Verifying Protocol Conformance for Logic-Based Communicating Agents*

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. Communication plays a fundamental role in multi-agents systems. One of the main issues in the design of agent interaction protocols is the verification that a given protocol implementation is "conformant" w.r.t. the abstract specification of it. In this work we tackle those aspects of the conformance verification issue, that regard the dependence/independence of conformance from the agent private state in the case of logic, individual agents, set in a multi-agent framework. We do this by working on a specific agent programming language, DyLOG, and by focussing on interaction protocol specifications described by AUML sequence diagrams. By showing how AUML sequence diagrams can be translated into regular grammars and, then, by interpreting the problem of conformance as a problem of language inclusion, we describe a method for automatically verifying a form of "structural" conformance; such a process is shown to be decidable and an upper bound of its complexity is given. We also give a set of properties that describes the influence of the agent private information on the conformance of its communication policies to protocol specifications.

1 Introduction

Multi-agent systems (MASs) often comprise heterogeneous components, different in the way they represent knowledge about the world and about other agents, as well as in the mechanisms used for reasoning about it. Notwithstanding heterogeneity, agents must cooperate, to execute a common task or compete for shared resources; interoperation is, normally, ruled by a set of shared *interaction protocols*. The design and implementation of interaction protocols are crucial steps in the development of a MAS. Following the development process, described in [21], for interaction protocol engineering, two different kinds of test are to be executed. The first consists in verifying the consistency of an *abstract protocol*

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

J. Leite and P. Torroni (Eds.): CLIMA V, LNAI 3487, pp. 196–212, 2005.

definition w.r.t. the original *requirements*, derived from the analysis phase, that it should embody. This verification, often called *validation test*, is typically done by means of model-checking techniques. A different problem is the one that we face in this work, which amounts to verify if a given *implementation*, which is an agent interaction policy, respects a given *abstract protocol definition*. This problem is known as *conformance testing*. Moreover, since the specific implementations[1] normally use the agent's private information (the agent's "state"), e.g. for deciding which utterances to articulate, one further question that arises is: to which extent does the agent internal state influence the conformance of an implementation to a given protocol specification? Indeed, depending on the result of tests on the agent's *internal state*, different executions could occur, only part of them being correct w.r.t. the specifications. Merely in the case in which the "structure" of the conversation policy is such that it is bound to produce correct conversations, conformance is not influenced at all by the agent's internal state (we will discuss this issue in Section 3).

In this work we tackle the problem of conformance verification for two specific languages: we use Agent UML (AUML for short, first specified in [27]) as the interaction protocol specification language and DyLOG [3, 5] as the conversation policy implementation language. In the literature one can, actually, find many formal techniques for protocol specification. A non-exhaustive list includes finite state automata [6, 24], petri nets [23, 11], temporal logic [15, 16] and UML-based languages. All these proposals are currently being studied and no definitive standard emerged yet. The reason for choosing AUML is that, despite its yet incomplete formal semantics (a proposal for a semantics based on petri nets can be found in [10]), this language bears some relevant advantages: it is based on the widespread and well-known UML standard, it is intuitive and easy to learn, there are graphical editors for the generation of code, and AUML sequence diagrams have been adopted by FIPA to represent agent interaction protocols. On the other hand, DyLOG is a logic language for programming agents, based on reasoning about actions and change in a modal framework, that allows the inclusion of a set of conversation policies, in the specification of an agent. The language refers to a mentalistic approach, where speech acts are represented as atomic actions with preconditions and effects on the executor's mental state. It allows the specification of *individual agents*, situated in a multi-agent context, each having a personal view of the world. The use of a declarative language is helpful because it allows the proof of properties of the *specific implementation* in a straightforward way. In particular, a language that explicitly represents and uses the agent internal state, is useful for proving to which extent certain properties depend on the agent mental state or on the semantics of the speech acts. For instance, in our work we perform hypothetical reasoning about the effects of conversations on the agent mental state, in order to find conversation plans which are proved to respect the implemented protocols, achieving at the same time some desired goal. The DyLOG language is briefly introduced in Section 2, for a thorough description of it see [5].

[1] In Java, in a logic language, etc.

Our goal for this work is, then, to study under which conditions a DyLOG implementation can be declared as being conformant to an AUML specification. To this aim, we will introduce different levels of abstraction w.r.t. the agent mental state by defining three degrees of conformance (*agent conformance*, *agent strong conformance*, and *protocol conformance*). We will describe their relations and, by interpreting the problem of conformance verification as a problem of inclusion of a context-free language (CFL) into a regular language, we will show a method for automatically verifying the strongest degree of conformance; an upper bound to its complexity is also given. When this kind of conformance holds, the implemented policy respects the specification whatever the rational effects of the speech acts are, whatever the agent mental state is.

As a last observation about AUML, some authors, who work on protocol validation, criticize the choice of this formalism because its lack of a formal semantics makes it difficult to validate the designed protocols w.r.t. the original specifications. They also criticize the choice of a mentalistic approach (á la FIPA) because at the level of protocol validation this approach has shown relevant flaws. The dissatisfaction to the mentalistic approach is mostly due to the difficulty of verifying that an agent acts according to a commonly agreed semantics, because it is not possible to have access to the agents' private mental state [30], a problem known as *semantics verification*. Some authors have proposed a *social approach* to agent communication [29], where communicative actions affect the "social state" of the system, rather than the internal states of the agents. The social state records the social facts, like the *permissions* and the *commitments* of the agents, which are created and modified along the interaction. The social approach overcomes the semantics verification problem by exploiting a set of established commitments between the agents, that are stored as part of the MAS social state. In this framework it is possible to formally prove the correctness of public interaction protocols with respect to the specifications outcoming from the analysis phases; such proof can be obtained, for instance, by means of model checking techniques [22, 25, 28, 30, 18, 7] (but not only, e.g., [9]).

