Defining Interaction Protocols using a Commitment-based Agent Communication Language*

Nicoletta Fornara
Università della Svizzera italiana
via Buffi 13, 6900 Lugano, Switzerland
nicoletta.fornara@lu.unisi.ch

Marco Colombetti
Università della Svizzera italiana
via Buffi 13, 6900 Lugano, Switzerland
Politecnico di Milano
Piazza L. da Vinci 32, I-20133 Milano, Italy
marco.colombetti@lu.unisi.ch

ABSTRACT

We propose a method for the definition of *interaction protocols* to be used in open multiagent systems. Starting from the assumption that *language* is the fundamental component of every interaction, we first propose a semantics for Agent Communication Languages based on the notion of *social commitment*, and then use it to define the meaning of a set of basic communicative acts. Second, we propose a verifiable and application-independent method for the definition of interaction protocols, whose main component is the specification of an *interaction diagram* specifying which actions may be performed by agents under given conditions. Interaction protocols fully rely on the application-independent meaning of communicative acts. We also propose a set of *soundness conditions* that can be used to verify whether a protocol is reasonable. Finally, our approach is exemplified by the definition of an interaction protocol for English auctions.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Languages and structures, Multiagent systems*

General Terms

Design, Standardization, Languages, Theory

Keywords

agent communication language, commitment, communicative act, interaction protocol, open interaction system, auction

*"STUDENT PAPER"

AAMAS'03, July 14–18, 2003, Melbourne, Australia.

1. INTRODUCTION

Interaction Protocols are patterns of behavior that agents have to follow to engage in a communicative interaction with other agents within a multiagent system (MAS). The specification of interaction protocols is crucial for the development of a MAS: the advent of Internet makes it urgent to develop general, application-independent methods for the definition of interaction protocols, to be used as components of open, dynamic, heterogeneous, and distributed interaction frameworks for artificial agents. Indeed, the definition of new interaction protocols is a critical task, because a badly designed protocol may lead to unsuccessful interactions; thus there is a need for general methods, criteria, and tools for protocol design. We think that there are some important properties that interaction protocols for open frameworks have to satisfy. In particular, an interaction protocol should:

- Specify legal sequences of communicative acts that form a complete interaction within a system. Every communicative act used in a protocol should maintain its meaning, as defined in a general, application-independent communicative act library of a standard Agent Communication Language (ACL).
- Enable interactions among purely reactive agents, that blindly follow a given protocol, and deliberative agents, that are able to reason about the consequences of actions, and decide whether to take or not to take part in an interaction.
- Allow for effective verification that agents behave in accordance to the specifications of the interaction protocol.

Moreover, a general method for the development of interaction protocols should allow a designer to verify whether a protocol is "sound" with respect to general, application-independent soundness criteria.

So far, several approaches to the definition of interaction protocols have been proposed. Some authors define interaction protocols as finite state machines or Petri nets (see for example [3] and [6]), but do not take into account the meaning of the exchanged messages, which in our opinion is crucial to obtain the properties listed above. Other approaches take into account the meaning of the exchanged messages, but do not rely on a standard ACL with application-independent semantics; for instance Esteva *et al.* [4] specify the protocols

available in an electronic institution using finite state machines, but define the meaning of only some of the message types using *ad-hoc* rules. An example of interaction protocol specification which fully takes into account the meaning of the exchanged messages is proposed by Yolum and Singh [10], who introduce a method based on event calculus to define protocols that may be used by artificial agents to determine flexible paths of interaction complying with the specifications. The main difference between Yolum and Singh's proposal and the one put forward in this paper is that with the method described in this paper all the preconditions and effects of the performance of communicative acts on the state of the interaction are completely specified; we also propose a method through which protocol designers may verify if a protocol is sound with respect to a number of general, application-independent soundness criteria related also to the meaning of the exchanged messages.

Our approach to agent interaction presupposes the definition of a standard ACL with unambiguous semantics. In a previous paper [5] we have shown how the semantics of an ACL can be defined in terms of (social) commitments. Our definitions set the rules for the execution of communicative acts, which are regarded as commitment-manipulation actions. Starting from such an analysis, which will be briefly summarized in Section 2, we show how an interaction protocol can be defined. It is important to remark that our protocols are defined starting from the communicative act library of a predefined ACL, and that all communicative acts preserve their general meaning when used within a protocol. As we shall see, an interaction protocol mainly consists of a set of rules that regulate the performance of certain communicative acts; part of these rules are expressed in terms of an *interaction diagram* that specifies which actions can be performed by the agents at every stage of the interaction. Of course, an arbitrary collection of rules does not necessarily define a reasonable interaction protocol. We therefore propose a set of application-independent and verifiable *soundness conditions*, which guarantee that protocols possess certain properties that are crucial for a successful interaction. Such conditions are expressed in terms of the content of the system state at each stage of the interaction, as consequence of the performance of communicative acts.

