Reasoning about logic-based agent interaction protocols

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)
E-mail: {baldoni,baroglio,mrt,patti}@di.unito.it

Abstract. The skill of reasoning about interaction protocols is very useful in many situations in the application framework of agent-oriented software engineering. In particular, we will tackle the cases of protocol selection, composition and implementation conformance w.r.t. an AUML sequence diagram. This work is based on DyLOG, an agent language based on modal logic that allows the inclusion, in the agent specification, of a set of interaction protocols.

1 Introduction

In Multi-Agent Systems (MASs) the communicative behavior of the agents plays a very important role, because it is the means by which agents cooperate for achieving a common goal or for competing for a resource. In order to rule communication, a set of shared protocols is commonly used. One of the most successful languages for designing them is AUML (Agent UML) [21]. This language is intuitive and easy to use for sketching the interactive behavior of a set of agents.

Our claim is that MAS engineering systems should encompass ways for obtaining declarative representations of protocols. The reason is that the use of declarative languages for protocol specification, although may be less intuitive, has the advantage of allowing the use of reasoning techniques in tasks like protocol validation or the verification of properties of the conversations within the system [13]. For instance, in [6] we proposed a logical framework, based on modal logic, that allows to include in an agent specification also a set of communication protocols. In this framework it is possible to reason about the effects of engaging specific conversations and to verify *properties* of the protocol: we can plan a conversation for achieving a particular goal, which respects the protocol, by checking if there is an execution trace of the protocol, after which the goal is satisfied. We can also verify if the composition of some protocols respects some desired constraint. Moreover, in [5, 7] we have shown how reasoning about conversation protocols can be used in an open application context where a personal assistant uses reasoning techniques for customizing the selection and the composition of web services.

The ability of reasoning about the properties of the interactions that occur among agents before they actually occur, may also by applied to support a MAS developer during the design phase of the MAS. For instance, the developer could be supported in the selection of already developed protocols from a library and in the verification of compositional properties. Another crucial problem, typical of this application framework and that we think could be tackled by means of reasoning techniques, is checking the *conformance* of a logic agent or of a protocol implementation to the specification given in AUML during the system design phase. Broadly speaking an agent is conformant to a given protocol if its behavior is always legal w.r.t. the protocol; more precisely conformance verification is interpreted as the problem of checking that an agent never performs any dialogue move that is not foreseen by the AUML specification.

In particular, in this work we survey over different problems that could be tackled by means of reasoning techniques: we sketch how protocol conformance could be verified, how by applying reasoning techniques it is possible to select a protocol from a catalogue of available AUML diagrams, and also how we could deal with issues arising from the composition of various protocols in the MAS.

The work is organized as follows. First of all we will briefly describe how speech acts and protocols can be represented in the DyLOG agent programming language. In Sections 3 and 4 we will describe some major issues inherent agent-oriented software engineering, and we will show by means of examples how reasoning techniques can be adopted for solving such problems. Conclusions follow.

2 Specification of interaction protocols in DyLOG

Logic-based executable agent specification languages have been deeply investigated in the last years [20]. In this section we will briefly introduce DyLOG, a high-level logic programming language for modeling and programming rational agents, based on a modal theory of actions and mental attitudes where modalities are used for representing actions as well as beliefs for modeling the agent's mental state [10, 6]. It accounts both for atomic and complex actions, or procedures. Atomic actions are either world actions, affecting the world, or mental actions, i.e. sensing and communicative actions which only affect the agent beliefs. Complex actions are defined through (possibly recursive) definitions, given by means of Prolog-like clauses and by making use of action operators like sequence, test and non-deterministic choice. The action theory allows to cope with the problem of reasoning about complex actions with incomplete knowledge and in particular to address the temporal projection and planning problem.

Intuitively DyLOG allows the specification of a rational agent that can reason about its own behavior, can choose a course of actions conditioned by its mental state, and can use sensors and communication for obtaining fresh knowledge. The language also allows agents to reason about their communicative behavior by means of techniques for proving existential properties of the kind: given a protocol and a set of desiderata, is there a specific conversation, that respects the protocol, which also satisfies the desired conditions? In the following we will describe how the communicative behavior of an agent can be represented in DyLOG and we will sketch the applicable reasoning techniques.