Nevertheless, AUML is being used, more and more often, in MAS development because it is intuitive for designers that have a background in UML and in the object-oriented approach, and for this reason it has an appeal for the deployment of agent systems in the industry world. Moreover, when developing the single agents, besides verifying that the agent respects the social commitments, it is important to study properties of their implementations and, in particular, to understand if and to which extent such properties depend on the agent's internal state (in the case of communication, on the semantics of the speech acts).

2 Specification of Communication in DyLOG

DyLOG [5] is a high-level logic programming language for modeling rational agents, based upon a modal logic of actions and mental attitudes where modalities are used for representing 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. A DyLOG agent can be provided with a *communication kit* that specifies its communicative behavior [3], defined in terms of interaction protocols, i.e. conversation policies that build on FIPA-like speech acts. The communication theory is a homogeneous component of the general agent theory; in particular, both the conversational policies, that guide the agent's communicative behavior, and the other policies, defining the agent's behavior, are represented by procedure definitions (represented by *axiom schema*). DyLOG agents can reason about their communicative behavior answering to queries like "given a protocol and a set of desiderata, is there a conversation, that respects the protocol, which also satisfies the desired conditions on the final mental state?".

2.1 The DyLOG Language in Brief

In DyLOG atomic actions are either world actions, affecting the world, or mental actions, i.e. sensing or communicative actions which only affect the agent beliefs. The set of atomic actions consists of the set $\mathcal{A}$ of the world actions, the set $\mathcal{C}$ of communicative acts, and the set $\mathcal{S}$ of sensing actions. For each atomic action a and agent ag_i we introduce the modalities $[a^{ag_i}]$ and $\langle a^{ag_i}\rangle$. $[a^{ag_i}]\alpha$ means that α holds after every execution of action a by agent ag_i; $\langle a^{ag_i}\rangle\alpha$ means that there is a possible execution of a (by ag_i) after which α holds. We use the modality $\Box$ to denote *laws*, i.e. formulas that hold always (after every action sequence). Our formalization of complex actions draws considerably from dynamic logic for the definition of action operators like sequence, test and non-deterministic choice. However, differently than [26], we refer to a *Prolog-like* paradigm: procedures are defined by means of (possibly recursive) Prolog-like clauses. For each procedure p, the language contains also the universal and existential modalities $[p]$ and $\langle p\rangle$. The mental state of an agent is described by a consistent set of *belief formulas* (we call it *belief state*). We use the modal operator $\mathcal{B}^{ag_i}$ to model the beliefs of agent ag_i. The modality $\mathcal{M}^{ag_i}$ is defined as the dual of $\mathcal{B}^{ag_i}$ and means that ag_i considers φ possible. A mental state contains what ag_i (dis)believes about the world and about the other agents (nested beliefs are needed for reasoning on how other agents beliefs can be affected by communicative actions). Formally it is a complete and consistent set of rank 1 and 2 belief fluents, where a *belief fluent* F is a belief formula $\mathcal{B}^{ag_i}L$ or its negation. L denotes a *belief argument*, i.e. a *fluent literal* l (f or $\neg f$) or a belief fluent of rank 1 ($\mathcal{B}l$ or $\neg\mathcal{B}l$).

All the modalities of the language are normal; $\Box$ is reflexive and transitive, its interaction with action modalities is ruled by $\Box\varphi \supset [a^{ag_i}]\varphi$. The epistemic modality $\mathcal{B}^{ag_i}$ is serial, transitive and euclidean. A non-monotonic solution to the persistency problem is given, which consists in maximizing assumptions about fluents after the execution of action sequences, based on an abductive framework.

2.2 The Communication Kit in Brief

The *behavior* of an agent ag_i is specified by a domain description, which includes, besides a specification of the agent *belief state*: (i) *action and precondition laws* for describing the atomic world actions in terms of their preconditions and their

affects on the agent's mental state, (ii) *sensing axioms* for describing atomic sensing actions, (iii) *procedure axioms* for describing complex behaviors, (iv) a *communication kit* that describes the agent communicative behavior by means of further axioms and laws of the kind mentioned above. In fact a *communication kit* consists of (i') a set of action and preconditions laws modeling a predefined set of primitive speech acts the agent can perform/recognize (ii') a set of sensing axioms for defining special sensing actions for getting new information by external communications (iii') a set of procedure axioms for specifying interaction protocols, which can be seen as a library of conversation policies the agent can follow when engaging a conversations with others.

Interaction Protocols are represented as procedures that build upon individual speech acts and specify conversation policies for guiding the agent communicative behavior. They are expressed by *axiom schema* of form:

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

p_0 is a procedure name and the p_i's ($i = 1, \ldots, m$) are either procedure names, atomic actions, or test actions (actions of the form $Fs?$, where Fs is a belief fluent conjunction); intuitively, the ? operator corresponds to checking the value of a fluent conjunction in the current state while the ; is the sequencing operator of dynamic logic. Since each agent has a subjective perception of the communication with other agents, given a protocol specification we expect to have as many procedural representations as the possible roles in the conversation.

The axiom schema used to define procedures have the form of *inclusion axioms*, which were the subject of previous work [4, 2], in which the class of multimodal logics, characterized by axioms that have the general form $\langle s_1 \rangle \ldots \langle s_m \rangle \varphi \subset \langle t_1 \rangle \ldots \langle t_n \rangle \varphi$, where $\langle s_i \rangle$ and $\langle t_i \rangle$ are modal operators, has been analyzed. These axioms have interesting computational properties because they can be considered as *rewriting rules*. In [14] this kind of axioms is used for defining *grammar logics* and some relations between formal languages and such logics are analyzed.

