expressed by the user. Such criteria could be used to achieve a greater flexibility, by relaxing some of the constraints in case no plan can be found or when a plan execution fails. In particular, in our opinion it would be extremely interesting to study *integrations* of replanning techniques with *compensation techniques*. Indeed, in the research community that studies planning the interest is mainly posed on the efficiency of the planner and replanning, if any, does not exploit any explicit representation of actions for undoing things, with a few exceptions [39]. Given the current state, whatever it is, forward actions are searched for reaching the goal. The process might have, as a side effect, the undoing of some partial achievement but this will be an occasional outcome. The main achievement of the proposed approach, however, is that the presence of public protocol/policy specifications gives the agent the possibility to *set up* plans. Even though this plan might be subject to assumptions (e.g. that a table will be available), the agent will *know in advance* if it is worthwhile to interact with that partner. In a way, the approach that has been proposed can be seen as a sieve that allows agents to brush off a number of partners before the interaction. Nevertheless, it is something more than a sieve because it also allows the identification of *courses of interaction* that the agent is willing to perform. In this perspective, it is actually a valuable tool for personalizing the interaction with web services according to the goals of the user.

Our proposal can be considered as an approach based on the process ontology, a *white box* approach in which part of the behavior of the services is available for a rational inspection. In this case, the deductive process exploits more semantic information: in fact, it does not only take into account the pre- and post-conditions, as above, it also takes into account the complex behavior (the communicative behavior) of the service. The idea of focussing on abstract descriptions of the communicative behavior is, actually, a novelty also with respect to other proposals that are closer to the agent research community and more properly set in the Semantic Web research field. The work that is surely the closest to ours is the one by the DAML-S (now OWL-S) coalition, that designed the language OWL-S [40]. An OWL-S service description has three conceptual levels: the profile, used for advertising and discovery, where inputs, outputs, preconditions and effects are enumerated, the process model, a declarative description of the structure of a service, and the grounding, that describes how an agent can access the service by means of low-level (SOAP) messages. To our aims, the most interesting component of an OWL-S web service description is the process model, which describes a service as atomic, simple (viewed as atomic) or composite, in a way inspired by the language GOLOG and its extensions [30,28,41,40]. However, to our knowledge, no approach for reasoning about the process model has been proposed yet. In the works by McIlraith et al. (e.g. [42]), indeed the most relevant for us, the idea is always to compose services that are viewed as *atomic*, and the composition is merely based on their preconditions and effects, exploiting techniques derived from the Situation Calculus. In particular, the precondition and effect, input and output lists are flat; no relation among them can be expressed, so it is impossible to understand if a service can follow alternative courses of interaction, among which one could choose. Indeed, the advantage of working at the protocols level is that by reasoning about protocols agents can personalize the interaction by selecting a course that satisfies user- (or service-) given requirements. This process can be started before the actual interaction takes place.

Of course, other approaches to matchmaking have been proposed and are being investigated. For instance, the frame-based approaches, like the UDDI registry service for WSDL web services [13]. In this case two sets of textual property values are compared, a service description and a query (i.e. the description of a desired service); both descriptions are based on partially pre-enumerated vocabularies of service types and properties. Close to frame-based approaches is the set-based modelling approach proposed in the WSMO service discovery. In this case the sets of objects to be compared are the set of user's goals (intended as the information that the user wants to receive as output) and the set of objects delivered after a web service execution. The two sets might be inter-related in some way by ontologies, which are considered as domain models. Actually, four different degrees of match are formalized, depending on additional properties that should hold and a complete formalization based on description logics is explained [43]. A refinement of this proposal, in which more elaborated descriptions are taken into account, is proposed as a second step. At this higher level of formalization, services are considered as

relations on an abstract state-space and are described in terms of inputs, outputs, preconditions, effects, assumptions, and post-conditions; in this extension exploits of interest are identified by applying transaction logics. Another category is that of deductive retrieval (also known as "reuse by contract"), well described in [44]. Here web services, interpreted as software components, are seen as black-boxes, whose description relies on the concept of Abstract Data Type (or ADT). The semantics of an ADT is given by a set of logic axioms. Part of the approaches in this family (the "plug in match" approaches [45]) use these logic axioms to identify the pre- and post-conditions to the execution of the web service. In this case also queries are represented by a set of pre- and post-conditions. The decision of whether a service matches a given query depends on the truth value of the formula $(pre_Q \supset pre_{WS}) \cap (post_Q \supset post_{WS})$, where pre_Q and $post_Q$ are the pre- and post-conditions of the query and pre_{WS} and $post_{WS}$ are the pre- and post-conditions of the service. Many works should be cited in this line of research, like NORA/HAMRR [46], feature-based classification [47], LARKS [48], SWS matchmaker [49], up to PDDL-based languages (PDDL stands for "Planning Domain Definition Framework" [50]) like the proposal in [51]. To our knowledge none of these works is based on reasoning on an explicit representation of the interactive behavior of the services, even in the case in which PDDL is used, the idea is to consider web services as atomic actions.

References

[1] W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, H. M. W. Verbeek, P. Wohed, Life after BPEL?, in: Proc. of WS-FM'05, Vol. 3670 of LNCS, Springer, 2005, pp. 35–50, invited speaker.

[2] M. Baldoni, C. Baroglio, N. Henze, Personalization for the Semantic Web, in: Reasoning Web, Vol. 3564 of LNCS Tutorial, Springer, 2005, pp. 173–212.

[3] G. Antoniou, M. Baldoni, C. Baroglio, R. Baungartner, F. Bry, T. Eiter, N. Henze, M. Herzog, W. May, V. Patti, S. Schaffert, R. Schidlauer, H. Tompits, Reasoning methods for personalization on the semantic web, Annals of Mathematics, Computing and Teleinformatics (AMCT) 2 (1) (2004) 1–24.

[4] F. Dignum, M. Greaves, Issues in agent communication, in: Issues in Agent Communication, Vol. 1916 of LNCS, Springer, 2000, pp. 1–16.

[5] F. Dignum (Ed.), Advances in agent communication languages, Vol. 2922 of LNAI, Springer-Verlag, 2004.

[6] FIPA, Communicative act library specification, Tech. rep., FIPA (Foundation for Intelligent Physical Agents) (2002).

[7] T. Finin, Y. Labrou, J. Mayfield, KQML as an Agent Communication Language, in: J. Bradshaw (Ed.), Software Agents, MIT Press, 1995.

[8] M. Baldoni, L. Giordano, A. Martelli, 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) (2004) 207–257.

[9] M. Baldoni, C. Baroglio, V. Patti, Web-based adaptive tutoring: an approach based on logic agents and reasoning about actions, Artificial Intelligence Review 22 (1) (2004) 3–39.

[10] A. Mamdani, J. Pitt, Communication protocols in multi-agent systems: A development method and reference architecture, in: Issues in Agent Communication, Vol. 1916 of LNCS, Springer, 2000, pp. 160–177.

[11] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, Reasoning about self and others: communicating agents in a modal action logic, in: Proc. of ICTCS'2003, Vol. 2841 of LNCS, Springer, 2003, pp. 228–241.

[12] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, Reasoning about interaction protocols for web service composition, in: Bravetti and Zavattaro [15], pp. 21–36, vol. 105 of Electronic Notes in Theoretical Computer Science.

[13] A. Barros, M. Dumas, P. Oaks, A critical overview of the web services choreography description language(ws-cdl), Business Process TrendsHttp://www.bptrends.com.

[14] BPEL4WS, http://www-106.ibm.com/developerworks/library/ws-bpel (2003).

[15] M. Bravetti, G. Zavattaro (Eds.), Proc. of the 1st Int. Workshop on Web Services and Formal Methods (WS-FM 2004), Elsevier Science Direct, 2004, vol. 105 of Electronic Notes in Theoretical Computer Science.

[16] M. Bravetti, L. Kloul, G. Zavattaro (Eds.), Proc. of the 2nd International Workshop on Web Services and Formal Methods (WS-FM 2005), no. 3670 in LNCS, Springer, 2005.

[17] WS-CDL, http://www.w3.org/tr/2004/wd-ws-cdl-10-20041217/ (2004).

[18] T. Berners-Lee, J. Hendler, O. Lassila, The Semantic Web, Scientific American.

[19] R. N. of Excellence, http://www.rewerse.org (2004).

[20] G. Antoniou, F. van Harmelen, A semantic web primer, MIT Press, 2004.

[21] WSMO, http://www.wsmo.org/ (2005).

[22] J. H. Odell, H. Van Dyke Parunak, B. Bauer, Representing agent interaction protocols in UML, in: Agent-Oriented Software Engineering, Springer, 2001, pp. 121–140, http://www.fipa.org/docs/input/f-in-00077/.

[23] F. for InteroPerable Agents, Fipa modeling: Interaction diagrams, Tech. rep., working Draft Version 2003-07-02 (2003).

[24] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, C. Schifanella, Verifying the conformance of web services to global interaction protocols: a first step, in: Proc. of 2nd Int. Workshop on Web Services and Formal Methods, WS-FM 2005, no. 3670 in LNCS, 2005, pp. 257–271.

[25] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, G. Zavattaro, Choreography and Orchestration: a synergic approach for system design, in: Proc. the 3rd Int. Conf. on Service Oriented Computing, 2005.

[26] K. Arisha, T. Eiter, S. Kraus, F. Ozcan, R. Ross, V. Subrahmanian, IMPACT: a platform for collaborating agents, IEEE Intelligent Systems 14 (2) (1999) 64–72.

[27] M. Fisher, A survey of concurrent METATEM - the language and its applications, in: D. Gabbay, H. Ohlbach (Eds.), Proc. of the 1st Int. Conf. on Temporal Logic, ICTL '94, Vol. 827 of LNAI, Springer-Verlag, 1994, pp. 480–505.

[28] G. D. Giacomo, Y. Lesperance, H. Levesque, Congolog, a concurrent programming language based on the situation calculus, Artificial Intelligence 121 (2000) 109–169.

[29] J. Leite, A. Omicini, P. Torroni, P. Yolum (Eds.), Int. Workshop on Declarative Agent Languages and Technology, New York City, NY, USA, 2004, http://centria.di.fct.unl.pt/~jleite/dalt04.

[30] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, R. B. Scherl, GOLOG: A Logic Programming Language for Dynamic Domains, J. of Logic Programming 31 (1997) 59–83.

[31] FIPA, Fipa 2000, fipa Query Interaction Protocol Specification, Tech. rep., FIPA (Foundation for Intelligent Physical Agents) (November 2000).

[32] K. Eshghi, R. Kowalski, Abduction compared with Negation by Failure, in: Proc. of ICLP '89, The MIT Press, Lisbon, 1989.

[33] M. Denecker, D. De Schreye, Representing Incomplete Knowledge in Abduction Logic Programming, in: Proc. of ILPS '93, The MIT Press, Vancouver, 1993.

[34] J. Bryson, D. Martin, S. McIlraith, L. A. Stein, Agent-based composite services in DAML-S: The behavior-oriented design of an intelligent semantic web (2002).
URL `citeseer.nj.nec.com/bryson02agentbased.html`

[35] K. Sycara, Brokering and matchmaking for coordination of agent societies: A survey, in: A. O. et al. (Ed.), Coordination of Internet Agents, Springer, 2001.

[36] D. J. Mandell, S. A. McIlraith, Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation, in: Proc. of the 2nd Int. Semantic Web Conference (ISWC2003), Sanibel Island, Florida, 2003.