The DyLOG language supports communication both at the level of primitive speech acts and at the level of interaction protocols. Following the mentalistic approach, *speech acts* are considered as atomic actions, described in terms of preconditions and effects on the agent mental state, of form $\mathsf{speech\_act}(ag_i, ag_j, l)$, where ag_i (sender) and ag_j (receiver) are agents and l (a fluent) is the object of the communication. Since speech acts can be seen as mental actions, affecting both the sender's and the receiver's mental state, we have modeled them by generalizing non-communicative action definitions, so to allow also the representation of the effects of an action executed by some other agent on the current agent mental state, described by a consistent set of *belief fluents*. Actually, in DyLOG each agent has a twofold, personal representation of the speech act: one is to be used when it is the sender, the other when it is the receiver. Such a representation provides the capability of *reasoning about* conversation effects from the subjective point of view of the agent holding the representation. In the speech act specification that holds when the agent is the sender, the preconditions contain some *sincerity condition* that must hold in its mental state. When it is the receiver, instead, the action is always executable. As an example, let us define the semantics of the *inform* speech act within the DyLOG framework:

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

Clause (a) states that $Self$ will execute an inform act only if it believes l (we use the modal operator $\mathcal{B}^{ag_i}$ to model the beliefs of agent ag_i) and it believes that the receiver ($Other$) does not know l. It also considers possible that the receiver will adopt its belief (the modal operator $\mathcal{M}^{ag_i}$ is defined as the dual of $\mathcal{B}^{ag_i}$, intuitively $\mathcal{M}^{ag_i}\varphi$ means the ag_i considers φ possible), clause (b), although it cannot be certain about it -autonomy assumption-. If agent $Self$ believes to be considered a trusted *authority* about l by the receiver, it is also confident that $Other$ will adopt its belief, clause (c). Instead, 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, clauses (f).

DyLOG supports also the development of *conversation protocols*, that build on individual speech acts and specify communication patterns guiding the agent communicative behavior during a protocol-oriented dialogue. Reception of messages is modeled as a special kind of sensing action, what we call *get message actions*. Indeed receiving a message is interpreted as a query for an external input, whose outcome is unpredictable. The main difference w.r.t. normal sensing actions is that *get message actions* are defined by means of speech acts performed by the interlocutor. Protocols are thus expressed by means of a collection of procedure axioms of the action logic, having form $\langle p_0\rangle\varphi \subset \langle p_1\rangle\langle p_2\rangle \ldots \langle p_n\rangle\varphi$, where p_0 is the procedure name the p_i's can be i's communicative acts or special sensing actions for the reception of message. Each agent has a subjective perception of the communication with other agents, for this reason each protocol has as many procedural representations as the possible roles in the conversation. The importance of roles has been underlined also recently in works, such as [17].

Given a set $\Pi_{\mathcal{C}}$ of simple action laws defining an agent ag_i's primitive speech acts, a set $\Pi_{\mathcal{S}get}$ of axioms for the reception of messages, and a set $\Pi_{\mathcal{CP}}$, of procedure axioms specifying a collection of conversation protocols, we denote by CKit^{ag_i} (the *communication kit* of a DyLOG agent ag_i), the triple $(\Pi_{\mathcal{C}}, \Pi_{\mathcal{CP}}, \Pi_{\mathcal{S}get})$. CKit^{ag_i} is a part of Π_{ag_i}, i.e. the domain description of the agent ag_i, including also S_0, i.e the initial set of ag_i's belief fluents, and eventually laws and axioms for specifying the agent non communicative behaviors.

2.1 Reasoning about interaction protocols in DyLOG

Given a DyLOG domain description Π_{ag_i} containing a CKit^{ag_i} with the specifications of the interaction protocols and of the relevant speech acts, a *planning* activity can be triggered by *existential queries* of form $\langle p_1\rangle\langle p_2\rangle \ldots \langle p_m\rangle Fs$, where each p_k $(k = 1, \ldots, m)$ may be an atomic or complex action (a primitive speech act or an interaction protocol), executed by our agent, or an external[1] speech act, that belongs to CKit^{ag_i}. Checking if the query succeeds corresponds to answering to the question "is there an execution of p_1, ..., p_m leading to a state where the conjunction of belief fluents Fs holds for agent ag_i?". Such an execution is a plan to bring about Fs. The procedure definition constrains the search space. During the planning process get_message actions are treated as sensing actions, whose outcome cannot be predicted before the actual execution: since agents cannot read each other's mind, they cannot know in advance the answers that they will receive.

Depending on the task that one has to execute, it may alternatively be necessary to take into account all of the possible alternatives (which, we can foresee them because of the existence of the protocol) or just to find one of them. In the former case, the extracted plan will be *conditional*, in the sense that for each get_message 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 Fs. In the latter case, instead, the plan is linear.

3 Reasoning about protocols in MAS design

Generally speaking MASs are made of heterogeneous agents, which have different ways of representing their knowledge and adopt different mechanisms for

[1] By the word *external* we denote a speech act in which our agent plays the role of the receiver.

reasoning about it. Despite heterogeneity, agents need to interact and exchange information in order to cooperate or compete for the control of shared resources. This is obtained by means of interaction protocols, which result in being the *connective tissue* of the system. MAS engineering systems support the design of interaction protocols by means of graphical editors for the AUML language.