The paper is organized as follows. Section 2 introduces a commitment-based framework for the definition of an ACL, and a minimal communicative act library that we consider essential to describe communicative interactions in an open MAS. In Section 3 we define a general method for the definition of interaction protocols, and introduce a set of soundness conditions, related to the meaning of the messages exchanged by the agents, which may be used to validate interaction protocols. In Section 4 we present a specification of a form of English auction, an interaction protocol widely used in electronic commerce applications, based on the formalism presented in the paper. Finally, in Section 5 we draw some conclusions.

2. A COMMITMENT-BASED AGENT COMMUNICATION LANGUAGE

A complete operational specification of a commitment-based ACL and a discussion of its motivations can be found in [5]. The semantics proposed in that paper is given by describing the effects that sending a message has on the social relationship between the sender and the receiver of the message using an unambiguous, objective, and public concept, that is, *social commitment*. We assume that the open system in which artificial agents interact consists of the following components:

- A group of registered agents $\{a, b, ...\}$.
- A variable set of *commitment objects* $\{C_1, C_2, ...\}$, which are instances of the commitment class discussed below.
- A variable set of *temporal proposition objects* $\{P, Q, ...\}$, which are instances of the corresponding class discussed below, and are used to express propositions about the application domain and the interaction process.
- A fixed set of actions that agents may perform, including both communicative acts belonging to a communicative act library and application domain actions.
- A fixed set of event-driven routines that automatically update the state of commitment objects. These routines are represented by *update rules* as described in Table 1.
- A set of domain-specific objects $\{O_1, O_2, ...\}$, which represent entities of the application world. Such entities may possess both "natural" or and "institutional" attributes; for example, the color of a product being sold is a natural attribute, while the price of the same product is an institutional attribute. Natural attributes are assumed to reflect the physical properties of the corresponding entities of the real world, and typically cannot be changed during an interaction (of course, they might be changed if some of the interacting agents were assumed to be physical robots). On the contrary, institutional attributes can be affected by the performance of certain communicative acts, in particular by declarations (as discussed below). We assume that each domain-specific object has a value-setting method for each of its institutional properties; for example the method "setState()" can be invoked to set the "state" property.
- A fixed set of *roles* $\{role_1, role_2, ...\}$. This concept is introduced to abstract from the specific agents that take part in an interaction.
- A fixed set of *authorizations* associated to roles, that specify which agent is authorized to perform a particular declaration (see Section 2.1 for details).

Commitment objects are used to represent the network of commitments binding the interacting agents; they have an internal structure, a life cycle, and a set of methods available for manipulation. The internal structure of a commitment object consists of the following fields:

- a unique commitment *identifier* (id);
- a reference to the commitment's *debtor*, that is, the agent that has the commitment;
- a reference to the *creditor*, that is, the agent relative to which the debtor is committed;

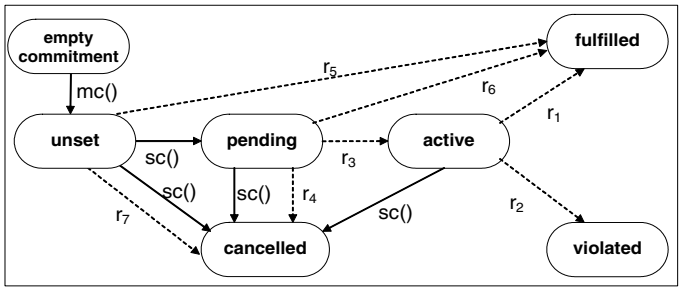

Figure 1: The life-cycle of commitments.

- the commitment's *content*, that is, the representation of the proposition (describing a state of affairs or a course of action) to which the debtor is committed relative to the creditor;
- the commitment's *conditions*, that is, a list of propositions that have to be satisfied in order for the commitment to become active;
- a *state*, taken from the finite set $\{unset, cancelled, pending, active, fulfilled, violated\}$, used to keep track of the dynamic evolution of the commitment; and
- a *timeout*, which is relevant only in the case of unset commitments, and will therefore be treated as an optional parameter.

Commitment objects will be represented with the following notation:

$$C_{id}(state, debtor, creditor, content|conditions\{, timeout\}).$$

We use temporal proposition objects to represent the content and the conditions of a commitment. A temporal proposition object consists of the following fields: a *statement*, which represents a proposition; the *truth_value* of the statement, which may be true (1), false (0) or undefined ($\perp$); a *time interval*, which may go from a single instant to the entire life of the system, relative to which the statement is considered; and a temporal *mode*, either ($\forall$) or ($\exists$), which specifies whether the statement should be true for the whole time interval or on at least an instant of the time interval. We assume that the truth value of temporal proposition objects is updated by a suitable "notifier". In particular: if the mode is '$\exists$' the notifier sets the truth-value to true if the statement becomes true at any point of the time interval, otherwise sets it to false when the time interval expires; if the mode is '$\forall$' the notifier sets the truth-value to false if the statement becomes false at any point of the time interval, otherwise sets it to true when the time interval expires. It is important to remark that the truth value of a temporal proposition object can switch from $\perp$ to 1 or 0, but then cannot change any more. In particular cases, as we shall see, it is possible to infer in advance that the statement of a temporal proposition object can no longer become true (false) within the associated time interval. In this case the notifier may set the truth value to false (true) before the time interval expires. To do so, the notifier may exploit specific inference rules (more on this later). Temporal proposition objects are represented with the following notation:

$$P_{identifier}(statement, time\ interval, mode, truth\_value).$$

As we have already said, temporal proposition objects are used to represent content and conditions of a commitment. In particular the conditions of a commitment consist of a list $[P, Q, ...]$ of temporal proposition objects that have to be satisfied in order for the commitment to become active. The truth value of a list of temporal proposition objects is computed as follows: (i) an empty list of temporal proposition objects is true; (ii) a true temporal proposition object is removed from the list; (iii) a list containing a false proposition object is false.

To make the notation simpler, when the list of conditions contains one temporal proposition object the square brackets are dropped. We also remark that a temporal proposition object, used to express the content or a condition of a commitment object, may in turn represent another commitment object. In particular temporal proposition objects can be used to represent conditions on the temporal evolution of commitments. An example of this is given in Subsection 2.1.4.

The life cycle of a commitment object is described by the finite state machine in Figure 1. The state of a commitment can change as an effect of the invocation of its basic methods (solid lines) or of environmental events (dotted lines), that is, of events that change the truth-value of a commitment's conditions or content. We assume that when a commitment object is declared, the constructor of the class creates an empty commitment object, $C_i()$. We represent the invocation of a method by the name of the object followed by a dot and by the name of the method with its parameter list. Commitments are created and manipulated through the following basic operations:

- *Make commitment.* By invoking the method $mc(a, b, P, Q)$ with arbitrary debtor a, creditor b, content P, and condition list Q, a new unset commitment object is created:

$$C_i().mc(a, b, P, Q) \rightarrow C_i(unset, a, b, P|Q)$$

- *Set commitment.* The method $sc(s)$ changes the current state of an existing commitment object to s:

$$C_i(-, a, b, P|Q).sc(s) \rightarrow C_i(s, a, b, P|Q)$$

- *Add condition.* The method $ac(R)$ adds a new temporal proposition object R to the conditions of the commitment:

$$C_i(s, a, b, P|Q).ac(R) \rightarrow C_i(s, a, b, P|R \bullet Q),$$

  where the symbol $\bullet$ denotes the operation of inserting a new element in a list.

Basic operations should not be viewed as actions that are directly performed by agents. Rather, they are low-level primitives used to implement operations on commitment objects, more specifically, agents manipulate commitments through a communicative act library.

Finally note that we defined the conditions under which commitments are fulfilled or violated, but we are not concerned with the management of violations e.g. in terms of sanctions, because this aspect lies beyond the use of commitments for the definition of ACL semantics.

Table 1: Update Rules.

event	action	rule
$P.truthval() = 1$	$C_i(a,a,b,P\|T) \rightarrow C_i(f,a,b,P\|T)$	1
$P.truthval() = 0$	$C_i(a,a,b,P\|T) \rightarrow C_i(v,a,b,P\|T)$	2
$Q.truthval() = 1$	$C_i(p,a,b,P\|Q) \rightarrow C_i(a,a,b,P\|T)$	3
$Q.truthval() = 0$	$C_i(p,a,b,P\|Q) \rightarrow C_i(c,a,b,P\|Q)$	4
$P.truthval() = 1$	$C_i(u,a,b,P\|T) \rightarrow C_i(a,a,b,P\|T)$	5
$P.truthval() = 1$	$C_i(p,a,b,P\|Q) \rightarrow C_i(f,a,b,P\|Q)$	6
$curr\_time > t$	$C_i(u,a,b,P\|Q,t) \rightarrow C_i(c,a,b,P\|Q)$	7
legend: $u = unset$, $p = pending$, $c = cancelled$, $a = active$, $f = fullfilled$, $v = violated$, $T = TRUE$		

2.1 Library of Communicative Acts

We shall now define the meaning of the basic types of communicative acts as identified by Speech Act Theory [8]. We extend the definitions of [5] by introducing the definition of a new commissive act, the *conditional accept* act, and a treatment of *declarations*; both will be used in the example of Section 4. In the following definitions the sign "$=_{def}$" means that performing the action represented on the left-hand side is the same as performing the action represented on the right-hand side, and the symbol ":=" means that the act represented on the left-hand side is actually performed through the invocation of the methods listed on the right-hand side.

2.1.1 Assertives

According to Speech Act Theory, the point of an assertive act is to commit the sender, relative to the receiver, to the truth of what is asserted. We consider the **inform** act as our prototypical assertive act. This act is used by agent a to inform agent b that P is the case. In a commitment-based approach, an act of informing can be defined as follows ($TRUE$ is the identically true temporal propositional object):

$$\begin{aligned}&\mathbf{inform}(a,b,P) :=\\ &\{C_i().mc(a,b,P,TRUE);\\ &C_i(unset,a,b,P|TRUE).sc(pending)\}.\end{aligned}$$

The final result is an active commitment, thanks to the intervention of Update Rule 3.