[37] J. Koehler, B. Srivastava, Web service composition: Current solutions and open problems, in: ICAPS 2003 Workshop on Planning for Web Services, 2003, pp. 28–35.

[38] M. Klein, A. Bernstein, Searching for services on the semantic web using process ontologies, in: Proc. of the Int. Semantic Web Working Symposium (SWWS), Stanford University, California (USA), 2001.

[39] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, P. Traverso, Planning and monitoring web service composition, in: Proc. of Planning and Monitoring Web Service Composition Artificial Intelligence: Methodology, Systems, Application (AIMSA) 2004, 2004.

[40] OWL-S, http://www.daml.org/services/owl-s/1.1/ (2004).

[41] S. McIlraith, T. Son, Adapting Golog for Programmin the Semantic Web, in: 5th Int. Symp. on Logical Formalization of Commonsense Reasoning, 2001, pp. 195–202.

[42] S. Narayanan, S. A. McIlraith, Simulation, verification and automated composition of web services, Honolulu, Hawaii (USA), 2002, pp. 77–88.

[43] U. Keller, R. L. A. Polleres, I. Toma, M. Kifer, D. Fensel, D5.1 v0.1 wsmo web service discovery, Tech. rep., WSML deliverable (2004).

[44] J. Peer, Towards automatic web service composition using ai planning techniques, http://www.mcm.unisg.ch/org/mcm/web.nsf/staff/jpeer (2003).

[45] A. M. Zamreski, J. M. Wing, Specification matching of software components, in: Proc. of the 3rd ACM SIGSOFT Symp. on the Foundations of Software Eng., 1995.

[46] B. Fischer, J. Schumann, NORA/HAMRR: Making deduction-based software component retrieval practical, in: Proc. of CADE-14 Workshop on Automated Theorem Proving in Software Engineering, 1997.

[47] J. Penix, P. Alexander, Efficient specification-based component retrieval, Automated Software Engineering.

[48] K. Sykara, S. Widoff, M. Klusch, J. Lu, LARKS: dynamic matchmaking among heterogeneous software agents in cyberspace, Autonomous Agents and Multi-Agent Systems.

[49] R. M. V. Haarslev, Description of the racer system and its applications, Stanford, California (USA), 2001.

[50] D. MacDermott, AI planning systems competition, AI Magazine 21 (2) (2000) 35–55.

[51] J. Peer, M. Vokovic, A proposal for a semantic web service description format, in: Proc. of the European Conf. on Web Services (ECOWS'04), Springer-Verlag, 2004.

[52] V. Patti, Programming Rational Agents: a Modal Approach in a Logic Programming Setting, Ph.D. thesis, Dipartimento di Informatica, Università degli Studi di Torino, Torino, Italy, available at `http://www.di.unito.it/~patti/` (2002).

A Goal directed proof procedure for DyLOG

This appendix presents the proof procedure used to build linear plans, making assumptions on sensing actions and on external communicative actions. Then, a variant that builds conditional plans is introduced, where all the possible values returned by sensing and by incoming communications are taken into account. For the sake of brevity, we do not report in this paper the demonstrations. Actually, these are very similar to those for DyLOG without communication kit, see [8,52] for details.

1) $$\frac{\overline{a}_{1\cdots m} \vdash \langle p'_1; \ldots; p'_{n'}; \overline{p}_{2\ldots n}\rangle Fs \text{ w. a. } \sigma}{\overline{a}_{1\cdots m} \vdash \langle p; \overline{p}_{2\ldots n}\rangle Fs \text{ w. a. } \sigma}$$ where $p \in \mathbf{P}$ and $\langle p\rangle\varphi \subset \langle p'_1; \ldots; p'_{n'}\rangle\varphi \in \Pi_{\mathbf{P}} \cup \Pi_{\mathbf{CP}}$

2) $$\frac{\overline{a}_{1\cdots m} \vdash Fs' \;\; \overline{a}_{1\cdots m} \vdash \langle \overline{p}_{2\ldots n}\rangle Fs \text{ w. a. } \sigma}{\overline{a}_{1\cdots m} \vdash \langle (Fs')?; \overline{p}_{2\ldots n}\rangle Fs \text{ w. a. } \sigma}$$

3) $$\frac{\overline{a}_{1\cdots m} \vdash Fs' \;\; \overline{a}_{1\cdots m}, a \vdash \langle \overline{p}_{2\ldots n}\rangle Fs \text{ w. a. } \sigma}{\overline{a}_{1\cdots m} \vdash \langle a; \overline{p}_{2\ldots n}\rangle Fs \text{ w. a. } \sigma}$$ where $a \in \mathbf{A} \cup \mathbf{C}$, and $\Box(Fs' \supset \langle a\rangle\top) \in \Pi_{\mathbf{A}} \cup \Pi_{\mathbf{C}}$

4) $$\frac{\overline{a}_{1\cdots m} \vdash \langle s^{\mathbf{B}^{ag_i} l}; \overline{p}_{2\ldots n}\rangle Fs \text{ w. a. } \sigma}{\overline{a}_{1\cdots m} \vdash \langle s; \overline{p}_{2\ldots n}\rangle Fs \text{ w. a. } \sigma}$$ where $s \in \mathbf{S}$ and $l \in dom(s)$

5) $$\frac{\overline{a}_{1\cdots m} \vdash \langle c; \overline{p}_{2\ldots n}\rangle Fs \text{ w. a. } \sigma}{\overline{a}_{1\cdots m} \vdash \langle g; \overline{p}_{2\ldots n}\rangle Fs \text{ w. a. } \sigma}$$ where $g \in \mathbf{S}get$ and $[g]\varphi \equiv [\bigcup_{c\in\mathbf{C}_g} c]\varphi$

6) $$\frac{\overline{a}_{1\cdots m} \vdash Fs}{\overline{a}_{1\cdots m} \vdash \langle\varepsilon\rangle Fs \text{ w. a. } \sigma}$$ where $\sigma = a_1; \ldots; a_m$

Fig. A.1. Rules (1-6) of the goal directed proof procedure for DyLOG. $\overline{a}_{1\cdots m}$ and $\overline{p}_{2\cdots n}$ stands for $a_1, \ldots, a_m$ and $p_2, \ldots, p_n$, respectively. l denotes a fluent literal or a done fluent while L denotes l or a belief fluent of rank 1.

A.1 Linear plan extraction

A query (see Section 3.4) of the form $\langle p_1; p_2; \ldots; p_n\rangle Fs$ succeeds if it is possible to execute in the given order p_1, p_2, ..., p_n, starting from the current state, in such a way that Fs holds in the resulting belief state. Since a state can be represented by the sequence of atomic actions performed for reaching it, in general, we write:

$$a_1, \ldots, a_m \vdash \langle p_1; p_2; \ldots; p_n\rangle Fs \text{ with answer (w.a.) } \sigma$$

where $a_1, \ldots, a_m$ represents the current state, to mean that the query can be proved with answer σ in the current state and from the domain description $(\Pi, \mathsf{CKit}^{ag_i}, S_0)$. The answer σ is an execution trace a_m, ... a_{m+k} of $p_1, \ldots, p_n$ in the current state. We denote by ε the initial mental state.

The first part of the proof procedure, rules (1–6) in Fig. A.1, deals with the execution of complex actions, sensing actions, primitive actions and test actions. The proof procedure reduces the complex actions in the query to a sequence of primitive actions and test actions, then it verifies if the execution of primitive actions is possible and if test actions are successful. To do this, it reasons about the execution of a sequence of primitive actions from the initial state and computes the values of fluents at different states. During a computation, a state is represented by a sequence of primitive actions $a_1, \ldots, a_m$. The value of fluents at a state is not explicitly recorded but it is computed when needed. The second part of the procedure, rules (7–14), allows the values of mental fluents in an agent ag_i state to be determined.

6) $\dfrac{\overline{a}_{1\cdots m} \vdash Fs}{\overline{a}_{1\cdots m} \vdash \langle\varepsilon\rangle Fs \text{ w. a. } \sigma}$ where $\sigma = a_1; \ldots; a_m$

7) $\dfrac{}{\overline{a}_{1\cdots m} \vdash \top}$

8a) $\dfrac{\overline{a}_{1\cdots m-1} \vdash Fs'}{\overline{a}_{1\cdots m} \vdash F}$ where $m > 0$ and $\Box(Fs' \supset [a_m]F) \in \Pi_{\mathbf{A}}$

8b) $\dfrac{}{\overline{a}_{1\cdots m} \vdash F}$ if $a_m = s^F$

8c) $\dfrac{\mathbf{not}\ \overline{a}_{1\cdots m} \vdash \neg F \quad \overline{a}_{1\cdots m-1} \vdash F}{\overline{a}_{1\cdots m} \vdash F}$ where $m > 0$

8d) $\dfrac{}{\varepsilon \vdash F}$ if $F \in S_0$

9) $\dfrac{\overline{a}_{1\cdots m} \vdash Fs' \quad \overline{a}_{1\cdots m} \vdash Fs''}{\overline{a}_{1\cdots m} \vdash Fs' \wedge Fs''}$

10) $\dfrac{\overline{a}_{1\cdots m} \vdash \mathbf{B}^{ag_i} L}{\overline{a}_{1\cdots m} \vdash \mathbf{M}^{ag_i} L}$

11) $\dfrac{\overline{a}_{1\cdots m} \vdash \mathbf{B}^{ag_i} l}{\overline{a}_{1\cdots m} \vdash \mathbf{B}^{ag_i}\mathbf{B}^{ag_i} l}$ 11') $\dfrac{\overline{a}_{1\cdots m} \vdash \mathbf{M}^{ag_i}\mathbf{M}^{ag_i} l}{\overline{a}_{1\cdots m} \vdash \mathbf{M}^{ag_i} l}$

12) $\dfrac{\overline{a}_{1\cdots m} \vdash \mathbf{M}^{ag_i} l}{\overline{a}_{1\cdots m} \vdash \mathbf{B}^{ag_i}\mathbf{M}^{ag_i} l}$ 12') $\dfrac{\overline{a}_{1\cdots m} \vdash \mathbf{M}^{ag_i}\mathbf{B}^{ag_i} l}{\overline{a}_{1\cdots m} \vdash \mathbf{B}^{ag_i} l}$

13) $\dfrac{\overline{a}_{1\cdots m} \vdash Done(a)\top}{\overline{a}_{1\cdots m} \vdash \mathbf{B}^{ag_i} Done(a)\top}$ 14) $\dfrac{}{\overline{a}_{1\cdots m} \vdash Done(a_m)\top}$

Fig. A.2. Rules (7-14) of the goal directed proof procedure for DyLOG. $\overline{a}_{1\cdots m}$ and $\overline{p}_{2\cdots n}$ stands for $a_1, \ldots, a_m$ and $p_2, \ldots, p_n$, respectively. l denotes a fluent literal or a done fluent while L denotes l or a belief fluent of rank 1.

Let us briefly comments the rules. To execute a *complex action* p the modality $\langle p\rangle$ is non-deterministically replaced with the modality in the antecedent of a suitable axiom, rule (1). To execute a *test action* $(Fs)?$, the value of Fs is checked in the current state; if Fs holds, the test action is eliminated otherwise the computation fails, rule (2). To execute a *primitive action* a, first precondition laws are checked to verify if the action is possible. If they hold, the computation moves to a new state in which the action has been performed, rule (3). To execute a *sensing action* s, rule (4), we non-deterministically replace it with one of the primitive actions which define it, that, when it is executable, will cause $\mathbf{B}^{ag_i} l$ and $\mathbf{B}^{ag_i}\neg l'$, for each $l' \in dom(s)$, with $l \neq l'$. Rule (5) deals with get_message actions: a get_message *action* g is non-deterministically replaced with one of the external communicative actions which define it.