The AUML specification of a protocol is obtained by means of sequence diagrams [21], in which the interactions among the participants are modeled as message exchange and are arranged in time sequences. Sequence diagrams have two dimensions: the vertical (time) dimension specifying when a message is sent or expected, and the horizontal dimension that expresses the participants and their different roles. This kind of representation is very high-level and usually needs to be further specified in order to arrive to real implementations. In fact, the semantics of the atomic speech acts is not given by AUML; at implementation time, depending on the chosen ontology of speech acts, it may be necessary to express constraints or preconditions that depend on the agent mental state, that are not reported in the sequence diagrams. Since AUML sequence diagrams do not represent complete programs, it is not possible to automatically translate them in a way that fully expresses the communicative behavior of one or more agents in the application scenario. The protocol is to be *implemented*. On the other hand, given a protocol implementation it would be nice to have the possibility of automatically verifying its conformance to the desired protocol.

As mentioned in the introduction, a program is conformant to a protocol if all the message exchanges that it produces are foreseen by the protocol. The adoption of a logic formalism for implementing the protocols greatly simplifies this kind of verification, as we will see in the case of DyLOG in the next section. Differently of [11], the conformance property will be expressed from a language-theoretic point of view instead of from a logic point of view, by interpreting the problem of conformance verification as a problem of inclusion of a context-free language (CFL) into a regular language. In this process, the particular form of axiom, namely *inclusion axiom*, used to define protocol clauses in a DyLOG implementation, comes to help us. These axioms have interesting computational properties because they can be considered as *rewriting rules* [9, 3]. In [12] this kind of axioms is used for defining *grammar logics* and some relations between formal languages and such logics are analyzed.

On the side of protocol implementation, logic languages for reasoning about action and change, like DyLOG, seem particularly suitable. The reason is that although AUML accounts for a fast and intuitive prototyping in a graphical environment, it does not straightforwardly allow the *proof of properties* of the resulting system. Nevertheless, given the crucial role that protocols play, the ability of *reasoning about properties* of the interactions, occurring among the agents, is a key stone of the design and engineering of agent systems. By using DyLOG (or a similar language) it is possible to use the reasoning techniques embedded in the language for executing various tasks. Thus, besides the verification of the conformance of an implementation to a protocol, other possible applications of reasoning techniques include the intelligent use of libraries. In fact, it is not likely that the designer of a MAS redesigns every time new protocols, instead, he/she will more likely use *catalogues* of available protocols, in which searching for one of interest (notice that search could be based on the concept of conformance). In this case, the designer will need to know *before* the actual implementation of the agent, if the selected protocol fits some specific requisites of the system that is being developed. This problem can be interpreted as the problem of verifying if it is possible that a property of interest holds after the execution of the protocol in a given initial state. In this line, the designer may be interested in verifying if the *composition* of a set of protocols will be useful in the current application.

These observations are particularly relevant in the development of MAS prototyping systems (e.g. DCaseLP [19, 1]): in fact, a system which supports the design of the (communicative) interaction between the agents in a MAS only by means of AUML sequence diagrams, can carry on verifications only by animating the system and checking what happens in specific cases. A solution that has clear limitations since we get information only about the cases that have been tried. The worst limitation is that it is not possible to return to the system engineer the assumptions under which a goal is achieved or a property holds after the interaction.

4 Reasoning about protocols: some examples

Let us now illustrate the usefulness of reasoning techniques in the application domain of MAS design by means of examples. To this aim, consider this scenario: a MAS designer must develop a set of interaction protocols for a restaurant and a cinema that, for promotional reasons, will cooperate as described hereafter. A customer that makes a reservation at the restaurant will get a free ticket for a movie shown by the cinema. By restaurant and cinema we do not mean a specific restaurant or cinema but a generic service provider that will interact with its customers according to the protocol. Figure 1 reports an example of protocols, designed in AUML, for the two families of providers; the protocol described by (iii) is followed by the restaurant while the protocol described by (i) and (ii) is followed by the cinema. Let us describe the translation of the two protocols into a DyLOG representation. Observe that each of them has two complementary views: the view of the customer and the view of the provider. In this example we report only the view of the customer, which is what we need for the reasoning process. In the following, 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*, C for *customer*, and so on. The restaurant protocol is the following:

(a) $\langle \mathsf{reserv\_rest\_1}_C(Self, Service, Time)\rangle\varphi \subset$
$\quad\langle \mathsf{yes\_no\_query}_Q(Self, Service, available(Time))\ ;$
$\quad\quad \mathcal{B}^{Self} available(Time)?\ ;$
$\quad\quad \mathsf{get\_info}(Self, Service, reservation(Time))\ ;$
$\quad\quad \mathsf{get\_info}(Self, Service, cinema\_promo)\ ;$
$\quad\quad \mathsf{get\_info}(Self, Service, ft\_number)\rangle\varphi$

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

Procedure (a) describes the customer-view of the restaurant protocol. The customer asks if a table is available at a certain time, if so, the restaurant informs the customer that a reservation has been taken and, also, it informs the customer that it gained a promotional free ticket for a cinema (*cinema_promo*) and it returns a code number (*ft_number*). Clause (b) shows how get_info can be implemented as an inform act executed by the service and having as recipient the customer. In the DyLOG syntax the question mark corresponds to checking the value of a fluent in the current state while the semicolon is the sequencing operator of two actions. The cinema protocol, instead is:

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

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

Supposing that the desired movie is available, the cinema alternatively accepts credit card payments (c) or promotional tickets (d).