A speech act c in $\mathcal{C}$ has form $\mathsf{speech\_act}(ag_s, ag_r, l)$, where ag_s (sender) and ag_r (receiver) are agents and l is the message content. Effects and preconditions are modeled by a set of effect and precondition laws. In particular, *effects* on ag_i's belief state of an action c are expressed by *action laws* of form:

$$\Box(\mathcal{B}^{ag_i} L_1 \wedge \ldots \wedge \mathcal{B}^{ag_i} L_n \supset [c^{ag_i}] \mathcal{B}^{ag_i} L_0) \tag{2}$$

$$\Box(\mathcal{M}^{ag_i} L_1 \wedge \ldots \wedge \mathcal{M}^{ag_i} L_n \supset [c^{ag_i}] \mathcal{M}^{ag_i} L_0) \tag{3}$$

Law (2) means that, after any sequence of actions ($\Box$), if the set of fluent literals $L_1 \wedge \ldots \wedge L_n$ (representing the preconditions of the action c) is believed by ag_i then, after the execution of c, L_0 (the effect of c) is also believed by ag_i. Notice that our representation of speech acts models only the dynamics of the mental state of the agent the we are implementing. Executing a speech act may cause an agent to have new beliefs (in its mental state), that are assumptions on what the others believe but the agent cannot be sure that the others actually have those beliefs. Law (3) states that when the preconditions of c are unknown to ag_i, after the execution of c, it will consider unknown also its effects. *Precondition laws*

specify mental conditions that make an action in $\mathcal{C}$ executable in a state. They have form:

$$\Box(\mathcal{B}^{ag_i} L_1 \wedge \ldots \wedge \mathcal{B}^{ag_i} L_n \supset \langle c^{ag_i} \rangle \top) \tag{4}$$

ag_i can execute c when its precondition fluents are in ag_i's belief state.

Get message actions are formalized as sensing actions, i.e. knowledge producing actions whose outcome cannot be predicted before the execution. In fact, from the perspective of the individual agent, expecting a message corresponds to query for an external input, thus it is natural to think of it as a special case of sensing. A get_message action is defined by the *inclusion axiom schema*:

$$[\mathsf{get\_message}(ag_i, ag_j, l)]\varphi \equiv [\bigcup_{\mathsf{speech\_act} \in \mathcal{C}_{\mathsf{get\_message}}} \mathsf{speech\_act}(ag_j, ag_i, l)]\varphi \tag{5}$$

Intuitively, $\mathcal{C}_{\mathsf{get\_message}}$ is a finite set of speech acts, which are all the possible communications that ag_i could expect from ag_j in the context of a given conversation. Hence, the information that can be obtained is calculated by looking at the effects of the speech acts in $\mathcal{C}_{\mathsf{get\_message}}$ on ag_i's mental state.

As a finale comment, in the operational interpretation of the language, (1) is handled as a rewriting rule. From a declarative semantics point of view, the rule is an axiom schema of the logic, hence its form. Intuitively, the set of formulas of kind (2), (3) and (4) define the theory, while those of form (1) and (5) define the characteristics of the multi-modal logic in which the formulas are interpreted.

Example 1. The following procedure axioms represent an implementation of the protocol in Fig. 1 as the conversation policy that the customer agent (*cus*) must use for interacting with the service provider (*sp*). Axioms implementing the query subprotocol follow. Since the AUML protocol contains two roles, the customer and the provider, the implementation must contain two views as well but for brevity we report only the view of the customer ($\mathsf{get\_cinema\_ticket}_C$). Similarly for the subprotocol for querying information: $\mathsf{yes\_no\_query}_Q$ implements the role of the querier and $\mathsf{yes\_no\_query}_I$ the one of the responder[2].

(a) $\langle \mathsf{get\_cinema\_ticket}_C(cus, sp, movie)\rangle\varphi \subset$
$\langle \mathsf{yes\_no\_query}_Q(cus, sp, available(movie))$;
$\mathcal{B}^{cus} available(movie)?$; $\mathsf{get\_info}(cus, sp, cinema(c))$;
$\mathsf{yes\_no\_query}_I(cus, sp, pay\_by(credit\_card))$;
$\mathcal{B}^{cus} pay\_by(credit\_card)?$; $\mathsf{inform}(cus, sp, cc\_number)$;
$\mathsf{get\_info}(cus, sp, booked(movie))\rangle\varphi$

(b) $\langle \mathsf{get\_cinema\_ticket}_C(cus, sp, movie)\rangle\varphi \subset$
$\langle \mathsf{yes\_no\_query}_Q(cus, sp, available(movie))$; $\mathcal{B}^{cus} available(movie)?$;
$\mathsf{get\_info}(cus, sp, cinema(c))$;
$\mathsf{yes\_no\_query}_I(cus, sp, pay\_by(credit\_card))$; $\neg\mathcal{B}^{cus} pay\_by(credit\_card)?\rangle\varphi$

[2] The subscripts next to the protocols names are a writing convention for representing the role that the agent plays: Q stands for *querier*, I stands for *informer*, C for *customer*.

(c) $\langle \mathsf{get\_cinema\_ticket}_C(cus, sp, movie)\rangle\varphi \subset$
$\langle \mathsf{yes\_no\_query}_Q(cus, sp, available(movie)); \neg\mathcal{B}^{cus} available(movie)?\rangle\varphi$
(d) $[\mathsf{get\_info}(cus, sp, Fluent)]\varphi \equiv [\mathsf{inform}(sp, cus, Fluent)]\varphi$

Protocol $\mathsf{get\_cinema\_ticket}_C$ works as follows: agent *cus* begins the interaction. After checking if the requested movie is available by the $\mathsf{yes\_no\_query}_Q$ protocol, it waits for an information (get_info) from the provider (*sp*) about which cinema shows it. Then, the provider asks for a payment by credit card by using the $\mathsf{yes\_no\_query}_I$ protocol. If the answer if positive *cus* communicates the credit card number and the confirmation of the ticket booking is returned to it, otherwise clause (b) is selected, ending the conversation. Clause (c) tackles the case in which the movie is not available; clause (d) describes get_info, which is a get_message action. In the following the get_answer and get_start definitions are instances of axiom schema (5): the right hand side of get_answer represents all the possible answers expected by *cus* from *sp* about *Fluent*, during a conversation ruled by $\mathsf{yes\_no\_query}_Q$.