Rule (6) deals with the case when no more actions are to be executed. The desired sequence of primitive actions has already been determined so, to check if Fs is true after it, rules (7-14) in Fig. A.2 are used. An *epistemic fluent* F holds at a state $a_1, \ldots, a_m$ if either F is an immediate effect of action a_m (rule 8a); or action a_m is a primitive action s^F (introduced to model the sensing action s), whose effect is to add F to the state (rule 8b); or F holds in the previous state $a_1, \ldots, a_{m-1}$ and

$$\text{4-bis)}\quad \frac{\forall l_k \in \mathbf{F},\ \overline{a}_{1\cdots m} \vdash \langle s^{\mathbf{B}^{ag_i} l}; \overline{p}_{2\ldots n}\rangle Fs \text{ w. a. } a_1;\ldots;a_m; s^{\mathbf{B}^{ag_i} l}; \sigma'_k}{\overline{a}_{1\cdots m} \vdash \langle s; \overline{p}_{2\ldots n}\rangle Fs \text{ w. a. } a_1;\ldots;a_m; s; (\bigcup_{k=1\ldots t}(\mathbf{B}^{ag_i} l_k?); \sigma'_k)}$$

$$\text{5-bis)}\quad \frac{\forall c_k \in \mathbf{C}_g,\ \overline{a}_{1\cdots m} \vdash \langle c_k; \overline{p}_{2\ldots n}\rangle Fs_i \text{ w. a. } a_1;\ldots;a_m; c_k; \sigma'_k}{\overline{a}_{1\cdots m} \vdash \langle g; \overline{p}_{2\ldots n}\rangle Fs_i \text{ w. a. } a_1;\ldots;a_m; g; (\bigcup_{k=1\ldots t}(\mathbf{B}^{ag_i} Done(c_k)\top?); \sigma'_k)}$$

Fig. A.3. A variant of the proof procedure for extracting conditional plans. In (4-bis) $s \in \mathbf{S}$ and $\mathbf{F} = \{l_1, \ldots, l_t\} = dom(s)$; in (5-bis) $g \in \mathbf{S}$ and $\{c_1, \ldots, c_t\}$= $\mathbf{C}_g$.

it persists after a_m, rule (8c); or $a_1, a_2, \ldots, a_m$ is the initial state and F already holds in it, rule (8d). Notice that rule (8c) allows to deal with the *frame problem*: F persists from a state to the next one unless the executed action a_m makes $\neg F$ true, i.e. it persists if $\neg F$ fails from $a_1, a_2, \ldots, a_m$. In this rule $\mathbf{not}$ represents *negation as failure*. Rule (9) deals with *conjunction*. Rule (10) allows $\mathbf{M}^{ag_i} l$ to be concluded from $\mathbf{B}^{ag_i} l$, this is justified by the property of seriality of the belief modality. Rules (11) and (11') have been introduced for coping with *transitivity of beliefs*. Rules (12) and (12') tackle their *euclideaness*. Rules (13) and (14) have been introduced to provide *awareness* of the action execution.

Under the assumption of *e-consistency*, i.e. for every set of action laws for a given action which may be applied in the same state, the set of their effects is consistent [33], of the domain description and of consistency of the initial situation, the proof procedure is *sound* w.r.t. the non-monotonic semantics. First, it is necessary to show that the proof procedure is sound and complete w.r.t. the monotonic Kripke semantics; then, it is possible to show the soundness of the non-monotonic part.

Finally, let $\langle p_1; \ldots; p_n\rangle Fs$ be an existential query and σ the answer returned by one of its successful derivations. It is possible to show that σ is effectively an execution trace of $p_1; \ldots; p_n$, that is, given a domain description, $\langle\sigma\rangle Fs \supset \langle p_1; \ldots; p_n\rangle Fs$. Moreover, σ is a *legal* sequence of atomic actions, it can actually be executed, and it always leads to a state in which Fs holds, i.e. the $\langle\sigma\rangle\top$ and $[\sigma]Fs$ hold.

A.2 Building conditional plans

Let us now introduce a variant of the proof procedure presented above which, given a query $\langle p_1; p_2; \ldots; p_n\rangle Fs$, computes a *conditional plan* σ. All the executions in σ are possible behaviors of the sequence $p_1; p_2; \ldots; p_n$. The new proof procedure is obtained by replacing rules (4) and (5) in Fig. A.1 (to handle sensing actions and get message actions, respectively) with rules (4-bis) and (5-bis) in Fig. A.3. As a difference with the previous case, when a sensing action is executed, the procedure now considers all the possible outcomes of the action, so that the computation splits in more branches. The resulting plan will contain a branch for each value that leads to success. The same holds for the get_message actions, which indeed are treated as a special case of sensing.

Interaction Protocols and Capabilities: A Preliminary Report*

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

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

Abstract. A typical problem of the research area on Service-Oriented Architectures is the composition of a set of existing services with the aim of executing a complex task. The selection and composition of the services are based on a description of the services themselves and can exploit an abstract description of their interactions. Interaction protocols (or choreographies) capture the interaction as a whole, defining the rules that entities should respect in order to guarantee the interoperability; they do not refer to specific services but they specify the roles and the communication among the roles. Policies (behavioral interfaces in web service terminology), instead, focus on communication from the point of view of the individual services. In this paper we present a preliminary study aimed to allow the use of public choreography specifications for generating executable interaction policies for peers that would like to take part in an interaction. Usually the specifications capture only the interactive behavior of the system as a whole. We propose to enrich the choreography by a set of *requirements* of capabilities that the parties should exhibit, where by the term "capability" we mean the skill of doing something or of making some condition become true. Such capabilities have the twofold aim of connecting the interactive behavior to be shown by the role-player to its internal state and of making the policy executable. A possible extension of WS-CDL with capability requirements is proposed.

1 Introduction

In various application contexts there is a growing need of being able to compose a set of heterogeneous and independent entities with the general aim of executing a task, which cannot be executed by a single component alone. In an application framework in which components are developed individually and can be based on

* 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.

J.J. Alferes et al. (Eds.): PPSWR 2006, LNCS 4187, pp. 63–77, 2006.

various technologies, it is mandatory to find a flexible way for glueing components. The solution explored in some in some research areas, like web services (WS) and multi-agent systems (MAS), is to compose entities based on dialogue. In web services the language WS-BPEL [20] has become the *de facto* standard for building executable composite services on top of already existing services by describing the flow of information in terms of exchanged messages. On the other hand, the problem of aggregating communicating agents into (open) societies is well-known in the research area about MASs, where a lot of attention has been devoted to the issues of defining interaction policies, verifying the interoperability of agents based on dialogue, and checking the conformance of policies w.r.t. global communication protocols [28,17,11].

As observed in [27,5], the MAS and WS research areas show convergences in the approach by which systems of agents, on a side, and composite services, on the other, are designed, implemented and verified. In both cases it is in fact possible to distinguish two levels. On the one hand we have a global view of the system as a whole, which is independent from the specific agents/services which will take part to the interaction (the design of the system). In the case of MASs [14] the design level often corresponds to a shared *interaction protocol* (e.g. represented in AUML [21]). In the case of web services this level corresponds to a *choreography* of the system (e.g. expressed in WS-CDL). In general, at this level a set of *roles*, which will be played by some peers, are defined. On the other hand we have the level concerning the implementation of the policies of the entities that will play the roles. These interactive behaviors must be given in some executable language, e.g. WS-BPEL in the case of web services.

In this proposal, we consider choreographies as *shared knowledge* among the parties. We will, then, refer to them as to *public* and non-executable specifications. The same assumption cannot be made about the interactive behavior of specific parties (be they services or agents). The behavior of a peer will be considered as being private, i.e. non-transparent from outside. Nevertheless, if we are interested in coordinating the interaction of a set of parties as specified by a given choreography, we need to *associate* parties to roles. Suppose that a service publishes the fact that it acts according to the role "seller" of a public choreography. In order to interact with that service it will be necessary to play another role, e.g. "customer", of the specified choreography, but for playing it, the service interactive behavior must *conform* to the specification given by the role [1,13,3]. *Checking* the *conformance* is a way for guaranteeing that the service can interact with services playing the other roles in the choreography [3].

Let us, now, suppose that a peer does not have a conformant policy for playing a certain role, but that is needs to take part to the interaction ruled by the choreography anyway. A possible solution is to define a method for generating, in an automatic way, a conformant policy from the role specification. The role specification, in fact, contains all the necessary information about what sending/receiving to/from which peer at which moment. As a first approximation, we can, then, think of translating the role as expressed in the specification language in a policy (at least into a policy *skeleton*) given in an executable language.

This is, however, not sufficient. In fact, it is necessary to bind the interactive (observable) behavior that is encoded by the role specification with the internal (unobservable) behavior that the peer must anyway have and with its internal state. For instance, the peer must have some means for retrieving or building the information that it sends. This might be done in several ways, e.g. by querying a local data base or by querying another service. The way in which this operation is performed is not relevant, the important point is to be sure that in principle the peer can execute it. For completing the construction of the policy, it is necessary to have a means for checking whether the peer can actually play the policy, in other words, if it has the *required capabilities*. This can only be done if we have a specification of which capabilities are required in the choreography itself. The capability verification can be accomplished role by role by the specific party willing to take part to the interaction.

This paper presents a work aimed to introduce the concept of capability in the global/local system/entity specifications, in such a way that capabilities can be accounted for during the processes that are applied for dynamically building and possibly customizing policies. Section 2 defines the setting of the work. Moreover, a first example of protocol (the well-known FIPA Contract Net protocol), that is enriched with capabilities, is reported. Section 3 introduces our notion of *capability test*, making a comparison with systems in which this notion is implicit. The use of reasoning techniques that can be associated with the capability test for performing a customization of the policy being constructed is also discussed. In Section 4 a possible extension of WS-CDL [29] with capability capability requirements is sketched. Conclusions follow.

2 Interaction Protocols and Capabilities

The concept of "interaction protocol" derives from the area of MASs. MASs often comprise heterogeneous agents, 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. In general, every agent in a MAS is characterized by a set of actions and/or a set of behaviors that it uses to achieve a specific goal. In order to interact with the others, an agent specification must describe also the communicative behavior.

When a peer needs to play a role in some interaction ruled by a protocol but it does not own a conformant policy, it is necessary that it *adopts* a new interaction policy. In an agent-framework, one might think of enriching the set of behaviors of the agent, which failed the conformance test, by asking other agents to supply a correct interaction policy. This solution has been proposed from time to time in the literature; recently it was adopted in Coo-BDI architectures [2]. CooBDI extends the BDI (*Belief, Desire, Intention*) model so that agents are enabled to cooperate through a mechanism of plan exchange. Such a mechanism is used whenever it is not possible to find a plan for pursuing a goal of interest by just exploiting the current agent's knowledge. The ideas behind the CooBDI theory have been implemented by means of WS technologies, leading to CooWS agents

[8]. Another recent work in this line of research is [26]: in the setting of the DALI language, agents can cooperate by exchanging sets of rule that either define a procedure, or constitute a module for coping with some situation, or are just a segment of a knowledge base. Moreover, agents have reasoning techniques that enable them to evaluate how useful the new information is. These techniques, however, cannot be directly imported in the context of Service-oriented Computing. The reason is that, while in agent systems it is not a problem to find out *during* the interaction that an agent does not own all the necessary actions, when we compose web services it is fundamental that the analogous knowledge is available before the interaction takes place.