2.1.2 Directives

As defined in Speech Act theory, the point of a directive act is to get the receiver to perform an action (possibly a speech act). We treat **request** as our basic directive act, and define it as the creation of an unset commitment with the sender as the creditor and the receiver as the debtor. The request by agent a to agent b to bring about P if condition list Q is satisfied is defined as:

$$\mathbf{request}(a,b,P,Q) := \{C_i().mc(b,a,P,Q)\}.$$

The receiver of a request can react in three different ways: it can perform the requested action, accept the request, or refuse it. Questions (or queries) are requests to be informed about something. Here we deal with only wh-questions; for a definition of yes-no-questions see [5]. In wh-questions the requested act of informing cannot be completely described by the sender (otherwise, why should it ask the question?). In this cases the sender provides a "template" for the answer, that is, a temporal proposition object $S(\mathbf{x})$ containing a meta-variable $\mathbf{x}$ that the receiver has to replace with a constant value c. A query has therefore the form:

$$\begin{aligned}&\mathbf{request}(a,b,P)\\ &\quad where\ P.statement() = \mathbf{inform}(b,a,S(\mathbf{x}))\\ &\mathbf{inform}(b,a,S(\mathbf{x})) =_{def} \mathbf{inform}(b,a,S(c)),\\ &\quad for\ some\ constant\ value\ c.\end{aligned}$$

This definition implies that the performance of the requested inform act with the temporal proposition $S(c)$ as a parameter makes the temporal proposition P true. Indeed, as remarked by Searle [9] the concept of a question is more general: by a question, an agent may request the execution of a non-assertive communicative act (like a directive, or a commissive). However, our definition above easily generalizes to such cases (an example can be found in Section 4).

2.1.3 Commissives

The point of a commissive act, as defined by Speech Act theory, is to commit the debtor, relative to the creditor, to the execution of an action of a given type. Here we define the basic commissive act of *promising*:

$$\begin{aligned}&\mathbf{promise}(a,b,P,Q) :=\\ &\{C_i().mc(a,b,P,Q);\ C_i(unset,a,b,P|Q).sc(pending)\}\end{aligned}$$

To make an unconditional promise the constant proposition object $TRUE$ is used as the condition, and thus the pending commitment created by the promise is immediately turned into an active commitment by Update Rule 3. Three types of commissive acts can be performed only in connection with an unset commitment, namely **accept**, **conditional accept** and **reject**. Accepting and rejecting are defined as follows:

$$\begin{aligned}&preconditions:\ \exists\, C_i(unset,b,a,P|Q))\\ &\mathbf{accept}(b,a,C_i(unset,b,a,P|Q)) :=\\ &\{C_i(unset,b,a,P|Q).sc(pending)\}\end{aligned}$$

$$\begin{aligned}&preconditions:\ \exists\, C_i(unset,b,a,P|Q))\\ &\mathbf{reject}(b,a,C_i(unset,b,a,P|Q)) :=\\ &\{C_i(unset,b,a,P|Q).sc(cancelled)\}\end{aligned}$$

Another useful commissive act is "conditional accept", which may be used by agents to negotiate the condition of an unset commitment. In particular, conditional acceptance will appear in the example proposed in Section 4. In fact, in the English Auction Protocol at every round of the bidding process the auctioneer accepts the currently highest bid on condition that no higher bids will be accepted later. In general, the debtor of an unset conditional commitment C_i can accept it provided that an additional condition, represented by a temporal proposition object, holds. Conditional acceptance transforms an unset commitment into a pending commitment, and adds a new condition to the original condition list of the unset commitment:

$$\begin{aligned}&preconditions:\ \exists\, C_i(unset,b,a,P|Q))\\ &\mathbf{condAccept}(b,a,C_i(unset,b,a,P|Q),R) :=\\ &\{C_i(unset,b,a,P|Q).ac(R);\\ &C_i(unset,b,a,P|R \bullet Q).sc(pending)\}\end{aligned}$$

Note that when condition R becomes true, the debtor is left with a pending conditional commitment of the form $C_i(pending,b,a,P|Q)$.

2.1.4 Proposals

A proposal is a combination of a directive and a commissive act. Even if proposals are not basic acts, they deserve special attention because they are crucial in many interesting application fields, like for example electronic commerce. A **propose** act can be defined as the parallel execution of a request and a promise, as denoted by the symbol ||:

$$\textbf{propose}(a, b, P, Q) =_{def}$$
$$\textbf{request}(a, b, P, Q) \;||\; \textbf{promise}(a, b, Q, S)$$
$$where\ S.statement() = C_i(pending, b, a, P|Q)$$

Note that in the above definition the statement of temporal object S represents the commitment object $C_i(pending, b, a, P|Q)$.