(e) $\langle \mathsf{yes\_no\_query}_Q(cus, sp, Fluent)\rangle\varphi \subset$
$\langle \mathsf{queryIf}(cus, sp, Fluent); \mathsf{get\_answer}(cus, sp, Fluent)\rangle\varphi$
(f) $[\mathsf{get\_answer}(cus, sp, Fluent)]\varphi \equiv [\mathsf{inform}(sp, cus, Fluent) \cup$
$\mathsf{inform}(sp, cus, \neg Fluent) \cup \mathsf{refuseInform}(sp, cus, Fluent)]\varphi$
(g) $\langle \mathsf{yes\_no\_query}_I(cus, sp, Fluent)\rangle\varphi \subset \langle \mathsf{get\_start}(cus, sp, Fluent);$
$\mathcal{B}^{cus} Fluent?; \mathsf{inform}(cus, sp, Fluent)\rangle\varphi$
(h) $\langle \mathsf{yes\_no\_query}_I(cus, sp, Fluent)\rangle\varphi \subset \langle \mathsf{get\_start}(cus, sp, Fluent);$
$\mathcal{B}^{cus}\neg Fluent?; \mathsf{inform}(cus, sp, \neg Fluent)\rangle\varphi$
(i) $\langle \mathsf{yes\_no\_query}_I(cus, sp, Fluent)\rangle\varphi \subset \langle \mathsf{get\_start}(cus, sp, Fluent);$
$\mathcal{U}^{cus} Fluent?; \mathsf{refuseInform}(cus, sp, Fluent)\rangle\varphi$
(j) $[\mathsf{get\_start}(cus, sp, Fluent)]\varphi \equiv [\mathsf{queryIf}(sp, cus, Fluent)]\varphi$

Given a set $\Pi_{\mathcal{C}}$ of action and precondition laws defining the 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 for specifying conversation protocols, we denote by CKit^{ag_i} the *communication kit* of an agent ag_i, that is the triple $(\Pi_{\mathcal{C}}, \Pi_{\mathcal{CP}}, \Pi_{\mathcal{S}get})$.

A *domain description* (DD) for agent ag_i, is a triple $(\Pi, \mathsf{CKit}^{ag_i}, S_0)$, where CKit^{ag_i} is ag_i's communication kit, S_0 is the initial set of ag_i's belief fluents, and Π is a tuple $(\Pi_{\mathcal{A}}, \Pi_{\mathcal{S}}, \Pi_{\mathcal{P}})$, where $\Pi_{\mathcal{A}}$ is the set of ag_i's world action and precondition laws, $\Pi_{\mathcal{S}}$ is a set of axioms for ag_i's sensing actions, $\Pi_{\mathcal{P}}$ a set of axioms that define the complex non-communicative behavior of the agent.

From a DD 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[3] speech act, that belongs to CKit^{ag_i}. In [3] we

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

presented a goal-directed proof procedure for the language based on negation as failure (NAF) which allows query of form $\langle p_1\rangle\langle p_2\rangle \ldots \langle p_m\rangle Fs$ to be proved from a given domain description and returns as answer an action sequence. A query of the form $\langle p_1; p_2; \ldots; p_n\rangle Fs$, where p_i, $1 \leq i \leq n$ ($n \geq 0$), is either a world action, or a sensing action, or a procedure name, or a test, succeeds if it is possible to execute $p_1, p_2, \ldots, p_n$ (in the order) starting from the current state, in such a way that Fs holds at the resulting state. In general, we will need to establish if a goal holds at a given state. Hence, we will write:

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

to mean that the query $\langle p_1; p_2; \ldots; p_n\rangle Fs$, i.e. $\langle p_1\rangle\langle p_2\rangle \ldots \langle p_n\rangle Fs$, can be proved from the DD $(\Pi, \mathsf{CKit}^{ag_i}, S_0)$ at the state $a_1, \ldots, a_m$ with answer σ, where σ is an action sequence $a_1, \ldots, a_m, \ldots a_{m+k}$ which represents the state resulting by executing $p_1, \ldots, p_n$ in the current state $a_1, \ldots, a_m$. ε denotes the initial state.