A viable alternative is to use the protocol definition for supplying the service with a new policy that is obtained directly from the definition of the role, that the peer would like to play. A policy skeleton could be directly synthesized in a semi-automatic way from the protocol description. A similar approach has been adopted, in the past, for synthesizing agent behaviors from UML specifications in [18]. However, a problem arises: protocols only concern communication patterns, i.e. the interactions of a peer with others, abstracting from all references to the internal state of the player and from all actions/instructions that do not concern observable communication. Nevertheless, in our framework we are interested in a policy that the peer will *execute* and, for permitting the execution, it is necessary to express to some extent also this kind of information. The conclusion is that if we wish to use protocols for synthesizing policy skeletons, we need to specify some more information, i.e. actions that allow us the access to the peer's internal state. Throughout this work we will refer to such actions as *capability requirements*.

The term "capability" has recently been used by Padgham et al. [22] (the work is inspired by JACK [9] and it is extended in [23]), in the BDI framework, for identifying the "ability to react rationally towards achieving a particular goal". More specifically, an agent has the capability to achieve a goal if its plan library contains at least one plan for reaching the goal. The authors incorporate this notion in the BDI framework so as to constrain an agent's goals and intentions to be compatible with its capabilities. This notion of capability is orthogonal w.r.t. what is proposed in our work. In fact, we propose to associate to a choreography (or protocol) specification, aimed at representing an interaction schema among a set of yet unspecified peers, a set of *requirements* of capabilities. Such requirements specify "actions" that peers, willing to play specific roles in the interaction schema, should exhibit. In order for a peer to play a role, some verification must be performed for deciding if it matches the requirements.

In this perspective, our notion of capability resembles more closely (sometimes unnamed) concepts, that emerge in a more or less explicit way in various frameworks/languages, in which there is a need for defining interfaces. One example is Jade [15], the well-known platform for developing multi-agent systems. In this framework policies are supplied as partial implementations with "holes" that the programmer must fill with code when creating agents. Such holes are represented by methods whose body is not defined. The task of the programmer is to implement the specified methods, whose name and signature is, however,

fixed in the partial policy. Another example is powerJava [6,7], an extension of the Java language that accounts for roles and institutions. Without getting into the depths of the language, a role in powerJava represents an interlocutor in the interaction schema. A role definition contains only the implementation of the interaction schema and leaves to the role-player the task of implementing the internal actions. Such calls to the player's internal actions are named "requirements" and are represented as method prototypes.

Checking whether a peer has the capability corresponding to a requirement is, in a way, a complementary test w.r.t. checking conformance. With a rough approximation, when I check conformance I abstract away from the behavior that does not concern the communication described by the protocol of interest, focussing on the interaction with a set of other peers that are involved, whereas checking capabilities means to check whether it is possible to tie the description of a policy to the execution environment defined by the peer.

2.1 An Example: The Contract Net Protocol

For better explaining our ideas, in this section we consider as a choreography the well-known FIPA ContractNet Protocol [12], pinpointing the capabilities that are required to a peer which would like to play the role of *Participant*. ContractNet is used in electronic commerce and in robotics for allowing entities, which are unable to do some task, to have it done. The protocol captures a pattern of interaction, in which the initiator sends a *call-for-proposal* to a set of participants. Each participant can either accept (and send a proposal) or refuse. The initiator collects all the proposals and selects one of them. Figure 1 describes the interactions between the *Initiator* and one of the *Participants* in a UML notation, that is enriched with dotted rectangles representing *capability requirements*. The capability requirements act as connecting points between the external, communicative behavior of the candidate role player and its internal behavior. In the example, three different capabilities can be detected, one for the role of *Initiator* and two for the *Participant*. Starting from an instance of the concept Task, the *Participant* must be able to evaluate it by performing the *evaluateTask* capability, returning an instance of the concept Proposal. Moreover, if its proposal is accepted by the *Initiator*, it must be able to execute the task by using the capability *executeTask*, returning an instance of concept Result. On the other side, the *Initiator* must have the capability *evaluateProposal* that chooses a proposal among those received from the participants.

In order to play the role of *Participant* a peer will, then, need to have the capabilities *evaluateTask* and *executeTask*, whereas it needs to have the capability *evaluateProposal* if it means to play the role of *Initiator*. As it emerges from the example, a capability identifies an action (in a broad sense) that might require some inputs and might return a result. This is analogous to defining a method or a function or a web service. So, for us, a capability will be specified by its name, a description of its inputs and a description of its outputs. This is not the only possible representation, for instance if we interpret them as actions, it would make sense to represent also their preconditions and effects.

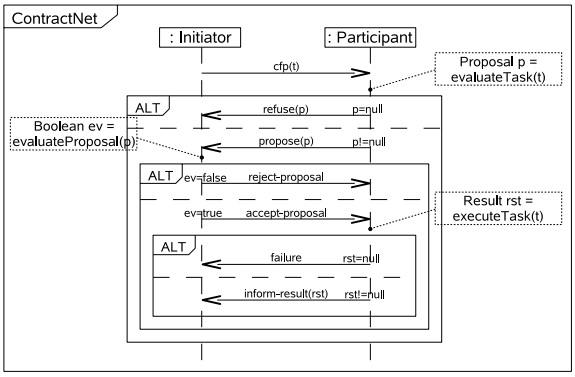


Fig. 1. The FIPA ContractNet Protocol, represented by means of UML sequence diagrams, and enriched with capability specifications

3 Checking Capabilities

In this section we discuss about possible implementations of the capability test, intended as the verification that a service satisfies the capability requirements given by a role. The capability test obviously depends on the way in which the policy is developed and therefore it depends on the adopted language. In Jade [15] there is no real capability test because policies already supply empty methods corresponding to the capabilities, the programmer can just redefine them. In powerJava the check is performed by the compiler, which verifies the implementation of a given interface representing the requirements. For further details see [6], in which the same example concerning the ContractNet protocol is described. In the scenario outlined in the previous section, the capability test is done *a priori w.r.t. all the capabilities required by the role specification* but the way in which the test is implemented is not predefined and can be executed by means of different matching techniques. We could use a simple *signature matching*, like in classical programming languages and in powerJava, as well more flexible forms of matching.

We consider particularly promising to adopt *semantic matchmaking* techniques proposed for matching web service descriptions with queries, based on *ontologies* of concepts. In fact semantic matchmaking supports the matching of capabilities with different names, though connected by an ontology, and with different numbers (and descriptions) of input/output parameters. For instance, let us consider the *evaluateProposal* capability associated to the role *Initiator* of the ContractNet protocol (see Figure 1). This capability has an input parameter (a proposal) and is supposed to return a boolean value, stating whether the proposal has been accepted or refused. A first example of flexible, semantics-based matchmaking consists in allowing a service to play the part of *Initiator* even though it does not have a capability of name *evaluateProposal*. Let us suppose that *evaluateProposal* is a concept in a shared ontology. Then, if the service has

a capability *evaluate*, with same signature of *evaluateProposal*, and *evaluate* is a concept in the shared ontology, that is more general than *evaluateProposal*, we might be eager to consider the capability as matching with the description associated to the role specification.

Semantic matchmaking has been thoroughly studied and formalized also in the Semantic Web community, in particular in the context of the DAML-S [24] and WSMO initiatives [16]. In [24] a form of semantic matchmaking concerning the input and output parameters is proposed. The ontological reasoning is applied to the parameters of a semantic web service, which are compared to a query. The limit of this technique is that it is not possible to perform the search on the basis of a goal to achieve. A different approach is taken in the WSMO initiative [16], where services are described based on their preconditions, assumptions, effects and postconditions. Preconditions concern the structure of the request, assumptions are properties that must hold in the current state, as well as effects will hold in the final state, while postconditions concern the structure of the answer. These four sets of elements are part of the "capability" construct used in WSMO for representing a web service. Moreover, each service has its own choreography and orchestration, although these terms are used in a different way w.r.t. our work. In fact, both refer to subjective views, the former recalls a state chart while the latter is a sequence of if-then rules specifying the interaction with other services. On the other hand, users can express goals as desired postconditions. Various matching techniques are formalized, which enable the search for a service that can satisfy a given goal; all of them presuppose that the goal and the service descriptions are ontology-based and that such ontologies, if different, can be aligned by an ontology mediator. Going back to our focus concerning capability matching, in the WSMO framework it would be possible to represent a "capability requirement", associated with a choreography, as a WSMO goal, to implement the "capabilities" of the specific services as WSMO capabilities, and then apply the existing matching techniques for deciding whether a requirement is satisfied by at least one of the capabilities of a service.

In order to ground our proposal to the reality of web services, in Section4, we will discuss a first possible extension of WS-CDL with capability requirements expressed as input and output parameters. For performing the capability test on this extension, it will be possible to exploit some technique for the semantic matchmaking based on input and output parameters, e.g. the one in [24].

3.1 Reasoning on Capabilities

In the previous sections we discussed the simple case when the capability test is performed w.r.t *all* the capabilities required by the role specification. In this case, based on some description of the required capabilities for a playing the role, we perform the matching among all required and actual service capabilities, thus we can say that the test allows to implement policies that perfectly fit the role, by envisioning all the execution paths foreseen by the role. This is, however, just a starting point. Further customization of the capability test w.r.t. some characteristic or goal of the service that intend to play a given role can be achieved by

combining the test with a reasoning phase on capabilities. For instance, by reasoning on capabilities from the point of view of the service candidate for playing the role, it would be possible to find out policies that implement the role but do not envision all the execution paths and thus do not require the entire list of capabilities associated to the role to be implemented.

Let us take the abstraction of a policy implementing a role w.r.t. all the capabilities required as a procedure with different *execution traces*. Each execution trace corresponds to a branch in the policy. It is likely that only a subset of the capabilities associated to a role will be used along a given branch. As an example, Figure 2 shows three alternative execution traces for a given policy, which contain references to different capabilities: one trace exploits capabilities *C1* and *C3*, the second one exploits *C1* and *C4*, the third one contains only *C2*.

We can think of a variant of the capability test in which only the execution traces concerning the specific call, that the service would like to enact, are considered. This set will tell us which capabilities are actually necessary in our execution context (i.e. given the specified input parameter values). In this perspective, it is not compulsory that the service has all the capabilities associated to the role but it will be sufficient that it has those used in this set of execution traces. Consider Figure 2 and suppose that for some given input values, only the first execution trace (starting from left) might become actually executable. This trace relies on capabilities *C1* and *C3* only: it will be sufficient that the service owns such capabilities for making the *policy call* executable.

Such kind of reasoning could be done by describing the ideal complete policy for a service aiming at implementing a given role in a declarative language that supports a-priori reasoning on the policy executions. In fact, if a *declarative representation* of the complete policy were given, e.g. see [4], it would be possible to perform a rational inspection of the policy, in which the execution is simulated. By reasoning we could select the execution traces that allow the service to complete the interaction for the inputs of the given call. Finally we could collect the capabilities used in these traces only (*C1*, *C3*, and *C4* but not *C2*) and restrict the capability test to that subset of capabilities.