2.1.5 Declarations

Declarations are a special type of communicative acts. Examples of declarations are "I pronounce you man and wife" or "I declare the auction open". The point of a declaration is to bring about a change in the world, obviously not in the physical or natural world but in an institutional world [1], that is, a conventional world relying on common agreement of the interacting agents (or, more precisely, of their designers). Declarations actually change the institutional world simply in virtue of their successful performance. In our interaction framework, to treat declarations we introduce objects with *institutional properties*, that is, conventional properties that result from common agreement, like for example the ownership of a product. Such properties can be affected by declaration acts. It is however necessary to identify which agents are *authorized* or *empowered* to perform a given declaration act in the system. Typically, authorizations are granted to agents in virtue of the *role* they play in an interaction, and thus authorizations are naturally associated to roles. To do so, we need to introduce a construct to express that an agent having a given role in the interaction system is empowered to bring about an institutional change of a given kind:

$$preconditions:$$
$$empowered(role_i, O_k.setProp_j()) \wedge a.role() = role_i$$
$$\textbf{declare}(a, O_k.prop_j = x) := \{O_k.setProp_j(x)\}.$$

3. INTERACTION PROTOCOLS

Having defined an essential Communicative Act Library we can now proceed to the specification of interaction protocols. An interaction protocol is defined by an *environment* and an *interaction diagram*. In particular, a protocol's environment defines:

- A nonempty set of *roles* that agents can play in the interaction. To each role, a set of specific authorizations may be associated.
- A nonempty set of *participants*, which are the agents interacting by using the protocol. Every participant must play a well-defined role in the interaction. The set of participants may vary during the execution of the protocol, but is always finite.
- A possibly empty set of global *constants* and *variables*, that may be subject to global *constraints*.
- A collection of commitment objects, temporal proposition objects, and domain-specific objects (see Section 2) used to represent all entities involved in the interaction.
- A set of *authorizations*, associated to roles, to perform certain institutional actions, in particular declarations.

A protocol's interaction diagram specifies which actions may be performed by each agent at each stage of the interaction. More precisely, an interaction diagram (see for example Figure 2) is defined by a finite graph in which:

- Every node represents a state of the interaction. To every state we can associate a representational content, that is, the set of all facts that hold at the state, expressed in terms of: protocol variable values, commitment objects, temporal proposition objects, and domain-specific objects.
- There is a single distinguished *initial node*, with no incoming edge, and a set of distinguished *final nodes*, with no outgoing edge. The interaction starts from the initial node and ends when a final node is reached.
- Every edge describes a transition from a state to another state. A transition may correspond to the execution of a communicative act or to the occurrence of a relevant environmental event; when the transition occurs, the content of the target state can be completely computed from the content of the source state, and from the semantics of the communicative act or a description of the environmental event responsible for the transition.
- When more than one communicative-act edge goes out of a given node, it is possible to specify the conditions (defined as arbitrary Boolean expressions) under which each act may be executed. As a whole, the set of condition-action pairs going out of a node behaves like a *guarded command* [2]: at least one of the actions must be executed, but the agent specified as the actor of the action is free to choose which action to perform among those whose *guard* is true. If all guards are mutually exclusive, the guarded command is equivalent to a sequence of if-then statements.
- It is possible to associate a cardinality to communicative act edges. In particular cardinality "1 to n" means that the same message is sent by one agent to n agents, and cardinality "1 to 1" means that a message is send by one agent to another agent.

3.1 Soundness conditions

To describe a sensible interaction pattern, an interaction protocol must satisfy a number of general, application-independent *soundness conditions*. A first, fairly trivial, set of conditions concerns the topology of the interaction diagram:

- Every node of the interaction diagram must be reachable from the initial node.
- There must be at least a final node.

Another, less trivial, set of soundness conditions concerns the content of states. Such conditions, express constraints related to the meaning of the exchanged messages, as defined by the communicative act library adopted.

- All communicative acts that are allowed by a protocol at state s must have their preconditions satisfied by the content associated to s when their guard is true. This condition guarantees that all communicative acts allowed by the interaction protocol may actually be executed.

- All commitments included in the content of a final state must be cancelled, fulfilled, or violated. This condition guarantees that the whole interaction has been completed.

An interesting problem is raised by the fact that during the execution of a protocol, the same state may be reached from the start state following different paths (i.e., performing different chains of actions). For example, a certain state of an interaction could be reached because an agent has autonomously made a promise or because the agent was requested to make a promise, accepted the request, and then fulfilled the resulting commitment by actually making the promise. If we abstract from the different paths, we intuitively feel that the interaction has reached the same state; however, if we compute the content of the state we get different results. The point is that these results, although different, are equivalent from the point of view of the interaction, in that they have the same "commissive import". More precisely, we say that state s is *equivalent* to state s' if and only if the contents of s and s' are identical, with the only exception of commitments that are fulfilled, violated, or cancelled. We can therefore formulate another soundness condition:

- If a state of an interaction can be reached through different paths, the contents of the state computed along the different paths must be equivalent.