4.1 Conformance

Supposing that the designer produced the AUML sequence diagrams reported in Figure 1, and, then, implemented them in DyLOG, the first problem to solve is to check the conformance of the implementation w.r.t. the diagrams. For the sake of simplicity, we will sketch the method that we mean to follow focusing on one of the drawn protocols: the reserv_cinema_1 protocol.

In order to verify the conformance of a DyLOG interaction protocol to an AUML interaction protocol, that the DyLOG program is supposed to implement, we represent the AUML sequence diagram as a *formal language* by means of a *grammar*. More precisely, it is quite easy to see that, given a sequence diagram, it is possible to represent the set of the possible dialogues by means of a *regular*

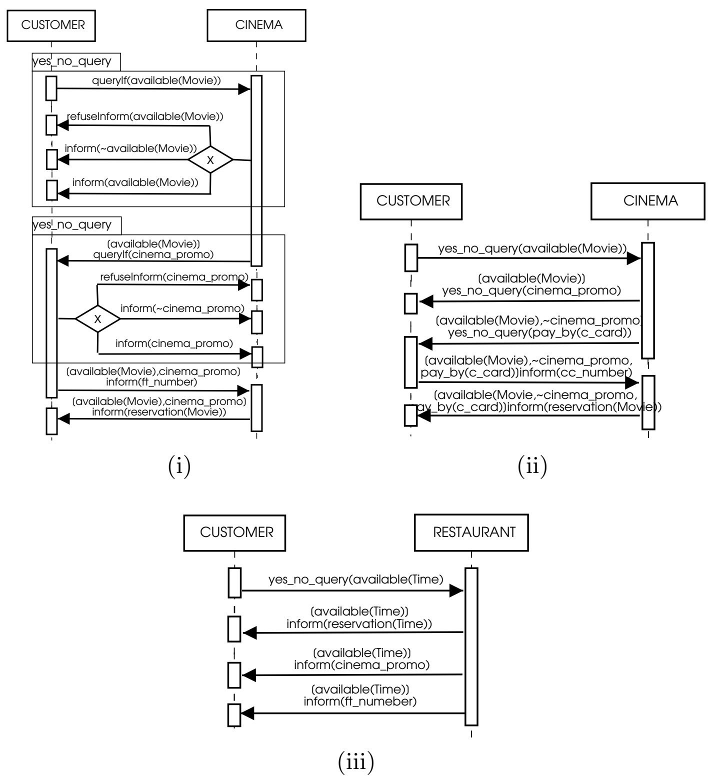
CUSTOMER
CINEMA
yes_no_query
queryIf(available(Movie))
refuseInform(available(Movie))
inform(~available(Movie))
X
inform(available(Movie))
yes_no_query
(available(Movie))
queryIf(cinema_promo)
refuseInform(cinema_promo)
inform(~cinema_promo)
X
inform(cinema_promo)
(available(Movie),cinema_promo)
inform(ft_number)
(available(Movie),cinema_promo)
inform(reservation(Movie))
(i)
CUSTOMER
CINEMA
yes_no_query(available(Movie))
(available(Movie))
yes_no_query(cinema_promo)
(available(Movie),~cinema_promo)
yes_no_query(pay_by(c_card))
(available(Movie),~cinema_promo,
pay_by(c_card))inform(cc_number)
(available(Movie,~cinema_promo,
pay_by(c_card))inform(reservation(Movie))
(ii)
CUSTOMER
RESTAURANT
yes_no_query(available(Time)
(available(Time))
inform(reservation(Time))
(available(Time))
inform(cinema_promo)
(available(Time))
inform(ft_numeber)
(iii)

Fig. 1. AUML interaction protocols representing the interactions between the customer and the provider: (i) and (ii) are followed by the cinema service, (iii) is followed by the restaurant. Formulas among square brackets represent conditions on the execution of the speech act.

grammar, whose set of atoms corresponds to the set of atomic speech acts. For example, let us consider the reserv_cinema_1 protocol (see Figure 1 (i)), the following grammar $G_{\mathsf{reserv\_cinema\_1}}$ represents it[2]:

$$
\begin{aligned}
&Q_0 \longrightarrow \mathsf{queryIf}(customer, cinema, available(Movie))\ Q_1\\
&Q_1 \longrightarrow \mathsf{refuseInform}(cinema, customer, available(Movie))\\
&Q_1 \longrightarrow \mathsf{inform}(cinema, customer, \neg available(Movie))\\
&Q_1 \longrightarrow \mathsf{inform}(cinema, customer, available(Movie))\ Q_2\\
&Q_2 \longrightarrow \mathsf{queryIf}(cinema, customer, cinema\_promo)\ Q_3\\
&Q_3 \longrightarrow \mathsf{refuseInform}(customer, cinema, cinema\_promo)\\
&Q_3 \longrightarrow \mathsf{inform}(customer, cinema, \neg cinema\_promo)\\
&Q_3 \longrightarrow \mathsf{inform}(customer, cinema, cinema\_promo)\ Q_4\\
&Q_4 \longrightarrow \mathsf{inform}(customer, cinema, ft\_number)\ Q_5
\end{aligned}
$$

[2] Details about the translation from AUML to the grammar can be found in [8].