Another possible customization task consists on reasoning about those execution traces that, after the execution, make a certain condition become true in

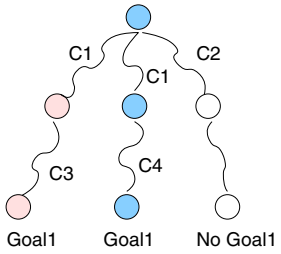


Fig. 2. Execution traces for a policy: two traces allow to reach a final state in which *goal1* is true but exploiting different capabilities

the service internal state. For instance, with reference to Figure 2, two out of the three possible executions lead to a final situation in which *goal1* holds. As a simple example of this case, let us suppose that a peer that wishes to play the role of "customer" with the general goal of purchasing an item of interest from a seller of interest, has a second goal, i.e. to avoid the use of credit cards. This goal can actually be seen as a constraint on the possible interactions. If the policy implementing the complete role allows three alternatives forms of payment (by credit card, by bank transfer and by check), the candidate customer is likely to desire to continue the interaction because some of the alternatives allow reaching the goal of purchasing the item of interest without using credit cards. It can, then, customize the policy by deleting the undesired path. If some of the capabilities are to be used *only* along the discarded execution path, it is not necessary for the candidate customer to have it.

Nevertheless a natural question arises: if I remove some of the possible execution paths of a policy, will it still be conformant to the specification? To answer to this question we can rely on our conformance test. In the specific case of the example, the answer would be positive. It would not be positive if we had a candidate seller that, besides having the general goal of selling items, has the second requirement of not allowing a specific form of payment (e.g. by bank transfer) and deletes the undesired path from the policy. Indeed, a customer that conforms to the shared choreography might require this form of payment, which is foreseen by the specification, but the candidate seller would not be able to handle this case leading to a deadlock.

It is also possible to generalize this approach and selecting the set of the execution traces that can possibly be engaged by a given service by using the information about the actual capabilities of the services. In fact, having the possibility of inspecting the possible evolutions of an ideal policy implementing the complete role, one could single out those execution traces that require the subset of capabilities that the service actually can execute. In this way, the policy can be customized w.r.t. the characteristic of the service, guaranteeing the success under determined circumstances.

Last but not least, the set of capabilities of a service could be not completely predefined but depending on the context and on privacy or security policies defined by the user: I might have a capability which I do not want to use in that circumstance. Also this kind of reasoning can be integrated in the capability test. In this perspective, it would be interesting to explore the use of the notion of *opportunity* proposed by Padmanabhan et al. [23] in connection with the concept of capability (but with the meaning proposed in [22], see Section 1).

4 A Case Study: Introducing Capability Requirements in WS-CDL

The most important formalism used to represent interaction protocols is WS-CDL (Web Services Choreography Description Language) [29]: an XML-based language that describes peer-to-peer collaborations of heterogeneous entities

from a global point of view. In this section, we propose a first proposal of extension of the WS-CDL definition where capability requirements are added in order to enable the automatic synthesis of policies described in the previous sections. Capability requirements are expressed as input and output parameters, then semantic matchmaking based on input and output parameters could be exploited as technique for performing the capability checking. The schema that defines this extension can be found at `http://www.di.unito.it/~ alice/WSCDL_Cap_v1/`.

```
1  <silentAction roleType="Participant">
2    <capability name="evaluateTask">
3      <input>
4        <parameter variable="cdl:getVariable('tns:t','','')"/>
5      </input>
6      <output>
7        <parameter variable="cdl:getVariable('tns:p','','')"/>
8      </output>
9    </capability>
10 </silentAction>
```

Fig. 3. Representing a capability in the extended WS-CDL. The tag *input* is used to define one of the input parameters, while *output* is used to define one of the output parameters.

In this scenario an operation executed by a peer often corresponds to an invocation of a web service, in a way that is analogous to a *procedure call*. Coherently, we can think of representing the concept of capability in the WS-CDL extension as a new tag element, the tag *capability* (see for instance Figure 3), which is characterized by its *name*, and its *input* and *output parameters*. Each parameter refers to a variable defined inside the choreography document. The notation `variable="cdl:getVariable('tns:t','','')"` used in Figure 3 is a reference to a variable, according to the definition of WS-CDL. In this manner inputs and outputs can be used in the whole WS-CDL document in standard ways (like Interaction, Workunit and Assign activities). In particular parameters can be used in guard conditions of Workunits inside a Choice activities in order to choose alternative paths (see below for an example). Notice that each variable refers also to a concept in a defined ontology.

A capability represents an operation (a call not a declaration) that must be performed by a role and which is non-observable by the other roles; this kind of activity is described in WS-CDL by *SilentAction* elements. The presence of silent actions is due to the fact that WS-CDL derives from the well-known *pi-calculus* by Milner *et al.* [19], in which silent actions represent the non-observable (or private) behavior of a process. We can, therefore, think of modifying the WS-CDL definition by adding capabilities as child elements of this kind of activity [1].

[1] Since in WS-CDL there is not the concept of observable action, capability requirements can describe only silent actions.

Returning to Figure 3, as an instance, it defines the capability *evaluateTask* for the role *Participant* of the Contract Net protocol. More precisely, *evaluateTask* is defined within a silent action and its definition comprises its name plus a list of inputs and outputs. The tags *capability*, *input*, and *output* are defined in our extension of WS-CDL. It is relevant to observe that each parameter refers to a variable that has been defined in the choreography.

```
1  <choice>
2    <workunit name="informResultWorkUnit"
3      guard="cdl:getVariable('tns:rst', '', '', 'tns:Participant') !=
                                                          'failure' ">
4      <interaction name="informResultInteraction">
5       ...
6      </interaction>
7    </workunit>
8    <interaction name="failureExecuteInteraction">
9      ...
10   </interaction>
11 </choice>
```

Fig. 4. Example of how output parameters can be used in a *choice* operator of a choreography

Choreographies not only list the set of capabilities that a service should have but they also identify the points of the interaction at which such capabilities are to be used. In particular, the values returned by a call to a capability (as a value of an output parameter) can be used for controlling the execution of the interaction. Figure 4 shows, for example, a piece of a choreography code for the role *Participant*, containing a *choice* operator. The *choice* operator allows two alternative executions: one leading to an inform speech act, the other leading to a failure speech act. The selection of which message will actually be sent is done on the basis of the outcome, previously associated to the variable *rst*, of the capability *executeTask*. Only when such variable has a non-null value the inform will be sent. The guard condition at line 3 in Figure 4 amounts to determine whether the task that the *Participant* has executed has failed.

To complete the example we sketch in Figure 5 a part of the ContractNet protocol as it is represented in our proposal of extension for WS-CDL. In this example we can detect three different capabilities, one for the role of *Initiator* and two for the role *Participant*. Starting from an instance of the type *Task*, the *Participant* must be able to evaluate it by performing the *evaluateTask* capability (lines 4-9), returning an instance of type *Proposal*. Moreover, it must be able to execute the received task (if its proposal is accepted by the *Initiator*) by using the capability *executeTask* (lines 26-31), returning an instance of type *Result*. On the other side, the *Initiator* must have the capability *evaluateProposal*, for choosing a proposal out of those sent by the participants (lines 15-20).

```
<sequence>
  <interaction name="callForProposalInteraction"> ...
  </interaction>
  <silentAction roleType="Participant">
    <capability name="evaluateTask">
      <input> ...  </input>
      <output> ...  </output>
    </capability>
  </silentAction>
  <choice>
    <workunit name="proposeWorkUnit" guard=...  >
      <sequence>
        <interaction name="proposeInteraction">
        </interaction>
        <silentAction roleType="Initiator">
          <capability name="evaluateProposal">
            <input> ...  </input>
            <output> ...  </output>
          </capability>
        </silentAction>
        <choice>
          <workunit name="acceptProposalWorkUnit" guard=...  >
            <sequence>
              <interaction name="proposeInteraction">
              </interaction>
              <silentAction roleType="Participant">
                   <capability name="executeTask">
                     <input> ...  </input>
                     <output> ...  </output>
                   </capability>
              </silentAction>
              <choice>
                <workunit name="informResultWorkUnit"
                  guard=...  >
                  <interaction name="informResultInteraction">
                  </interaction>
                </workunit>
                <interaction name="failureExecuteInteraction">
                </interaction>
              </choice>
            </sequence>
          </workunit>
          <interaction name="rejectProposalInteraction">
          </interaction>
        </choice>
       </sequence>
    </workunit>
    <interaction name="evaluateTaskRefuseInteraction">
    </interaction>
  </choice>
</sequence>
```

Fig. 5. A representation of the FIPA ContractNet Protocol in the extended WS-CDL

As we have seen in the previous sections, it is possible to start from a representation of this kind for performing the capability test and check if a service can play a given role (e.g. *Initiator*). Moreover, given a similar description it is also possible to synthesize the skeleton of a policy, possibly customized w.r.t. the capabilities and the goals of the service that is going to play the role. To this aim, it is necessary to have a translation algorithm for turning the XML-based specification into an equivalent schema expressed in the execution language of interest.

5 Conclusions

This work presents a preliminary study aimed to allow the use of public choreography specifications for automatically synthesizing executable interaction policies for peers that would like to take part in an interaction but that do not own an appropriate policy themselves. To this purpose it is necessary to link the abstract, communicative behavior, expressed at the protocol level, with the internal state of the role player by means of actions that might be non-communicative in nature (capabilities). It is important, in an open framework like the web, to be able to take a decision about the possibility of taking part to a choreography before the interaction begins. This is the reason why we have proposed the introduction of the notion of capability at the level of choreography specification. A capability is the specification of an action in terms of its name, and of its input and output parameters. Given such a description it is possible to apply matching techniques in order to decide whether a service has the capabilities required for playing a role of interest. In particular, we have discussed the use of semantic matchmaking techniques, such as those developed in the WSMO and DAML-S initiatives [24], for matching web service descriptions to queries.

We have shown how, given a (possibly) declarative representation of the policy skeletons, obtained from the automatic synthesis process, it is possible to apply further reasoning techniques for customizing the implemented policy to the specific characteristic of the service that will act as a player. Reasoning techniques for accomplishing this customization task are under investigation. In particular, the techniques that we have already used in previous work concerning the personalization of the interaction with a web service [4] seem promising. In that work, in fact, we exploited a kind of reasoning known as *procedural planning*, relying on a logic framework. Procedural planning explores the space of the possible execution traces of a procedure, extracting those paths at whose end a goal condition of interest holds. It is noticeable that in presence of a sensing action, i.e. an action that queries for external input, all of the possible answers are to be kept (they must all lead to the goal) and none can be cut off. In other words, it is possible to cut only paths that correspond to some action that are under the responsibility of the agent playing the policy. The waiting for an incoming message is exactly a query for an external input, as such the case of the candidate seller that does not allow a legal form of payment cannot occur.

Our work is close in spirit to [25], where the idea of keeping separate procedural and ontological descriptions of services and to link them through semantic annotations is introduced. In fact our WS-CDL extension can be seen as procedural description of the interaction enriched with capabilities requirements, while semantic annotations of capability requirements enable the use of ontological reasoning for the capability test phase. Presently, we are working at more thorough formalization of the proposal that will be followed by the implementation of a system that turns a role represented in the proposed extension of WS-CDL into an executable composite service, for instance represented in WS-BPEL. WS-BPEL is just a possibility, actually any programming language by means of which it is possible to develop web services could be used.