The situation is even more complex when the definition of an interaction protocol has a loop, that is, a cycle in the interaction diagram. Interaction loops naturally appear when a sequence of communication acts can be repeated several times, like for example in the English Auction Protocol (Section 4). The existence of loops makes it possible to reach the same state following different paths in the interaction diagram. In this case, however, the notion of equivalence discussed above is still necessary but no longer sufficient. This problem is well known in computer programming, and it can be solved by introducing the concept of a *loop invariant*. For example, consider again the protocol for an English Auction. At a generic iteration, the auctioneer is committed to selling the product currently under the hammer to a specific agent for a specific price, on condition that no higher price will be offered. Of course, the specific agent that made the highest offer, as well as the associated price, will change from one iteration to another one. However, we can describe the situation in terms of loop invariants, by saying that the auctioneer is committed to selling the product to *the agent that made the highest offer*, for *the price defined by such an offer*, on condition that no higher offer will be made. The soundness condition given above can now be reformulated as follows:

- If a state of an interaction can be reached through different paths, the contents of the state computed along the different paths, expressed in terms of suitable invariants, must be equivalent.

4. THE ENGLISH AUCTION PROTOCOL

In this section we present a specification of a form of English auction protocol using the framework proposed so far. We chose this protocol as an example because it is used in many electronic commerce applications on the web, and because it is fairly complex: in particular, it is an interesting example of iterative interaction protocol. In this example we consider the interaction process needed to sell a single product o, which can obviously be repeated to sell several products.

4.1 The Environment

The environment of the English Auction Protocol includes the following elements:

- **Roles**. *Auctioneer* and *Client*.

- **Participants**. One agent, a, in the role of *Auctioneer* and n agents, $\{p_1, ..., p_n\}$, in the role of *Client*.

- **Constants and Constraints**. t_{max}, the maximum duration of the auction; t_1, the deadline for the payment; t_2, the deadline for the delivery. $t_{max} < t_1 < t_2$.

- **Domain-specific objects and actions**.
  Object o, representing the product on sale, with a $resPrice$ field for the reservation price.
  Object A, representing the auction, with fields for the following variables (initialized as indicated): $state$ = "*closed*"; $askPrice = 0$; t_{end} automatically set to t_{system} when $state$ is set to "closed"; $t_{inactivity}$ i.e. the maximum time of inactivity between two subsequent bids.
  The action of transferring the ownership of an object or of a sum of money to another agent is an institutional action involving the institutional notions of ownership and money. For the sake of simplicity, we treat here this action as a primitive domain action. The fact that agent a transfers to agent b the ownership of x (an object or a sum of money) is represented by $give(a, b, x)$.

- **Variables** The environment has the following variables (initialized as indicated): $newAskPrice = o.resPrice()$; $value_{win} = 0$; t_{system}, a global clock accessible to all participants; $t_{bid} = 0$, the time of the last accepted bid; i a counter that is automatically incremented every time the bidding process is iterated.

- **Authorizations**. The auctioneer is empowered to open and close the auction and to set the ask price of the product on sale:
  $empowered(Auctioneer, A.setState())$,
  $empowered(Auctioneer, A.setAskPrice())$.

4.1.1 Scheme of Temporal Proposition Objects

In the interaction framework proposed so far the *content language* used to express the content and the condition fields of commitment objects is based on the use of temporal proposition objects. Given that in complex interactions like the ones that follow the English Auction Protocol many temporal proposition objects are involved, we concisely describe them through *schemes*, that represent possible temporal proposition object in parametric form. Parameters will be bound to specific values when the interaction actually takes place and an instance of the temporal proposition

object is created (truth values are always initialized to $\perp$, and therefore are not indicated in schemes). In our example, parameter *now* is initialized at t_{system} when the temporal proposition object is created; and parameter v and v' are used to indicate an amount of money.

- Scheme P_j represents the proposition "the auctioneer gives product o to client p_j, in the time interval from the end of the auction to t_2":
  $P_j(give(a, p_j, o), t_{end}...t_2, \exists)$;
- Scheme $Q_{j,v}$ represents the proposition "client p_j gives the amount v of money to the auction house represented by the auctioneer, in the time interval from the end of the auction to t_1": $Q_{j,v}(give(p_j, a, v), t_{end}...t_1, \exists)$;
- Scheme $S_{j,i}$ represents the proposition "client p_j makes a proposal during iteration i":
  $S_{j,i}(propose(p_j, a, P_j, Q_j(\mathbf{x})), now...now+t_{inactivity}, \exists)$;
- Scheme $U_{j,v}$ represents the proposition "the auctioneer is committed, relative to client p_j, to proposition P_j under condition $Q_{j,v}$, in the time interval from now to the end of the auction":
  $U_{j,v}(C_{id2j}(pending, a, p_j, P_j|Q_{j,v}), now...t_{end}, \exists)$;
- Scheme $W_{v'}$ represents the proposition "the auctioneer does not accept any proposal with value greater than v' in the time interval from now to the end of the auction":
  $W_{v'}(\neg\ \exists j\ (condAccept(a, p_j, C_{id2j}(unset, a, p_j, P_j|Q_{j,v}), W_v) \wedge v > v')), now...t_{end}, \exists)$;