$Q_5 \longrightarrow \mathsf{inform}(cinema, customer, reservation(Movie))$

By means of $L(G_{\mathsf{reserv\_cinema\_1}})$ we denote the language generated by it. Intuitively, it represents all the *legal* conversations.

Now, we are in the position to give our first definition of conformance: the *agent conformance*.

Definition 1 (Agent conformance). *Let $D = (\Pi, \mathsf{CKit}^{ag_i}, S_0)$ be a domain description, $\mathsf{p}_{dylog} \in \mathsf{CKit}^{ag_i}$ be an implementation of the interaction protocol p_{AUML} defined by means of an AUML sequence diagram. Moreover, let us define the set $\Sigma(S_0)$ as $\{\sigma | (\Pi, \mathsf{CKit}^{ag_i}, S_0) \vdash_{ps} \langle \mathsf{p}_{dylog} \rangle \top$ w. a. $\sigma\}$. We say that the agent described by means of D is* conformant w.r.t. the sequence diagram p_{AUML} *if and only if:*

$$\Sigma(S_0) \subseteq L(G_{\mathsf{p}_{AUML}}) \quad (1)$$

In other words, the agent conformance property holds if we can prove that every conversation, that is an instance of the protocol implemented in our language (an execution trace of p_{dylog}), is a legal conversation according to the grammar that represents the AUML sequence diagram p_{AUML}; that is to say, that conversation is also generated by the grammar $G_{\mathsf{p}_{AUML}}$.

The agent conformance depends on the initial state S_0. Different initial states can determine different possible conversations (execution traces). A notion of agent conformance, that is independent from the initial state, can also be defined:

Definition 2 (Agent strong conformance). *Let $D = (\Pi, \mathsf{CKit}^{ag_i}, S_0)$ be a domain description, let $\mathsf{p}_{dylog} \in \mathsf{CKit}^{ag_i}$ be an implementation of the interaction protocol p_{AUML} defined by means of an AUML sequence diagram. Moreover, let us define the set $\Sigma = \bigcup_S \Sigma(S)$, where S ranges over all possible initial states. We say that the agent described by means of D is* strongly conformant w.r.t. the sequence diagram p_{AUML} *if and only if:*

$$\Sigma \subseteq L(G_{\mathsf{p}_{AUML}}) \quad (2)$$

In other words, the agent strong conformance property holds if we can prove that every conversation for every possible initial state is a legal conversation, i.e. it is also generated by the grammar that represents the AUML sequence diagram. It is easy to see that agent strong conformance (2) implies agent conformance (1).

Agent strong conformance, differently than agent conformance, does not depend on the initial state but it still depends on the set of speech acts defined in CKit^{ag_i}. A stronger notion of conformance should require that a DyLOG implementation is conformant w.r.t. an AUML sequence diagram independently of the semantics of the speech acts. In other world, we would like to prove a sort of "structural" conformance of the implemented protocol w.r.t. the corresponding AUML sequence diagram. In order to do this, we define a formal grammar from the DyLOG implementation of a conversation protocol. In this process, the particular form of axiom, namely *inclusion axiom*, used to define protocol clauses in a DyLOG implementation, comes to help us. Actually, such axioms have a natural interpretation as rewriting rules [2, 12].

Given a domain description $(\Pi, \mathsf{CKit}^{ag_i}, S_0)$ and an conversation protocol $\mathsf{p}_{dylog} \in \mathsf{CKit}^{ag_i} = (\Pi_{\mathcal{C}}, \Pi_{\mathcal{CP}}, \Pi_{\mathcal{S}get})$, we define the grammar $G_{\mathsf{p}_{dylog}} = (T, V, P, S)$, where T is the set of all terms that define the set of speech acts in $\Pi_{\mathcal{C}}$, V is the set of all the terms that define a conversation protocol or a get message action in $\Pi_{\mathcal{CP}}$ or $\Pi_{\mathcal{S}get}$. P is the set of production rules of the form $p_0 \longrightarrow p_1 p_2 \ldots p_n$ where $\langle p_0 \rangle \varphi \supset \langle p_1 \rangle \langle p_2 \rangle \ldots \langle p_n \rangle \varphi$ is an axiom that defines either a conversation protocol (that belongs to $\Pi_{\mathcal{CP}}$) or a get message action (that belongs to $\Pi_{\mathcal{S}get}$). Note that test actions $\langle Fs? \rangle$ are not reported in the production rules. Finally, the start symbol S is the symbol p_{dylog}. Let us define $L(G_{\mathsf{p}_{dylog}})$ as the language generated by means of the grammar $G_{\mathsf{p}_{dylog}}$. It is easy to see that $L(G_{\mathsf{p}_{dylog}})$ is a context-free language since $G_{\mathsf{p}_{dylog}}$ is a context-free grammar. Intuitively, the language $L(G_{\mathsf{p}_{dylog}})$ represents all the possible sequences of speech acts (conversations) allowed by the DyLOG protocol p_{dylog} independently of the evolution of the mental state of the agent.