References

1. M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Specification and verification of agent interactions using social integrity constraints. In *Proc. of the Workshop on Logic and Communication in Multi-Agent Systems, LCMAS 2003*, volume 85(2) of *ENTCS*, 2003. Elsevier.
2. D. Ancona and V. Mascardi. Coo-BDI: Extending the BDI Model with Cooperativity. In *Proceedings of the 1st Declarative Agent Languages and Technologies Workshop (DALT'03)*, pages 109–134. Springer-Verlag, 2004. LNAI 2990.
3. 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.
4. M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about interaction protocols for customizing web service selection and composition. *J. of Logic and Algebraic Programming, special issue on WS and Formal Methods*, 2006. To appear.
5. 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 2nd Int. Workshop on Web Services and Formal Methods, WS-FM 2005*, volume 3670 of *LNCS*, pages 257–271. Springer, September, 2005.
6. M. Baldoni, G. Boella, and L. van der Torre. Bridging Agent Theory and Object Orientation: Importing Social Roles in Object Oriented Languages. In *Post-Proc. of the Int. Workshop on Programming Multi-Agent Systems, ProMAS 2005*, volume 3862 of *LNCS*, pages 57–75. Springer, 2006.
7. M. Baldoni, G. Boella, and L. van der Torre. powerjava: Ontologically Founded Roles in Object Oriented Programming Languages. In *Proc. of 21st SAC 2006, Special Track on Object-Oriented Programming Languages and Systems*, 2006. ACM.
8. L. Bozzo, V. Mascardi, D. Ancona, and P. Busetta. CooWS: Adaptive BDI agents meet service-oriented computing. In *Proc. of the Int. Conference on WWW/Internet*, pages 205–209, 2005.
9. P. Busetta, N. Howden, R. Ronquist, and A. Hodgson. Structuring BDI agents in functional clusters. In *Proc. of the 6th Int. Workshop on Agent Theories, Architectures, and Languages (ATAL99)*, 1999.
10. 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.
11. F. Dignum, editor. *Advances in agent communication languages*, volume 2922 of *LNAI*. Springer-Verlag, 2004.

12. Foundation for Intelligent Physical Agents. http://www.fipa.org.
13. F. Guerin and J. Pitt. Verification and Compliance Testing. In *Communication in Multiagent Systems*, volume 2650 of *LNAI*, pages 98–112. Springer, 2003.
14. M. P. Huget and J.L. Koning. Interaction Protocol Engineering. In *Communication in Multiagent Systems*, volume 2650 of *LNAI*, pages 179–193. Springer, 2003.
15. Jade. http://jade.cselt.it/.
16. U. Keller, R. Laraand A. Polleres, I. Toma, M. Kifer, and D. Fensel. D5.1 v0.1 wsmo web service discovery. Technical report, WSML deliverable, 2004.
17. 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.
18. M. Martelli and V. Mascardi. From UML diagrams to Jess rules: Integrating OO and rule-based languages to specify, implement and execute agents. In *Proc. of the 8th APPIA-GULP-PRODE Joint Conf. on Declarative Programming (AGP'03)*, pages 275–286, 2003.
19. R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.
20. OASIS. Business process execution language for web services.
21. J. H. Odell, H. Van Dyke Parunak, and B. Bauer. Representing agent interaction protocols in UML. In *Agent-Oriented Software Engineering*, pages 121–140. Springer, 2001. http://www.fipa.org/docs/input/f-in-00077/.
22. L. Padgham and P. Lambrix. Agent capabilities: Extending BDI theory. In *AAAI/IAAI*, pages 68–73, 2000.
23. V. Padmanabhan, G. Governatori, and A. Sattar. Actions made explicit in BDI. In *Advances in AI*, number 2256 in LNCS, pages 390–401. Springer, 2001.
24. M. Paolucci, T. Kawmura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *First International Semantic Web Conference*, 2002.
25. M. Pistore, L. Spalazzi, and P. Traverso. A minimalist approach to semantic annotations for web processes compositions. In *ESWC*, pages 620–634, 2006.
26. Arianna Tocchio and S. Costantini. Learning by knowledge exchange in logical agents. In *Proc. of WOA 2005: Dagli oggetti agli agenti, simulazione e analisi formale di sistemi complessi*, november 2005. Pitagora Editrice Bologna.
27. W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, H. M. W. Verbeek, and P. Wohed. Life after BPEL? In *Proc. of WS-FM'05*, volume 3670 of *LNCS*, pages 35–50. Springer, 2005. Invited speaker.
28. Michael Wooldridge and Simon Parsons. Issues in the design of negotiation protocols for logic-based agent communication languages. In *Agent-Mediated Electronic Commerce III, Current Issues in Agent-Based Electronic Commerce Systems*, volume 2003 of *LNCS*. Springer, 2001.
29. WS-CDL. http://www.w3.org/tr/ws-cdl-10/.

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,

* 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.

F. Toni and P. Torroni (Eds.): CLIMA VI, LNAI 3900, pp. 265–283, 2006.

or cannot, take part in the system. In general, based on this abstraction, open 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 verifying 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 [24, 25]. Following [23], 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 [10, 9, 29], and two kinds of conformance verifications have been studied: *a priori* conformance verification, and *run-time* conformance verification (or compliance) [14, 15, 21]. 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 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 [7], 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 [13, 16]. By this translation it is, in fact, possible to prove

properties that the original representation does not allow. In this context, in [5] we have shown an easy algorithm for translating AUML sequence diagrams to finite state automata thus enabling the verification of conformance.

In [5] 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.

More specifically, 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. Notice that 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 [17, 28, 20]. However, the speech acts semantics does not play a role in our proposal, which concerns an orthogonal aspect of the interaction in Multi Agent Systems.

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 $\mathsf{m}(\overleftarrow{a\,g})$, and the latter, where ag is the sender, by $\mathsf{m}(\overrightarrow{a\,g})$. 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 shorthands, 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 $\mathsf{m}(\overleftarrow{a\,g_r})$ from the point of view of ag_r, and $\mathsf{m}(\overrightarrow{a\,g_s})$ 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.

Definition 1 (Conversation). *A conversation is a sequence σ of messages taken from a given set $\mathcal{SA}$ of speech acts.*

In this work we face the problem of conformance verification and interpret a priori conformance as a property that relates two *formal languages* [22], 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. Each of these languages represents a set of conversations. In the case of the protocol specification, it is intuitive that it will be the set of all the possible conversations allowed by the protocol among the partners. In the case of the single agent's policy, it will be the set of the possible conversations that the agent can carry on according to the policy. Of course, at execution time, depending on the interlocutor and on the circumstances, only one conversation at a time is actually expressed, however, for verifying conformance *a priori* we need to consider them all as a set.

Definition 2 (Policy language). *Given a policy p^{ag}_{lang}, where p is the policy name, $lang$ is the language in which this is implemented, and ag is the agent that uses it, we denote by $L(p^{ag}_{lang})$ the set of conversations that ag can carry on according to p.*

Definition 3 (Protocol language). *Given a conversation protocol p_{spec}, where p is the protocol name, and $spec$ is the language in which it is represented, we denote by $L(p_{spec})$ the set of conversations that a set of agents, that interact according to p, can carry on.*

The assumption that we make throughout this paper is that the two languages $L(p^{ag}_{lang})$ and $L(p_{spec})$ are *regular*. 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*.

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 4 (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.

Definition 5 (Interoperability). Interoperability *is the capability of a set of agents of actually producing a conversation when interacting with one 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 a set of rules -the protocol- that characterize such execution

context; as long as the new agent satisfies the rules, the interoperability with the other components of the system is guaranteed. Thus in protocol-based systems the interoperability of an agent with others can be proved by checking the communicative behavior of the agent against the rules of the system, 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.

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

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 Definition 4 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.

> **Expectation 1.** *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.

> **Expectation 2.** *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*.

> **Expectation 3.** *The third observation is that we expect that a policy always allows an agent to utter one of the messages foreseen by the protocol at every point of the possible conversations. This means that it is not necessary that a policy envisions all the possible alternative utterances, but it is required to foresee at least one of them that allows the agent to proceed with its conversations.*

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. 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 guarantees 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 conversations 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 will it 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*, denoted by $M(p^{ag}_{lang})$ and $M(p_{spec})$ respectively, that we can assume as having the *same alphabet* (see [22]). 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 labeled either by incoming messages or by outgoing messages w.r.t. *ag*.

Definition 7 (IO-automaton w.r.t. *ag*). *Given an automaton $M = (Q, \Sigma, \delta, q_0, F)$, let $E_q = \{m \mid \delta(p, m) = q\}$ for $p, q \in Q$. We say that M is an IO-automaton w.r.t. ag 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* (Definition 4). For the so built automaton, it is easy to prove the following property.

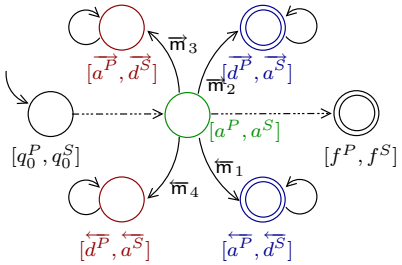


Fig. 1. A general schema of the M_{conf} automaton. From bottom-right, clockwise, cases (a), (b), (c), and (d).

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

The definition of IO-automata is used in the following for the construction of the automaton M_{conf}.

Definition 8 (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 $\overrightarrow{q} = [\overrightarrow{d^P}, \overrightarrow{a^S}]$ in Q is an O-state, such that $\overrightarrow{d^P}$ is dead and $\overrightarrow{a^S}$ is alive, we set $\delta(\overrightarrow{q}, m) = \overrightarrow{q}$ for every m in Σ, and we put $\overrightarrow{q}$ in F.

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.

Rule (d) is the symmetric case of (c) (Fig. 1, message $\overrightarrow{\mathsf{m}}_2$), it 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, to guarantee Expectation 3, we add the following requirement. The intuitive reason is that we would like an agent, which is supposed to utter a message at a certain point of its conversation, to actually do it, thus making the conversation, in which it is engaged, proceed.

Definition 9 (Complete automaton). *Let us denote by* $Messout(q)$ *the set:*

$$Messout(q) = \{\mathsf{m}(\overrightarrow{a}\overrightarrow{g}) \mid \delta(q, \mathsf{m}(\overrightarrow{a}\overrightarrow{g})) = p \wedge p \text{ is alive } \}$$

We say that the automaton M_{conf} *is* complete *iff for all states of form* $[q^P, q^S]$ *of* M_{conf}*, such that* $Messout(q^S) \neq \emptyset$*, the following holds:*

- $Messout(q^P) \neq \emptyset$*;*
- *if we substitute* $Messout(q^P)$ *to* $Messout(q^S)$ *in* M_{spec}*, the state* q^S *remains* alive*.*

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 10 (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 the union of the 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 Definition 10, 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 $L(p_{spec})$, when $L(p^{ag}_{lang}) \neq \emptyset$ and $L(p_{spec}) \neq \emptyset$.*

Proof. Since p^{ag}_{lang} is conformant, $L(p^{ag}_{lang})$ is accepted by M_{conf}. Then, by construction M_{conf} does not contain any state $[\overrightarrow{a^P}, \overrightarrow{d^S}]$, where a^P corresponds to an alive state in $M(p^{ag}_{lang})$ and d^S is a dead state in $M(p_{spec})$, due to illegal messages uttered by the agent. By construction it also does not contain any state $[\overleftarrow{d^P}, \overleftarrow{a^S}]$ due to incoming messages that are not accounted for by the policy. Obviously, no conversation σ accepted by states of the kind $[\overrightarrow{d^P}, \overrightarrow{a^S}]$ can belong to $L(p^{ag}_{lang})$ because the agent cannot utter the messages required to reach such states. Finally, no conversation produced by the agent will be accepted by states of the kind $[\overleftarrow{a^P}, \overleftarrow{d^S}]$ because by definition the protocol cannot utter illegal messages. Now, σ' is a common prefix, therefore it leads to a state of the automaton M_{conf} of the kind $[a^P, a^S]$ (i.e., both states are alive, see Figure 1). Due to *policy conformance*, all the incoming messages (w.r.t. the agent), that are labels of kind $\mathsf{m}(\overleftarrow{a}\overleftarrow{g})$ of outgoing edges, must be foreseen by the policy and in the case of outgoing messages (that is labels of kind $\mathsf{m}(\overrightarrow{a}\overrightarrow{g})$ of outgoing edges), the policy must foresee at least one of them in such a way that a^S is kept alive (*completeness* of M_{conf}). Therefore, either the above state $[a^P, a^S]$ is already a final state of M_{conf} and $\sigma'' = \varepsilon$ or from $[a^P, a^S]$ it is possible to perform one more common step, leading to a state of the same kind, i.e. composed of two alive states for the reasons exposed before. This an actual step ahead towards a final state due to conformance. In fact, for these properties there must be an edge outgoing from a^S, that leads to another alive state different from a^S, and the same edge must exist also in $M(p^{ag}_{lang})$; this edge will be one of the outgoing edges of a^P. We can choose to follow this edge also in the automaton M_{conf}. We can iteratively repeat this reasoning and, since the number of nodes is finite, we will eventually reach an accepting state, identifying a common conversation. **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 Definition 10, and denoting by I the intersection

$$I = \bigcap_{ag_i}^{i=1,2} L(p^{ag_i}_{lang_i})$$

we can prove ag_1 and ag_2 *interoperability*, that is they will produce a legal conversation, when interacting with one another. The proof 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 proof, it is possible to prove that a common legal conversation must exist when both policies satisfy the conformance test given by Definition 10.