4.1.2 Communicative Acts

In this section specific conditions (guards) for the performance of the communicative acts used in the English Auction Protocol are given. Obviously in order that the communicative act is successfully performed also the preconditions defined in the Library of Communicative Acts have to be satisfied. Some of these acts have to be repeated at every round i of the bidding process. In order to be able to refer to commitment objects created by previously preformed communicative acts we report also the effects of the performance of communicative acts: they can also be computed from the definitions given in the library of communicative acts. Therefore in the protocol communicative acts retain the semantics defined in the library of communicative acts, this contrasts with the approaches in which the semantics of communicative acts is affected by the protocol [7]. Moreover the performance of certain communicative acts changes the value of some environmental variables.

- The auctioneer declares the auction open (state s_0).
  $guards: A.state() = "closed"$
  $\mathbf{declare}(a, A.state = "open")$
- The auctioneer declares the current ask-price of the ongoing auction (state s_1, s_6, s_{11}).
  $guards: A.askPrice() < newAskPrice$
  $\mathbf{declare}(a, A.askPrice() = newAskPrice)$
- The auctioneer makes the "call for proposals" (state s_2, s_5, s_7, s_{10}).
  $\mathbf{request}(a, p_j, S_{j,i})$
  $effects: C_{id1j}(unset, p_j, a, S_{j,i})$
- One participant makes its proposal (state s_3, s_8).
  $guards: \{(t_{system} < t_{max}), (t_{system} - t_{bid} < t_{inactivity})\}$
  $\mathbf{propose}(p_j, a, P_j, Q_{j,v})$
  $effects: \{C_{id2j}(unset, a, p_j, P_j|Q_{j,v}),$
  $C_{id3j}(pending, p_j, a, Q_{j,v}|U_{j,v}),$
  $S_{j,v}.truth\_value() = 1\}$
- If the value of the proposal is greater than the current ask-price the auctioneer has to accept it (state s_4, s_9).
  $guards:\ v > A.askPrice();$
  $\mathbf{condAccept}(a, p_j, C_{id2j}(unset, a, p_j, P_j|Q_{j,v}), W_v)$
  $effects: \{C_{id2j}(pending, a, p_j, P_j|[Q_{j,v}, W_v]),$
  $\forall\ v' < v\ W_{v'}.truth\_value() = 0\}$
  $variable\ updates: \{newAskPrice = v,\ t_{bid} = t_{system}\}$
- If the value of the proposal is less than or equal to the current ask-price the auctioneer has to reject it (state s_4, s_9).
  $guards:\ v \leq A.askPrice();$
  $\mathbf{reject}(a, p_j, C_{id2j}(unset, a, p_j, P_j|Q_{j,v}))$
  $effects: \{C_{id2j}(cancelled, a, p_j, P_j|Q_{j,v}),$
  $U_{j,v}.truth\_value() = 0\}$
- The auctioneer can declare closed the auction only if the time of inactivity is equal to the constant value defined at the beginning of the auction or when the fixed end time of the auction is reached (state s_3, s_8).
  $guards: \{(t_{system} \geq t_{max}) \vee$
  $(t_{system} - t_{bid} > t_{inactivity}), A.state() = "open"\}$
  $\mathbf{declare}(a, A.state = "closed")$
  $effects: \{W_{value_{win}}.truth\_value() = 1,$
  $U_{j,value_{win}}.truth\_value() = 1\}$
  $variable\ updates: \{t_{end} = t_{system},$
  $value_{win} = newAskPrice\}$

4.2 Interaction Diagram

The interaction diagram that specifies the English auction Protocol is reported in Figure 2. It is possible to verify that an actual interaction complies with the protocol by checking that the sequence of communicative acts bring from the unique start state to one of the final states. Moreover it is possible to prove the soundness of this protocol specification. In particular the contents of each state accessible through different paths results equivalent. For states s_9, s_{10}, s_{11}, that are in the loop of the protocol, it is necessary to identify a loop invariant describing p_j as the client who made the highest offer.

