

Representing Agent Interaction Protocols with Agent UML

Marc-Philippe Huget¹ and James Odell²

¹ Leibniz-IMAG/MAGMA
46, Avenue Félix Viallet
F-38031 Grenoble Cedex
France

Marc-Philippe.Huget@imag.fr

² Agentis Software, Inc.
3646 W. Huron River Dr.
Ann Arbor, MI 48103
USA
email@jamesodell.com

Abstract. Several modeling techniques exist to represent agent interaction protocols mostly based on work done in distributed systems. These modeling techniques do not take the agent features such as the autonomy into account. Agent Interaction Protocol designers are now considering specific modeling techniques that contain these features. In this paper, we present the second version of the Agent UML interaction diagrams dedicated to interaction protocols, and based on UML 2.0.

1 Introduction

Designing an agent interaction protocol is realized via several steps (mostly based on the ones found in communication protocol engineering [2]). The main step is certainly the formal description phase in which the informal description of the protocol is formalized into a formal description. This step is crucial since it conditions the protocol design success: an incomplete formal description will lead to an implemented protocol that does not answer to user needs. Currently, there exist several (formal and semi-formal) description techniques to describe protocols mostly based on work performed in communication protocol engineering (such as automata [1] or Petri nets [3]) or specifically designed to agent interaction protocols (such as Agent UML or ANML [7]). Agent Interaction Protocol designers create from scratch new formal description techniques in order to cope with agent requirements such as autonomy. Agent UML was designed with this requirement in mind. Actually, Odell and Bauer—the fathers of Agent UML—designed a modeling language which is an extension of an acknowledged modeling language, UML [6]. As Odell et al. explain it in [5], it is worthwhile to define a modeling language that is a refinement of a well-known modeling language since in this case, learning this one will be simplified. Moreover, UML is widespread in industry, thus it will help software engineers moving from software systems

to multiagent systems. Finally, several strong industrial tools already exist for UML. All these concerns give birth to Agent UML in 1999.

Recently, UML knew a major improvement via the UML 2.0 specification [6] and past UML 1.x sequence diagrams were greatly modified. Thus, it seems reasonable to update Agent UML in order to now consider the UML 2.0 interaction diagrams as background of Agent UML. In this paper, we present the new specification of Agent UML Interaction diagrams based on UML 2.0 Interaction diagrams.

The remaining of this paper is structured as follows. Section 2 describes the UML 2.0 Interaction diagram specification. For sake of simplicity and brevity, we omit some parts in the description. This is due to the relationships of elements in the Interaction diagram with elements outside this diagram. For instance, an Interaction is a specialization of InteractionFragment and Behavior, an InteractionFragment is a specialization of NamedElement and so on. As much as possible, we try to let the section readable by readers that are not expert in UML 2.0. The Agent UML Interaction diagrams are defined as a profile. It means that an Agent UML Interaction diagram is defined from UML Interaction diagrams to which some elements are modified, and some new elements are added. Section 3 depicts the Agent UML Interaction diagram profile. Section 4 describes an example of Agent UML Interaction diagram. Finally, Section 5 concludes the paper and discusses future directions of this work.

2 UML 2.0 Interaction Diagrams

The inter-process communication is captured by the set of Interaction diagrams in UML 2.0 [6]. Actually, the Interaction diagrams represent a family of diagrams:

Sequence diagrams. A diagram that depicts an interaction by focusing on the sequence of messages that are exchanged, along with their corresponding event occurrences on the lifelines. Unlike a communication diagram, a sequence diagram includes time sequences but does not include object relationships. A sequence diagram can exist in a generic form (describes all possible scenarios), and in an instance form (describes one actual scenario). Sequence diagrams are the most common form to represent protocols.

Interaction Overview diagrams. A diagram that depicts interactions through a variant of activity diagrams in a way that promotes overview of the control flow. It focuses on the overview of the flow of control where each node can be an interaction diagram.

Communication diagram. A diagram that focuses on object relationships where the message passing is central. The sequencing of messages is given through a sequence numbering scheme. Sequence diagrams and communication diagrams express similar information but show it in different ways.

Timing diagram. An interaction diagram that shows the change in state or condition of a lifeline over linear time. The most common usage is to show the change in state of an object over time in response to accepted events or stimuli.

In this paper, we focus on the sequence diagrams since it is the main one. We briefly consider the other diagrams in Section 5 and particularly, their possible use for agent interaction protocols. The remaining of this section describes the different classes in the UML 2.0 Interaction diagram specification. For sake of simplicity, we direct the explanations to be more readable. As a consequence, some classes are omitted or explanations are reduced. For a detailed description of the specification, readers can consult [6].