Definition 3 (Protocol conformance). *Let $D = (\Pi, \mathsf{CKit}^{ag_i}, S_0)$ be a domain description, let $\mathsf{p}_{dylog} \in \mathsf{CKit}^{ag_i}$ be an implementation of the interaction protocol p_{AUML} defined by means of an AUML sequence diagram. We say that p_{dylog} is* conformant to the sequence diagram p_{AUML} *if and only if:*

$$L(G_{\mathsf{p}_{dylog}}) \subseteq L(G_{\mathsf{p}_{AUML}}) \tag{3}$$

It is possible to prove that *protocol conformance* (3) implies *agent strong conformance* (2).

In this case, it is straightforward to prove that for the customer view $\mathsf{reserv\_cinema\_1}_C$ the protocol conformance holds w.r.t. protocol $\mathsf{reserv\_cinema\_1}$. It is worth noting the following property of the protocol conformance.

Proposition 1. *Protocol conformance is decidable.*

Proof. Equation (3) is equivalent to $L(G_{\mathsf{p}_{dylog}}) \cap \overline{L(G_{\mathsf{p}_{AUML}})} = \emptyset$. Now, $L(G_{\mathsf{p}_{dylog}})$ is a context-free language while $L(G_{\mathsf{p}_{AUML}})$ is a regular language. Since the complement of a regular language is still regular, $\overline{L(G_{\mathsf{p}_{AUML}})}$ is a regular language. The intersection of a context-free language and a regular language is a context-free language. For context-free languages, the emptiness is decidable [15].

Proposition 1 tells us that an algorithm for verifying protocol conformance exists. However, we also have a straightforward methodology for implementing protocols in a way that conformance w.r.t. the AUML specification is respected. In fact, we can build our implementation starting from the grammar $G_{\mathsf{p}_{AUML}}$, and applying the inverse of the process that we described for passing from a DyLOG implementation to the grammar $G_{\mathsf{p}_{dylog}}$. In this way we obtain a skeleton of a DyLOG implementation of p_{AUML} that is to be completed by adding the desired ontology for the speech acts and customized with tests. Such an implementation trivially satisfies protocol conformance and, then, all the other degrees of conformance defined above.

4.2 Composition

One example in which it is useful to reason about protocol composition, is the situation in which the same customer is supposed to interact with the restaurant and the cinema providers, one after the other. In fact, the developer must be sure that the customer, by interacting with the composition (by sequentialization) of the two protocols, will obtain what desired. In particular, suppose that the developer wants to verify if the protocols that he/she defined allow the following interaction: it is possible to make a reservation at the restaurant and, then, at the cinema, taking advantage of the promotion. Let us consider the query:

$$\begin{aligned}&\langle \mathsf{reserv\_rest\_1}_C(customer, restaurant, dinner) \ ; \\ &\quad \mathsf{reserv\_cinema\_1}_C(customer, cinema, movie)\rangle \\ &\qquad (\mathcal{B}^{customer} cinema\_promo \wedge \mathcal{B}^{customer} reservation(dinner) \wedge \\ &\qquad \mathcal{B}^{customer} reservation(movie) \wedge \mathcal{B}^{customer}\mathcal{B}^{C} ft\_number)\end{aligned}$$

that amounts to determine if it is possible to compose the interaction so to reserve a table for dinner ($\mathcal{B}^{customer} reservation(dinner)$) and to book a ticket for the movie *movie* ($\mathcal{B}^{customer} reservation(movie)$), exploiting a promotion ($\mathcal{B}^{customer} cinema\_promo$). The obtained free ticket is to be spent ($\mathcal{B}^{customer}$ $\mathcal{B}^{cinema}$ ft_number), i.e. *customer* believes that after the conversation the chosen cinema will know the number of the ticket given by the selected restaurant.

In the case in which the customer has neither a reservation for dinner nor a reservation for the cinema or a free ticket, the query succeeds, returning the following linear plan:

$\mathsf{queryIf}(customer, restaurant, available(dinner))$;
$\boxed{\mathsf{inform}(restaurant, customer, available(dinner)) \ ;}$
$\mathsf{inform}(restaurant, customer, reservation(dinner))$;
$\mathsf{inform}(restaurant, customer, cinema\_promo)$;
$\mathsf{inform}(restaurant, customer, ft\_number)$;
$\mathsf{queryIf}(customer, cinema, available(movie))$;
$\boxed{\mathsf{inform}(cinema, customer, available(movie)) \ ;}$
$\mathsf{queryIf}(cinema, customer, cinema\_promo)$;
$\mathsf{inform}(customer, cinema, cinema\_promo)$;
$\mathsf{inform}(customer, cinema, ft\_number)$;
$\mathsf{inform}(cinema, customer, reservation(movie))$

This means that there is first a conversation between *customer* and *restaurant* and, then, a conversation between *customer* and *cinema*, that are instances of the respective conversation protocols, after which the desired condition holds.