Theorem 1 (Interoperability w.r.t. an interaction protocol). *Given two policies $p^{ag_1}_{lang_1}$ and $p^{ag_2}_{lang_2}$ that are conformant to a protocol specification p_{spec}, according to Definition 10, 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 $\sigma \in I$.*

Proof. First of all, it is trivial that σ' is also a prefix of $L(p_{spec})$. By the previous property, we are sure that both ag_1 and ag_2 contain some legal conversations. We need to prove that at least of these is common. Let us consider the automaton that accepts the intersection of $M(p^{ag_1}_{lang})$ and $M(p^{ag_2}_{lang})$. Since σ' is a common prefix, there must be a path in such automaton, that leads to a state $[q^{ag_1}, q^{ag_2}]$. Due to *policy conformance*, all the incoming messages w.r.t. ag_1, foreseen by the protocol specification, must be foreseen also by the policy. On the other side, ag_2 must utter at least one of them, due to its conformance (its M_{conf} must be complete). Therefore, it is possible to continue the conversation at least one more common step. In the case of messages that are outgoing w.r.t. ag_1 the policy must foresee at least one of them in such a way that q^{ag_1} is kept alive (completeness of M_{conf}), while on the other side, ag_2 must be able to handle all the possible alternatives (conformance), therefore, also in this case it is possible to continue the conversation. In both cases all the performed steps are legal w.r.t. the protocol specification. Therefore, either the above state $[q^{ag_1}, q^{ag_2}]$ is a final state and $\sigma'' = \varepsilon$ or from $[q^{ag_1}, q^{ag_2}]$ it is possible to perform one more common step for the reasons exposed before. Proceeding in a way that is analogous to what done in the proof of Prop. 2, due to conformance and considering each agent as playing the role of the protocol specification w.r.t. to the other, this an actual step ahead towards a final state. Therefore, we will eventually reach an accepting state, that identifies a common conversation. **q.e.d.**

The *third expectation* is guarantedd by the completeness of M_{conf}. The role plaied by completeness is, therefore, to guarantee that two agents, playing the two roles of the same protocol, will be able to lead to an end their conversations. Without this property we could only say that whenever the two agents will be able to produce a conversation, this will be legal. We lose the certainty of the capability of producing a conversation.

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. A naive procedure for performing the test can be obtained directly from Definitions 8 and 9 and from the well-known automata theory [22]. The following theorem holds.

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

4 The DyLOG Language: A Case Study

In this section we show how the presented approach particularly fits logic languages, using as a case study the DyLOG language [7], 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) [5]. This is, however, just an example. The same approach could be applied to other logic languages. In the following we will briefly recall how interaction policies can be described in the language DyLOG. For examples and for a thorough description of the core of the language see [7, 4].

DyLOG [7] 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, 4]. 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:

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

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:

$$\textsf{get\_message}(ag_i, ag_j, l) \textbf{ is}$$
$$\quad \textsf{speech\_act}_1(ag_j, ag_i, l) \textbf{ or } \ldots \textbf{ or } \textsf{speech\_act}_k(ag_j, ag_i, l)$$

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 determining if there is a possible execution of the enumerated actions after which the condition *Fs* holds. If the answer is positive, a

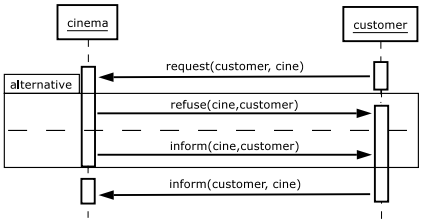


Fig. 2. AUML sequence diagram

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. [19]) 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 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 [26]. 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$(customer, cine, ack)$

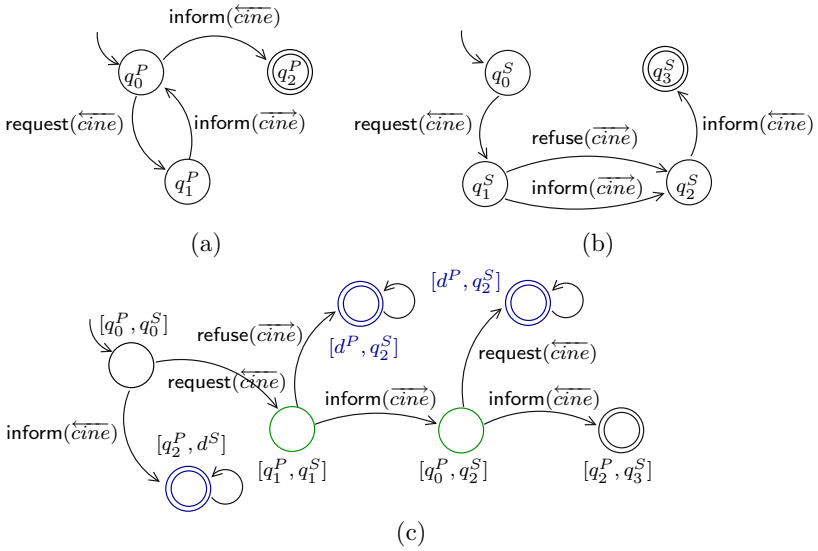


Fig. 3. (a) Policy of agent *cine*; (b) protocol specification; (c) M_{conf} automaton. Only the part relevant to the discussion is shown.

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 [5] 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 [5] 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.

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(*cus-tomer*, *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 cinema. 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 policy by $\mathsf{get\_info\_movie}^{cine}_{DyLOG}$. In general, it allows all the conversations that begin with a (possibly empty) series of exchanges of kind $\mathsf{request}(\overleftarrow{cine})$ followed by $\mathsf{inform}(\overrightarrow{cine})$, concluded by a message of kind $\mathsf{inform}(\overleftarrow{cine})$.

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 analyze M_{conf} for answering our queries. Trivially, the automaton is complete and it accepts both languages (of the policy, $L(\mathsf{get\_info\_movie}^{cine}_{DyLOG})$, and of the specification, $L(\mathsf{get\_info\_movie}_{AUML})$), 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 [15]. 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 colleagues 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* (Definition 10) 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* [15], 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.

A crucial difference concerns interoperability. In our framework, given two policies *each* of which is *conformant* to a protocol specification, their *interoperability* can be proved. Thus, 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, while Endriss et al. do not discuss this issue and do not formally prove that interoperability is entailed by (all or some of) their three definitions of conformance. 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).

This work is, actually, a deep revision of the work that the authors presented at [5], where the verification of a priori conformance was faced only in the specific case in which DyLOG [7] is used as the policy implementation language and AUML [26] 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 [5], 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 and not sufficient to express all the desiderata connected to this term, which are, instead, well-captured by our definitions of policy conformance.

The proposal that we have described in this paper is, actually, just a first step of a broader research. As a first step, we needed to identify the core of the problem, those key concepts and requirements which were necessary to capture and express the intuition behind a priori conformance, in the perspective of guaranteeing interoperability. Hence, the focus on interactions that involve two partners and do not account for concurrent operations. Under such restrictions, the choice of finite state automata fits very well and has the advantage of bearing along decidability.

Finite state automata, despite some notational inadequacy [20], are commonly used for representing protocols: for instance they have been used for representing both KQML protocols [8] and FIPA protocols [17]. In [5] we have presented an

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

algorithm for translating AUML protocol specifications in finite state automata, focussing -on the side of sequence diagrams- on the operators used to specify FIPA protocols, which are: message, alternative, loop, exit, and reference to a sub-protocol. Some concrete example of application to the specification of complex protocols are the English Auction [27] and the Contract Net Protocol [18]. As a future work we mean to study an extension to policies (and protocols) that involve many partners as well as an extension to policies (and protocols) that use concurrent operators. For the latter problem in the literature there are well studied formalisms such as process algebras that can be used for representing protocols involving concurrency elements. It could be interesting to study how to import on the new basis the lessons learnt in the current research.

Concerning works that address the problem of verifying the conformance in systems of communicating agents by using model checking techniques (e.g. [19]), to the best of our knowledge, the issue of interoperability is not tackled or, at least, this does not clearly emerge. For instance, Giordano, Martelli and Schwind [19] 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 [21] 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.

Techniques for proving if the local agent's policy *conforms* to the abstract protocol specification can have an interesting and natural application in the web service field. In fact a need of distinguishing a global and a local view of the interaction is recently emerging in the area of Service Oriented Architectures. In this case there is a distinction between the *choreography* of a set of peers, i.e. a global specification of the way a group of peers interact, and the concept of *behavioral interface*, seen as the specification of the interaction from the point of view of an individual peer. The recent W3C proposal of the choreography language WS-CDL [30] is emblematic. In fact the idea behind it is to introduce specific *choreography languages* as languages for a high-level specification, captured from a global perspective, distinguishing this representation from the other two, that will be based upon ad hoc languages (like BPEL or ebXML).

Taking this perspective, choreographies and agent interaction protocols undoubtedly share a common purpose. In fact, they both aim at expressing *global interaction protocols*, i.e. rules that define the global behavior of a system of cooperating parties. The respect of these rules guarantees the interoperability of the parties (i.e. the capability of *actually* producing an interaction), and that the interactions will satisfy given requirements. One problem that becomes crucial is the development of formal methods for verifying if the behavior of a peer respects a choreography [11, 12]. On this line, in [6] we moved the first steps toward the application of the conformance test proposed in the present paper for verifying *at design time* (a priori) that the internal processes of a web service enable it to participate appropriately in the interaction.

Acknowledgement. The authors would like to thank the anonimous reviewers for their helpful suggestions and Francesca Toni for the discussion that we had in London.