5. CONCLUSIONS

In this paper we presented an application independent method for the definition of interaction protocols, based on the meaning of the exchanged messages, that can be used to define patterns of interaction in open, dynamic, and heterogeneous agent systems. The method proposed is based on a general ACL, whose semantics is defined in terms of commitments, and on a further component defining protocol-specific interaction rules. The resulting interaction protocols

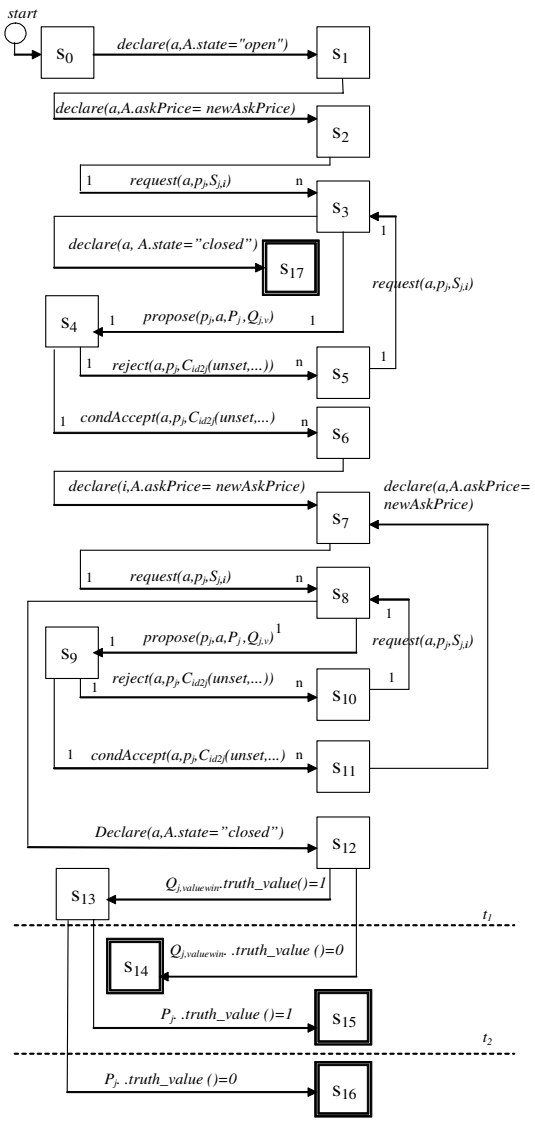

Figure 2: Interaction diagram of the English Auction Protocol.

are verifiable, in the sense that is possible to test whether an agent is behaving in accordance to it. Moreover, soundness condition are proposed to verify if a the structure of a given interaction protocol is reasonable. We also show how our method can be used to define a complex and common interaction protocol, the English Auction.

With respect to our previous operational proposal [5] of a commitment-based ACL semantics, in this paper we introduce a treatment of conditional acceptance and of declarations.

Our method for the definition of interaction protocols differs from most existing proposals, in that it is based on the use of an application-independent library of communicative acts, whose meaning is fully preserved when they occur in a protocol. With respect to the proposal put forward by Yolum and Singh in [10], our approach is focussed on the protocol design phase more than on the possibility of shortcutting predefined interaction patterns at run time. Indeed, we expect that agents used in practical application will mostly be simple reactive agents; if this idea is correct, proving the soundness of a protocol at design time is more important than allowing agents to plan intelligent variations of existing protocols. In principle, however, a deliberative agent with reasoning capabilities could understand our protocols on the basis of an ontology of commitment, linguistic knowledge (i.e., knowledge of a Communicative Act Library with semantics), and the ability to reason on interaction diagrams (i.e., a version of finite state machines).

6. REFERENCES

[1] M. Colombetti and M. Verdicchio. An analysis of agent speech acts as institutional actions. In *Proc. First International Joint Conference on Autonomous Agents and MultiAgent Systems AAMAS 2002*, pages 1157–1164, New York, 2002. ACM Press.

[2] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall International, 1976.

[3] M. d'Inverno, D. Kinny, and M. Luck. Interaction protocols in agentis. In *Proceedings of the Third International Conference on Multi-Agent Systems*, pages 112–119. IEEE Press, 1998.

[4] M. Esteva, J. Rodríguez-Aguilar, C. Sierra, P. Garcia, and J. L. Arcos. On the formal specification of electronic institutions. *Lecture Notes in Computer Science*, 1991:126–147, 2001.

[5] N. Fornara and M. Colombetti. Operational specification of a commitment-based agent communication language. In *Proc. First International Joint Conference on Autonomous Agents and MultiAgent Systems AAMAS 2002*, pages 535–542, New York, 2002. ACM Press.

[6] H. Mazouzi, A. E. Fallah, and S. Haddad. Open protocol design for complex interactions in multi-agent systems. In *Proc. First International Joint Conference on Autonomous Agents and MultiAgent Systems AAMAS 2002*, pages 517–526, New York, 2002. ACM Press.

[7] J. Pitt and A. Mamdani. A protocol-based semantics for an agent communication language. In D. Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol1)*, pages 486–491, S.F., 1999. Morgan Kaufmann Publishers.

[8] J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, United Kingdom, 1969.

[9] J. R. Searle. Conversation. In H. Parret and J. Verschueren, editors, *(On) Searle on Conversation*, pages 7–29. J. Benjamins Publishing Company, Amsterdam, 1991.

[10] P. Yolum and M. P. Sing. Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proc. First International Joint Conference on Autonomous Agents and MultiAgent Systems AAMAS 2002*, pages 527–534, New York, 2002. ACM Press.