Notice that the linear plan will actually lead to the desired goal given that some assumptions about the provider's answers hold. In the above plan, assumptions have been outlined with a box. For instance, that a seat at the cinema is free. The difference with the other inform acts in the plan (from a provider to the customer) is that while for those the protocol does not offer any alternative, the outlined ones correspond just to one of the possible answers foreseen by the protocol. In the example they are answers foreseen by a yes_no_query protocol (see Figure 1 (i) and [6]). The actual answer can be known only at execution time, however, thanks to the existence of the protocol, it is possible to understand the conditions that lead to success.

4.3 Selection

Another situation in which the developer may need support is to search into a library of available policies for an interaction protocol that describes a service of interest and that is suitable to the application that he/she is designing. For instance, the developer must design a protocol for a restaurant that allows to make a reservation not necessarily using a credit card. In this case the developer will first search the library of available protocols looking for those that satisfy this request. This search cannot be accomplished only based on descriptive keywords but requires a form of reasoning on the way in which the interaction is carried on. Suppose that *search_service* is a procedure that allows one to search into a library for a protocol in a given category; then, the query would look like:

$$\langle search\_service(restaurant, Protocol)\ ;\ Protocol(customer, service, time)\rangle \\ (\mathcal{B}^{customer}\neg\mathcal{B}^{service} cc\_number \wedge \mathcal{B}^{customer} reservation(time))$$

which means: look for a protocol that has one possible execution, after which the service provider does not know the customer's credit card number, and a reservation has been taken.

5 Conclusions and related work

In this paper we have proposed a logic-based approach to agent interaction protocol specification and we have shown the advantages of using reasoning techniques based on such a logical formalization, especially in the context of agent-oriented software engineering (AOSE). We used as agent language DyLOG, a high-level logic programming language for modeling and programming rational agents, based on a modal theory of actions and mental attitudes, that allows to include in the agent specification a set of interaction protocols. The DyLOG language allows reasoning about the effects of engaging specific conversations. By doing so, an agent can plan a conversation for achieving a particular goal, that respects the protocol, while the system designer can exploit the same reasoning techniques to select a protocol from a library or to verify if the composition of a set of given protocols respects some desired constraint.

Moreover, we have shown that our logical representation of protocols allows us to deal with the matter of checking the agent conformance w.r.t. a protocol represented as a AUML interaction diagram. This is a crucial problem to face in an AOSE perspective and it can be considered as a part of the process of engineering interaction protocols sketched in [16]. Indeed, supposing to have both an

AUML sequence diagram, which formally specifies a protocol, and an implementation of the same protocol in DyLOG, a key problem to solve is checking the conformance of the implementation to the diagram. In fact, when protocols are implemented in DyLOG they become part of the agent communication policies, which are usually determined by the agent mental state. What we check is if this strategy is conformant to the AUML protocol, at least in the sense that the agent never performs any illegal dialogue move. In other words we prove a sort of "structural" conformance of the implemented protocol w.r.t the specification.

The problem of checking the agent conformance to a protocol in a logical framework has been faced also in [11]. In a broad sense our notion of conformance bears along some similarities with the notion of agent weak conformance, introduced in this work. In [11] agent communication strategies and protocol specification are both represented by means of sets of *if-then rules* in a logic-based language, which relies on abductive logic programming. A notion of weak conformance is introduced, which allows to check if the possible moves that an agent can make, according to a given communication strategy, are legal w.r.t. the protocol specification. The conformance test is done by abstracting from (i.e. disregarding) any condition related to the agent private knowledge, which is not considered as relevant in order to decide weak conformance, although it could, actually, prevent an agent from performing a particular move.

Our framework allows us to give a finer notion of conformance, for which we can distinguish different degrees of abstraction with respect to the agent private mental state. It allows us to define in an elegant manner which parts of a protocol implementation must fit the protocol specification and to describe in a modular way how the protocol implementation can be enriched with respect to the protocol specification, without compromising the conformance. Such an enrichment is important when using logic agents, that support sophisticated forms of reasoning. Indeed, the ability of reasoning about the properties of the interactions that occur among agents before they actually occur, may by applied to support a MAS developer during the design phase. We have recently begun to study [4] the methodological and physical integration of DyLOG into the DCaseLP [1, 18] MAS prototyping environment following the what already done for integrating tuProlog [14]. The aim of this work is to enrich the DCaseLP framework with the ability of reasoning about AUML interaction protocols thanks to a translation from AUML to DyLOG. In this context it will be extremely useful to have formal methods for proving conformance properties.