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, and V. Patti. Reasoning about interaction protocols for customizing web service selection and composition. *Journal of Logic and Algebraic Programming, Special issue on Web Services and Formal Methods*, 2006. to appear.
5. 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*, number 3487 in LNCS, pages 192–212. Springer, 2005.
6. 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 2nd Int. Workshop on Web Services and Formal Methods, WS-FM 2005*, number 3670 in LNCS, pages 257–271, 2005.
7. 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.
8. M. Barbuceanu and M. Fox. Cool: A language for describing coordination in multiagent systems. In *Proceedings International Conference on Multi Agent Systems (ICMAS'95)*, pages 17–24. MIT Press, Massachusetts, USA, 1995.
9. 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*, number 3487 in LNCS, pages 178–195. Springer, 2004.
10. R. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model Checking AgentSpeak. In *Proc. of 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2003*, 2003.
11. M. Bravetti, L. Kloul, and G. Zavattaro, editors. *Proc. of the 2nd International Workshop on Web Services and Formal Methods (WS-FM 2005)*, number 3670 in LNCS. Springer, 2005.
12. N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and Orchestration: a synergic approach for system design. In *Proc. the 3rd Int. Conf. on Service Oriented Computing*, 2005.
13. L. Cabac and D. Moldt. Formal semantics for auml agent interaction protocol diagrams. In *Proc. of AOSE 2004*, pages 47–61, 2004.
14. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Protocol conformance for logic-based agents. In G. Gottlob and T. Walsh, editors, *Proc. of IJCAI-2003*, pages 679–684. Morgan Kaufmann Publishers, August 2003.

15. 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.
16. R. Eshuis and R. Wieringa. Tool support for verifying UML activity diagrams. *IEEE Trans. on Software Eng.*, 7(30), 2004.
17. FIPA. Fipa 97, specification part 2: Agent communication language. Technical report, FIPA (Foundation for Intelligent Physical Agents), November 1997.
18. L. Giordano, A. Martelli, and C. Schwind. Specifying and verifying interaction protocols in a temporal action logic. *Journal of Applied Logic (Special issue on Logic Based Agent Verification)*. Accepted for publication.
19. 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.
20. F. Guerin. *Specifying Agent Communication Languages*. PhD thesis, Imperial College, London, April 2002.
21. 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.
22. J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley Publishing Company, 1979.
23. M. P. Huget and J.L. Koning. Interaction Protocol Engineering. In H.P. Huget, editor, *Communication in Multiagent Systems*, volume 2650 of *LNAI*, pages 179–193. Springer, 2003.
24. 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.
25. N. Maudet and B. Chaib-draa. Commitment-based and dialogue-based protocols: new trends in agent communication languages. *Knowledge engineering review*, 17(2), 2002.
26. 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.
27. J. Pitt, F. Guerin, and C. Stergiou. Protocols and intentional specifications of multi-party agent conversations for brokerage and auctions. In *Autonomous Agents 2000*, pages 269–276, Barcelona, 2000. ACM Prtess.
28. M. P. Singh. A social semantics for agent communication languages. In *Proc. of IJCAI-98 Workshop on Agent Communication Languages*, Berlin, 2000. Springer.
29. C. Walton. Model checking agent dialogues. In J. Leite, A. Omicini, P. Torroni, and P. Yolum, editors, *Declarative agent languages and technologies II, DALT 2004*, number 3476 in LNCS, pages 132–147. Springer, 2005.
30. WS-CDL. http://www.w3.org/tr/2004/wd-ws-cdl-10-20041217/. 2004.

Conformance and Interoperability in Open Enviroments

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)
Email: {baldoni,baroglio,mrt,patti}@di.unito.it

***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.**

I. 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 the one 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* [10]. 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 (although it can refuse the answer). 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 [14]. 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], [12], [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 III-A). 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 III). 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.

II. 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 m? (*incoming message*) and 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 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 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 [12].

Definition 2.1 (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 2.2 (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 2.3 (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 2.4 (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 2.5 (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.

III. 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.

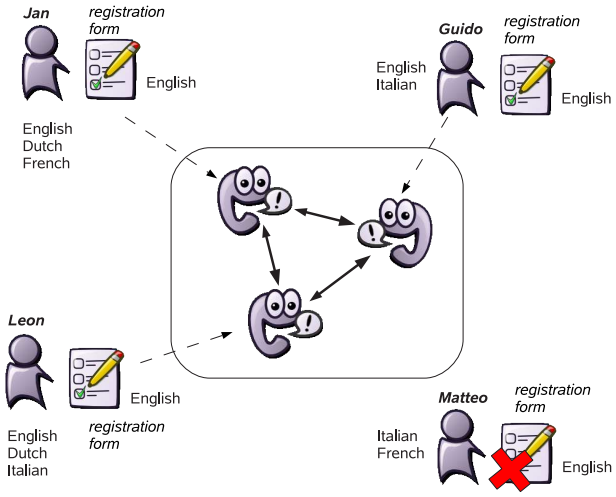


Fig. 1. Example of the summer school.

Figure 1 shows an intuitive example, in which a group of persons wish to attend a summer school. Each of them can speak and understand different languages. For instance, Jan can speak English, Dutch, and French. The school registration form requires the interested attendee to speak and understand English, which is the official language of the school. This requirement allows a person to decide if it will be in condition to interact with the other participants before attending the school. So, for instance, Matteo, who speaks Italian and could therefore interact with Guido, will not be in condition to understand the other participants. Jan and Leon could interact by speaking Ducth, however, since they also know English, they will be able to interact with all the other attendees and so they will be in condition to participate. The fact that they understand other languages besides the one required by the "school protocol" does not compromise their interoperability with the others. In fact, within the context of the school everybody will speak English with them. Interoperability is compromised when one does not understand (part of) the protocol (e.g. Matteo) or when one decides to speak a language that is not agreed (e.g. Leon when speaking Dutch).

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 3.1 (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.

A. 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 2, 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,

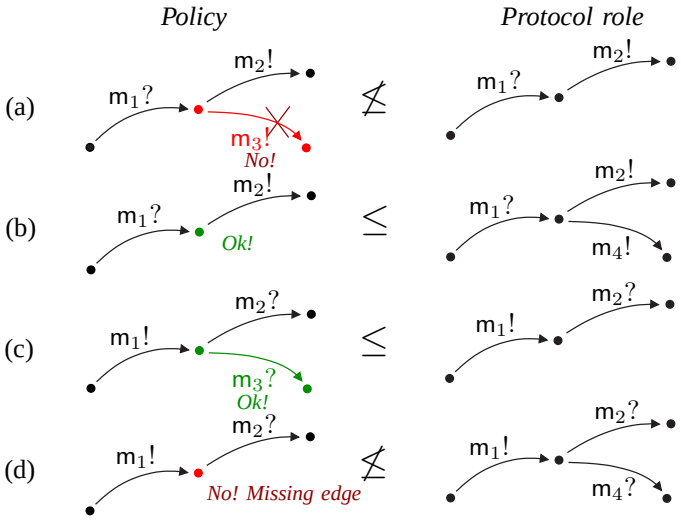


Fig. 2. A set of cases that exemplifies our expectations about a conformant policy: cases (b) and (c) do not compromise interoperability, hence they should pass the conformance test; cases (a) and (d) instead should not pass the conformance test.

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 2, 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 2, 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 2, 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.

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

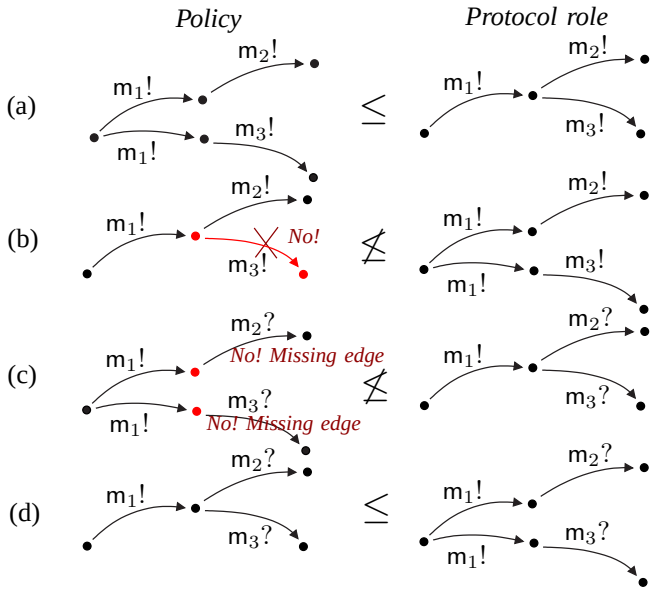


Fig. 3. A set of cases that exemplifies our expectations about a conformant policy: differently than in Figure 2, for every row, the policy and the role produce the same conversations but the structure of their implementations differ.

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 [13]. Figure 3 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 2 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 2.

The case of row (b), Figure 3, 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 2). Finally, the case of row (d) does not compromise the conformance, following what reported in Figure 2, row (c).

B. 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 2 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 3 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 3.3 and 3.4). This is what we expect from a conformant policy and from our definition of interoperability. Let A an FSA, let us denote by $\mathrm{Succ}(l, s)$ the set of states $\{s' \mid (s, l, s') \in A.T\}$.

Definition 3.2 (Conformant simulation): Given two FSA's A_1 and A_2, 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

1) $A_1.s_0 \mathcal{R} A_2.s_0$;
2) if $s_i \mathcal{R} s_j$, where $s_i \in A_1.S$ and $s_j \in A_2.S$, then
   a) for every $(s_i, \mathsf{m}!, s_{i+1}) \in A_1.T$, $\mathrm{Succ}(\mathsf{m}!, s_j) \neq \emptyset$ and $s_{i+1} \mathcal{R} s'$ for every $s' \in \mathrm{Succ}(\mathsf{m}!, s_j)$;
   b) for every $(s_j, \mathsf{m}?, s_{j+1}) \in A_2.T$, $\mathrm{Succ}(\mathsf{m}?, s_i) \neq \emptyset$ and $s_{j+1} \mathcal{R} s'$ for every $s' \in \mathrm{Succ}(\mathsf{m}?, s_i)$;

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 3.2 does not

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

imply the fact that "$A_1 \leq A_2$ entails $A_1 \subseteq A_2$". Instead, the following proposition holds.

Proposition 3.3: 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 3.4: 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 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 3.5 (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 3.6 (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 3.4, they will be able to conclude their interaction producing a legal accepted run. Therefore, Theorem 3.6 implies Definition 3.1 (interoperability).

IV. 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.* [11], 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.* [12], 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 2, it excludes the cases (b) and (c), and it also excludes cases (a) and (d) from Figure 3, which do not compromise interoperability. A recent proposal, in this same line, is [18], which suffers of the same limitations.

1) 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*. ACM, 2004, pp. 72–78.

[3] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services*. Springer, 2004.

[4] A. 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*, ser. LNCS State-of-the-Art Survey, vol. 3900. Springer, 2006, pp. 265–283.
[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*, ser. LNCS, no. 3487. Springer, 2005, pp. 192–212.
[8] ——, "Verifying the conformance of web services to global interaction protocols: a first step," in *Proc. of WS-FM 2005*, ser. LNCS. Springer, September, 2005, vol. 3670, pp. 257–271.
[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] B. Eckel, *Thinking in Java*. Prentice Hall, 2005.
[11] U. Endriss, N. Maudet, F. Sadri, and F. Toni, "Logic-based agent communication protocols," in *Advances in agent communication languages*, ser. LNAI, vol. 2922. Springer-Verlag, 2004, pp. 91–107, invited contribution.
[12] 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.
[13] R. v. 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`.
[14] F. Guerin and J. Pitt, "Verification and Compliance Testing," in *Communication in Multiagent Systems*, ser. LNAI, H. Huget, Ed., vol. 2650. Springer, 2003, pp. 98–112.
[15] G. 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.