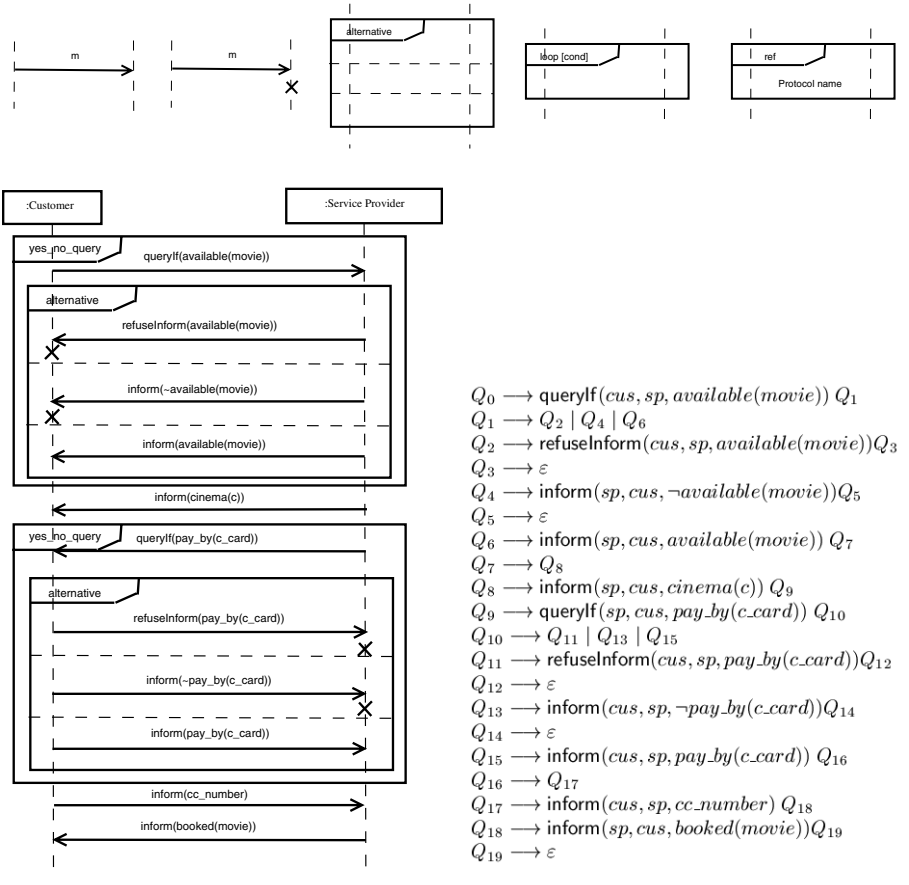
3 Protocol Conformance

In AUML a protocol is specified by means of sequence diagrams [27], which model the interactions among the participants as message exchanges, arranged in time sequences. The vertical (time) dimension specifies when a message is sent (expected), the horizontal dimension expresses the participants and their roles. The current proposal [17], enriches the set of possible operators of the language; particularly interesting is the possibility of representing loops, calls to subprotocols, and exit points. Generally speaking, given a protocol implementation it would be nice to have a means for automatically verifying its *conformance* to the desired AUML specification. The technique that we follow consists in turning this problem into a problem of formal language inclusion. To this aim, given a sequence diagram, we define a formal grammar which generates a language, that is the set of all the conversations allowed by the diagram itself. The algorithm used to this purpose is described in Section 3.1. On the other hand, given a DyLOG implementation of a protocol, we define a language that is compared to the previously obtained one: if the language obtained from the implementation is included in the one obtained from the sequence diagram we conclude that a type of conformance holds. We, actually, define three degrees of conformance (*agent conformance*, *agent strong conformance*, and *protocol conformance*), characterized by different levels of abstraction from the agent private mental state, which correspond to different ways of extracting the language from the implementation. These definitions allow us to define which parts of a protocol implementation must fit the specification and to describe in a modular way how the implementation can be enriched with respect to the specification, without compromising the conformance. Such an enrichment is important when using logic agents, that support sophisticated forms of reasoning.

3.1 Turning an AUML Sequence Diagram into a Linear Grammar

In the following we show how it is possible to translate an AUML sequence diagram, as defined in [17], into a grammar. Using the notation of [20], a grammar G is a tuple (V, T, P, S), where V is a set of variables, T a set of terminals, P of production rules, and S is the start symbol. By $L(G)$ we will denote the language generated by grammar G, i.e. the set of sentences in T^* that are generated starting from S, by applying rules in P.

On the side of sequence diagrams we focus on the operators used to specify FIPA protocols, which are: message, alternative, loop, exit, and reference to a sub-protocol (see top of Fig. 1).



$Q_0 \longrightarrow \mathsf{queryIf}(cus, sp, available(movie))\ Q_1$
$Q_1 \longrightarrow Q_2 \mid Q_4 \mid Q_6$
$Q_2 \longrightarrow \mathsf{refuseInform}(cus, sp, available(movie))Q_3$
$Q_3 \longrightarrow \varepsilon$
$Q_4 \longrightarrow \mathsf{inform}(sp, cus, \neg available(movie))Q_5$
$Q_5 \longrightarrow \varepsilon$
$Q_6 \longrightarrow \mathsf{inform}(sp, cus, available(movie))\ Q_7$
$Q_7 \longrightarrow Q_8$
$Q_8 \longrightarrow \mathsf{inform}(sp, cus, cinema(c))\ Q_9$
$Q_9 \longrightarrow \mathsf{queryIf}(sp, cus, pay\_by(c\_card))\ Q_{10}$
$Q_{10} \longrightarrow Q_{11} \mid Q_{13} \mid Q_{15}$
$Q_{11} \longrightarrow \mathsf{refuseInform}(cus, sp, pay\_by(c\_card))Q_{12}$
$Q_{12} \longrightarrow \varepsilon$
$Q_{13} \longrightarrow \mathsf{inform}(cus, sp, \neg pay\_by(c\_card))Q_{14}$
$Q_{14} \longrightarrow \varepsilon$
$Q_{15} \longrightarrow \mathsf{inform}(cus, sp, pay\_by(c\_card))\ Q_{16}$
$Q_{16} \longrightarrow Q_{17}$
$Q_{17} \longrightarrow \mathsf{inform}(cus, sp, cc\_number)\ Q_{18}$
$Q_{18} \longrightarrow \mathsf{inform}(sp, cus, booked(movie))Q_{19}$
$Q_{19} \longrightarrow \varepsilon$

Fig. 1. On top a set of AUML operators is shown. Below, on the left the sequence diagram, representing the interaction protocol between a cinema booking service and a customer, is reported with its corresponding production rules

Algorithm 1 (Generating $G_{p_{AUML}}$) The grammar construction is done in the following way. We will denote by the variable *last* the most recently created variable. Initially T and P are empty. Define the start symbol as Q_0, initialize *last* to Q_0 and $V := \{Q_0\}$, then, we apply the translation rules described by cases hereafter, according to the sequence given by the AUML diagram:

- given a *message arrow*, labeled by m, create a new variable Q_{new}, $V := V \cup \{Q_{new}\}$, $T := T \cup \{\mathsf{m}\}$, $P := P \cup \{last \longrightarrow \mathsf{m}Q_{new}\}$, finally, set $last := Q_{new}$;
- given an *exit operator*, add to P a production $last \longrightarrow \epsilon$, $last :=\perp$ (undefined);
- given an *alternative* operator with n branches, apply to each branch the grammar construction algorithm, so obtaining a grammar $G'_i = (V'_i, T'_i, P'_i, S'_i)$ with $last'_i$ being the last variable used inside that branch. Let us assume that $V'_1 \cap \ldots \cap V'_n \cap V = \emptyset$ (it is sufficient to rename all variables in the V'_i's), then create a new variable Q_{new}. $V := V \cup V'_1 \cup \ldots \cup V'_n \cup \{Q_{new}\}$, $T := T \cup T'_1 \cup \ldots T'_n$, $P := P \cup P'_1 \cup \{last \longrightarrow S'_1\} \cup \ldots \cup P'_n \cup \{last \longrightarrow S'_n\}$, moreover $P := P \cup \{last'_i \longrightarrow Q_{new}\}$ for each $i \in [1, n]$ such that $last'_i \neq \perp$, and finally we set *last* to Q_{new};
- given a *loop*, apply the grammar construction algorithm to its body, so obtaining a grammar $G' = (V', T', P', S')$ with a value for $last'$. Let us assume that $V' \cap V = \emptyset$, then create Q_{new}, $V := V \cup V' \cup \{Q_{new}\}$, $T := T \cup T'$, $P := P \cup P' \cup \{Q_{last} \longrightarrow S', last \longrightarrow Q_{new}\}$ if $last' \neq \perp$ then $P := P \cup \{last' \longrightarrow last\}$, and $last := Q_{new}$;
- given a *subprotocol reference*, apply the grammar construction algorithm to the called subprotocol, so obtaining a grammar $G' = (V', T', P', S')$ with a value for $last'$. Let us assume that $V' \cap V = \emptyset$, then increment *new*, create Q_{new}, $V := V \cup V' \cup \{Q_{new}\}$, $T := T \cup T'$, $P := P \cup P' \cup \{Q_{last} \longrightarrow S'\}$, if $last' \neq \perp$ then $P := P \cup \{last' \longrightarrow Q_{new}\}$, and $last := Q_{new}$;

Proposition 1. *The set of conversations allowed by an AUML sequence diagram is a regular language.*

Proof. The Algorithm 1 produces a *right linear grammars* (variables appear only at the rigth end of productions), so the generated language is *regular* [20].

By this translation we give to the set of conversations encoded by the sequence diagram a structural semantics (although no semantics is given to the single speech acts). The grammar could, then, be translated into a finite-state automaton, another formal tool often used to represent interaction protocols, as mentioned in the introduction. As a last observation, the produced grammar may contain redundancies and could be simplified using standard algorithms [20].

Consider, as an example, the sequence diagram in Fig. 1: it represents an interaction protocol with two agent roles (Customer, *cus*, and Service Provider, *sp*): the protocol rules the interaction of a cinema booking service with each of its customers, and will be used as a running example along the paper. Suppose, now, to have a DyLOG implementation of the specification given by the diagram. The technique that we apply for verifying if it is conformant (w.r.t. the definitions given in Section 3) to the specifications, intuitively works as follows. If

we can prove that all the conversations produced by the implementation belong to the language generated by the grammar into which the specification can be translated (see Fig. 1), then the implementation can be considered conformant.

3.2 Three Degrees of Conformance

We have shown how AUML sequence diagrams can be translated into regular grammars. By interpreting the problem of conformance as a problem of formal language inclusion, we will describe a method for automatically verifying the strongest of the three degrees of conformance (protocol conformance). The verification of protocol conformance is shown to be decidable and an upper bound of its complexity is given.

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*

$$\Sigma(S_0) = \{\sigma \mid (\Pi, \mathsf{CKit}^{ag_i}, S_0) \vdash \langle \mathsf{p}_{dylog} \rangle \top \ w.\ a.\ \sigma\}$$

We say that the agent described by means of D *is* conformant to the sequence diagram p_{AUML} *if and only if*

$$\Sigma(S_0) \subseteq L(G_{\mathsf{p}_{AUML}}) \tag{6}$$

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). One can define a notion of agent conformance that is independent from the initial state.

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 to the sequence diagram p_{AUML} *if and only if*

$$\Sigma \subseteq L(G_{\mathsf{p}_{AUML}}) \tag{7}$$

The agent strong conformance property holds if we can prove that every conversation for every possible initial state is a legal conversation. It is obvious by definition that agent strong conformance (7) implies agent conformance (6).

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}. In fact, an execution trace σ is built taking into account test actions and the semantics of the speech acts (defined by executability precondition laws and action laws).

A stronger notion of conformance should require that a DyLOG implementation is conformant to an AUML sequence diagram *independently from 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.

Algorithm 2 (Generating $G_{p_{dylog}}$) Given a domain description $(\Pi, \mathsf{CKit}^{ag_i}, S_0)$ and a 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 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 $p_0 \longrightarrow p_1 p_2 \dots p_n$ where $\langle p_0 \rangle \varphi \subset \langle p_1; p_2; \dots; p_n \rangle \varphi$ 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, in the latter case, we add a production rule for each alternative speech act in $\mathcal{C}_{\mathsf{get\_message}}$ see (5), moreover, the test actions $Fs?$ are not reported in the production rules;
- 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 $G_{\mathsf{p}_{dylog}}$.

Proposition 2. *Given a domain description $(\Pi, \mathsf{CKit}^{ag_i}, S_0)$ and a conversation protocol $\mathsf{p}_{dylog} \in \mathsf{CKit}^{ag_i} = (\Pi_{\mathcal{C}}, \Pi_{\mathcal{CP}}, \Pi_{\mathcal{S}get})$, $L(G_{\mathsf{p}_{dylog}})$ is a context-free language.*

Proof. The proposition follows from the fact that $G_{\mathsf{p}_{dylog}}$ is a context-free grammar (CFG).

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 from the evolution of the mental state of the agent. For example, clause (a) of $\mathsf{get\_cinema\_ticket}_C$ presented in the previous section is represented as follows:

$\mathsf{get\_cinema\_ticket}_C(cus, sp, movie) \longrightarrow$
$\quad \mathsf{yes\_no\_query}_Q(cus, sp, available(movie))$
$\quad \mathsf{get\_info}(cus, sp, cinema(c))$
$\quad \mathsf{yes\_no\_query}_I(cus, sp, pay\_by(credit\_card))$
$\quad inform(cus, sp, cc\_number)$
$\quad \mathsf{get\_info}(cus, sp, booked(movie))$

Definition 3 (Protocol conformance). *Given a domain description* $DD = (\Pi, \mathsf{CKit}^{ag_i}, S_0)$*, 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*

$$L(G_{\mathsf{p}_{dylog}}) \subseteq L(G_{\mathsf{p}_{AUML}}) \tag{8}$$

We then interpret the verification of conformance as a containment of formal languages problem; in particular, that a CFL is included in a regular language. By doing so, we verify the structural matching of the implementation to the specification.