Acknowledgement

This research is partially supported by MIUR Cofin 2003 "Logic-based development and verification of multi-agent systems" national project.

References

1. E. Astesiano, M. Martelli, V. Mascardi, and G. Reggio. From Requirement Specification to Prototype Execution: a Combination of a Multiview Use-Case Driven

Method and Agent-Oriented Techniques. In J. Debenham and K. Zhang, editors, *Proceedings of the 15th International Conference on Software Engineering and Knowledge Engineering (SEKE'03)*, pages 578–585. The Knowledge System Institute, 2003.

2. M. Baldoni. *Normal Multimodal Logics: Automatic Deduction and Logic Programming Extension.* PhD thesis, Dipartimento di Informatica, Università degli Studi di Torino, Italy, 1998. Available at `http://www.di.unito.it/~baldoni/`.
3. M. Baldoni. Normal Multimodal Logics with Interaction Axioms. In D. Basin, M. D'Agostino, D. M. Gabbay, S. Matthews, and L. Viganò, editors, *Labelled Deduction*, volume 17 of *Applied Logic Series*, pages 33–53. Applied Logic Series, Kluwer Academic Publisher, 2000.
4. M. Baldoni, C. Baroglio, I. Gungui, A. Martelli, M. Martelli, V. Mascardi, V. Patti, and C. Schifanella. Reasoning about agents' interaction protocols inside dcaselp. In J. Leite, A. Omicini, P. Torroni, and P. Yolum, editors, *Proc. of the International Workshop on Declarative Language and Technologies, DALT'04*, New York, July 2004. To appear.
5. M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about Conversation Protocols in a Logic-based Agent Language. In A. Cappelli and F. Turini, editors, *AI*IA 2003: Advances in Artificial Intelligence, 8th Congress of the Italian Association for Artificial Intelligence*, volume 2829 of *LNAI*, pages 300–311. Springer, September 2003.
6. M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about self and others: communicating agents in a modal action logic. In C. Blundo and C. Laneve, editors, *Theoretical Computer Science, 8th Italian Conference, ICTCS'2003*, volume 2841 of *LNCS*, pages 228–241, Bertinoro, Italy, October 2003. Springer.
7. M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about interaction protocols for web service composition. In M. Bravetti and G. Zavattaro, editors, *Proc. of 1st Int. Workshop on Web Services and Formal Methods, WS-FM 2004*, Electronic Notes in Theoretical Computer Science. Elsevier Science Direct, 2004.
8. M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Verifying protocol conformance for logic-based communicating agents. Submitted, 2004.
9. M. Baldoni, L. Giordano, and A. Martelli. A Tableau Calculus for Multimodal Logics and Some (Un)Decidability Results. In H. de Swart, editor, *Proc. of TABLEAUX'98*, volume 1397 of *LNAI*, pages 44–59. Springer-Verlag, 1998.
10. 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), 2004. To appear.
11. Ulrich Endriss, Nicolas Maudet, Fariba Sadri, and Francesca Toni. Logic-based agent communication protocols. In F. Dignum, editor, *Advances in agent communication languages*, volume 2922 of *Lecture Notes in Artificial Intelligence (LNAI).* Springer-Verlag, 2004. To appear.
12. L. Fariñas del Cerro and M. Penttonen. Grammar Logics. *Logique et Analyse*, 121-122:123–134, 1988.
13. F. Guerin and J. Pitt. Verification and Compliance Testing. In H.P. Huget, editor, *Communication in Multiagent Systems*, volume 2650 of *LNAI*, pages 98–112. Springer, 2003.
14. I. Gungui and V. Mascardi. Integrating tuProlog into DCaseLP to engineer heterogeneous agent systems. Parma, Italy, June 2004. In this volume.
15. J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley Publishing Company, 1979.

16. 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.
17. S. Breuckner J. Odell, H. Van Dyke Parunak and M. Fleischer. Temporal aspects of dynamic role assignment. In J. Odell P. Giorgini, J. Müller, editor, *Agent-Oriented Software Engineering (AOSE) IV*, volume (forthcoming) of *LNCS*, Berlin, 2004. Springer.
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 F. Buccafurri, editor, *Proceedings of the 8th APPIA-GULP-PRODE Joint Conference on Declarative Programming (AGP'03)*, pages 275–286, 2003.
19. M. Martelli, V. Mascardi, and F. Zini. Caselp: a prototyping environment for heterogeneous multi-agent systems. Available at http://www.disi.unige.it/person/MascardiV/Download/aamas-journal-MMZ04.ps.gz.
20. V. Mascardi, M. Martelli, and L. Sterling. Logic-Based Specification Languages for Intelligent Software Agents. *Theory and Practice of Logic Programming Journal*, 2004. to appear.
21. J. Odell, H. V. D. Parunak, and B. Bauer. Extending UML for agents. In *Proceedings of the Agent-Oriented Information System Workshop at the 17th National Conference on Artificial Intelligence.* 2000.