Proposition 3. Protocol conformance *(8) implies* agent strong conformance *(7) and the latter implies* agent conformance *(6).*

Proof. It is sufficient to prove that $\Sigma \subseteq L(G_{p_{dylog}})$. We give a sketch of proof. Actually, let us consider the application of proof rule (1) and (4) in the proof of $(\Pi, \mathsf{CKit}^{ag_i}, S_0) \vdash_{ps} \langle \mathsf{p}_{dylog} \rangle \top$ w.a. σ, it is possible to build a derivation $\mathsf{p}_{dylog} \Rightarrow_* \sigma$ where each derivation step is determined by selecting the production rule that is obtained from the inclusion axiom of the the corresponding rule (1) and (4) that has been applied. This shows that $\sigma \in L(G_{p_{dylog}})$. The second part of the proposition trivially derives from definitions.

Proposition 4. *Protocol conformance is decidable.*

Proof. Equation (8) is equivalent to $L(G_{\mathsf{p}_{dylog}}) \cap \overline{L(G_{\mathsf{p}_{AUML}})} = \emptyset$. Now, $L(G_{\mathsf{p}_{dylog}})$ is a CFL 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 CFL and a regular language is a CFL. For CFLs, the emptyness is decidable [20].

Proposition 4 tells us that an algorithm for verifying protocol conformance exists. In [13, 8] a procedure to verify the containment property of a CFL in a regular language is given, that takes $O(|P_{G_{p_{dylog}}}| \cdot |V_{G_{p_{AUML}}}|^3)$ time and $O(|P_{G_{p_{dylog}}}| \cdot |V_{G_{p_{AUML}}}|^2)$ space.

Example 2. Let us consider the $\mathsf{yes\_no\_query}_I$ DyLOG procedure, presented in Section 2.2, clauses (g)-(j). In the case in which *Fluent* holds $available(movie)$, Algorithm 2 produces the following grammar $G_{\mathsf{yes\_no\_query}_{I\,dylog}}$:

$\mathsf{yes\_no\_query}_I(cus, sp, available(movie)) \longrightarrow$
  $\mathsf{get\_start}(cus, sp, available(movie))\ \mathsf{refuseInform}(cus, sp, available(movie))$
$\mathsf{yes\_no\_query}_I(cus, sp, available(movie)) \longrightarrow$
  $\mathsf{get\_start}(cus, sp, available(movie))\ \mathsf{inform}(cus, sp, available(movie))$
$\mathsf{yes\_no\_query}_I(cus, sp, available(movie)) \longrightarrow$
  $\mathsf{get\_start}(cus, sp, available(movie))\ \mathsf{inform}(cus, sp, \neg available(movie))$
$\mathsf{get\_start}(cus, sp, available(movie)) \longrightarrow$
  $\mathsf{queryIf}(cus, sp, available(movie))$

It is easy to see that the language produced by it is the following and that it contains three possible conversations:

$$L(G_{\mathsf{yes\_no\_query}_{I\,dylog}}) = \{$$
$$\mathsf{queryIf}(cus, sp, available(movie))\mathsf{refuseInform}(cus, sp, available(movie)),$$
$$\mathsf{queryIf}(cus, sp, available(movie))\mathsf{inform}(cus, sp, available(movie)),$$
$$\mathsf{queryIf}(cus, sp, available(movie))\mathsf{inform}(cus, sp, \neg available(movie))\ \}$$

The grammar $G_{\mathsf{yes\_no\_query}_{I\,AUML}}$, obtained starting from the AUML specification of the protocol, is similar to the one shown in Fig. 1, productions from Q_1 through Q_7, where Q_7 produces ε instead of Q_8. The language $L(G_{\mathsf{yes\_no\_query}_{I\,AUML}})$ contains the same conversations of $L(G_{\mathsf{yes\_no\_query}_{I\,dylog}})$, therefore the *protocol conformance* trivially holds. This is a structural conformance, in the sense that no information about the agent private state is taken into account nor the semantics of the speech acts is.

Now, the speech acts might have different semantics (different preconditions or effects); for instance, we can imagine two $inform$ implementations, the first can be executed when the informer knows a certain fact, the other when it knows a fact and it believes that the querier does not know it. Depending on its semantics, an $inform$ act might or might not be executed in a given agent mental state. Thus, generally, the interaction dynamics of the speech act semantics and the agent belief states might enable or disable conversations even when using the same agent policy. Nervertheless, since protocol conformance holds, by Proposition 3 we can state that the obtained conversations will *always* be legal w.r.t. the AUML specification; the private information of the agent and the semantics of the speech acts will, at most, reduce the set of possible conversations but they will never introduce new, uncorrect sequences.

4 Conclusions and Related Work

In this work we face the problem of verifying if the implementation of an interaction protocol as an internal policy of a logic-based agent is conformant to the protocol abstract specification, in the special case of DyLOG agents implementing AUML specifications. We have taken advantage from the logical representation of protocols in DyLOG as inclusion axioms, by interpreting the conformance problem as a problem of language inclusion.

Verifying the conformance of protocol implementations is a crucial problem in an AOSE perspective, that can be considered as a part of the process of engineering interaction protocols sketched in [21]. In this perspective the techniques discussed along our paper, actually, suggest a straightforward *methodology* for directly implementing protocols in DyLOG so that conformance to 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 can 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.

The problem of checking the agent conformance to a protocol in a logical framework has been faced also in [12]. In [12] 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 disregarding any condition related to the agent private knowledge, which is not considered as relevant in order to decide weak conformance. On this respect, our notion of conformance is similar to the notion of agent weak conformance described above. However, our approach allows to tackle a broader class of protocols: we are not restricted to protocols that sequentially alternate the dialogue moves of the agents. Furthermore, while in [12] conformance avoids to deal with the dialogue history, our notion of conformance takes into account the whole context of the conversation, due to the fact that it considers sequences of dialogue acts. This can be done thanks to the modal logic framework, which allows to naturally deal with contexts.

Moreover, 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. This allows us to decide 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, whose ability of reasoning about the properties of the interactions among agents before they actually occur, may be a powerful tool for supporting MAS designers.

So far we have focussed on the conformance of the policies of a single agent to a protocol specification. A current research issue that we are studying concerns the conditions by which our notion of conformance can be proved compositional. Intuitively, given two agent policies that are conformant to the same protocol and that encode the different roles foreseen by it, it would be interesting to prove that the actual interaction of the two agents will also be conformant.

Some authors (e.g. [29]) have proposed a different approach to agent communication, the *social* approach, in which communicative actions affect the "social state" of the system, rather than the internal states of the agents. The social state records the social facts, like the *permissions* and the *commitments* of the agents, which are created and modified along the interaction. Different approaches enable different types of properties to be proved [19]. For instance the mental approach is not well suited for the verification of *open* multi-agent systems, where the history of communications is observable, but the internal states of the single agents may not be accessed [29]. Therefore the social approach is taken in works such as the one in [1], where an open society of agents is considered and the prob-

lem of verifying on the fly the compliance of the agents' behavior to protocols specified in a logic-based formalism (Social Integrity Constraints) is addressed by taking the point of view of an external entity that detects faulty behaviors.

Acknowledgement. The authors would like to thank Dr. Giuseppe Berio for the discussion about Agent UML.

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 H. Haddad, A. Omicini, R. L. Wainwright, and L. M. Liebrock, editors, *Proc. of the 2004 ACM Symposium on Applied Computing, SAC 2004*, pages 72–78, Nicosia, Cyprus, 2004. ACM.
2. 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.
3. 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.
4. 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.
5. M. Baldoni, L. Giordano, A. Martelli, and V. Patti. Programming Rational Agents in a Modal Action Logic. *Annals of Mathematics and Artificial Intelligence, Special issue on Logic-Based Agent Implementation*, 41(2-4):207–257, 2004.
6. M. Barbuceanu and M.S. Fox. Cool: a language for describing coordination in multiagent systems. In *the 1st Int. Conf. on Multi-Agent Systems (ICMAS-95)*. AAAI Press, 1995.
7. J. Bentahar, B. Moulin, J. J. Ch. Meyer, and B. Chaib-Draa. A computational model for conversation policies for agent communication. In this volume.
8. A. Bouajjani, J. Esparza, A. Finkel, O. Maler, P. Rossmanith, B. Willems, and P. Wolper. An efficient automata approach to some problems on context-free grammars. *Information Processing Letters*, 74(5–6):221–227, 2000.
9. A. Bracciali, P. Mancarella, K. Stathis, and F. Toni. On modelling declaratively multiagent systems. In Leite et al. [25], pages 76–92.
10. L. Cabac and D. Moldt. Formal semantics for auml agent interaction protocol diagrams. In *Proc. of AOSE 2004*, 2004.
11. R. S. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng. Modeling agent conversation with colored petri nets. In *Autonomous Agents Workshop on Conversation Policies*, 1999.
12. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Logic-based agent communication protocols. In F. Dignum, editor, *Advances in agent communication languages*, volume 2922 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 91–107. Springer-Verlag, 2004.

13. J. Esparza, P. Rossmanith, and S. Schwoon. A uniform framework for problems on context-free grammars. *EATCS Bulletin*, 72:169–177, October 2000.
14. L. Fariñas del Cerro and M. Penttonen. Grammar Logics. *Logique et Analyse*, 121-122:123–134, 1988.
15. M. Finger, M. Fisher, and R. Owens. Metatem: modeling reactive systems using executable temporal logic. In *the Int. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA-AIE)*, 1993.
16. M. Fisher and M.J. Wooldridge. Specifying and executing protocols for cooperative actions. In *the Int. Working Conf. on Cooperative Knowledge-Based Systems (CKBS-94)*, 1994.
17. Foundation for InteroPerable Agents. Fipa modeling: Interaction diagrams. Technical report, 2003. Working Draft Version 2003-07-02.
18. L. Giordano, A. Martelli, and C. 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.
19. 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-Verlag, 2003.
20. J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley Publishing Company, 1979.
21. 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-Verlag, 2003.
22. M. Kacprzak, A. Lomuscio, and W. Penczek. Verification of multiagent systems via unbounded model checking. In *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS04)*, New York, NY, USA, 2004.
23. J.-L. Koning, G. Franois, and Y. Demazeau. Formalization and pre-validation for interaction protocols in multiagent systems. In *the 13th European Conference on Artificial Intelligence (ECAI-98)*, 1998.
24. K. Kuwabara, T. Ishida, and N. Osato. Agentalk : describing multiagent coordination protocols with inheritance. In *7th Int. Conf. on Tools for Artificial Intelligence (ICTAI-95)*, 1995.
25. J. Leite, A. Omicini, P. Torroni, and P. Yolum, editors. *Int. Workshop on Declarative Agent Languages and Technology*, New York City, NY, USA, July 2004. Volume 3476 of *LNAI*. Springer-Verlag, 2005.
26. H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. GOLOG: A Logic Programming Language for Dynamic Domains. *J. of Logic Programming*, 31:59–83, 1997.
27. 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.
28. L. R. Pokorny and C. R. Ramakrishnan. Modeling and verification of distributed autonomous agents using logic programming. In Leite et al. [25], pages 172–187.
29. M. P. Singh. A social semantics for agent communication languages. In *Proc. of IJCAI-98 Workshop on Agent Communication Languages*, Berlin, 2000. Springer.
30. C. Walton. Model checking agent dialogues. In Leite et al. [25], pages 156–171.