Even if it is not written in the UML 2.0 Interaction diagram specification, sequence diagrams are timely ordered from the top of the diagram to the bottom. It means that, except if we are using Continuations³, it is possible to easily order message sequences by reading the diagram from top to bottom. The X axis represents the different participants in the communication. Three notions are integrated within UML 2.0 Interaction diagrams: *traces*, *event occurrences* and *fragments*. If we do a parallel with protocols, a trace is a legal sequence of messages. An event occurrence is any event that can intervene during the communication such as message sending or receiving. A fragment is a piece of an interaction. It means it is the lifelines involved in the fragment and the set of traces (eventually modified by some operators defined below).

2.1 Interaction

A sequence diagram in UML 2.0 is organized around a frame called *Interaction*. This frame is defined as a unit of behavior and contains, among others, the protocol name (prefixed by the keyword *sd* for sequence diagram), the set of objects that are in relation and the sequence of messages between these objects. This Interaction is not a closed unit since it can send (receive) messages from (to) other sequence diagrams via *Gates*. An Interaction is depicted by a solid-outline rectangle with a pentagon in the upper left corner of the rectangle as shown on Figure 1. The content of the pentagon is the keyword *sd* followed by the protocol name and eventually parameters. The notation within the frame comes in several forms Sequence diagrams, Interaction Overview diagrams, Communication diagrams or Timing diagrams.

2.2 Lifeline

A Lifeline in UML 2.0 represents an individual participant in the interaction. A Lifeline describes as well the presence of the participant in the interaction. A participant entering later in the interaction has a Lifeline lower than others. A participant ending prematurely the interaction has a Lifeline finishing before the others. The Lifeline notation is a symbol consisting of a rectangle forming its head followed by a vertical line that represents the lifeline of the participant as shown on Figure 1. Information identifying the lifeline is displayed inside the rectangle in the following format: *name : class name* where *name* is the instance of the class called *class name*. On Figure 1, *xx* is a class name and *w* is a class instance.

³ Classes from UML 2.0 or from Agent UML have their first letter capitalized.

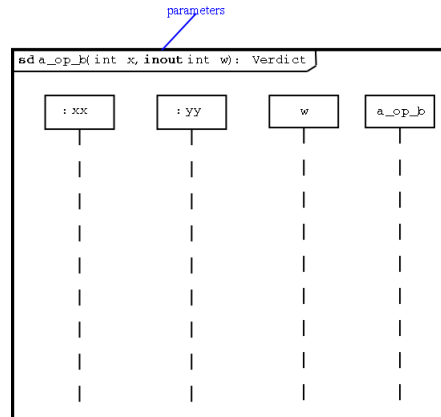


Fig. 1. The UML 2.0 Interaction Notation.

2.3 Message

A Message defines a particular communication between Lifelines of an interaction. A Message is defined between a sender and a receiver. There are several kinds of Message in UML 2.0:

- a message is *complete* if there is a sending event occurrence and a receiving event occurrence.
- a message is *lost* if it is known that there is a sending event occurrence but there is no receiving event occurrence. This is particularly the case in unreliable communication.
- a message is *found* if a receiving event occurrence is known but there is no (known) sending event occurrence. This is the case where the sender is outside the scope of the description. It could be an activity that this communication does not take into account.

The Message notation is a directed line from the sender lifeline to the receiver lifeline. The form of the line or the arrow head reflects the properties of the message:

- an asynchronous message has an open arrow head.
- a synchronous message typically represents method calls and is shown with a filled arrow head. The reply message from a method has a dashed line.
- an object creation message has a dashed line with an open arrow head.
- a lost message is described as a small black circle at the arrow end of the message.
- a found message is described as a small black circle at the beginning of the message.

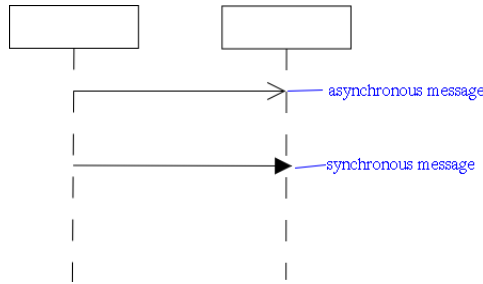


Fig. 2. The UML 2.0 Message Notation.

2.4 Constraint

A constraint in UML 2.0 is called a *StateInvariant*. It describes a constraint on the state of a Lifeline. If the constraint is evaluated to true, next event occurrences are considered as invalid and are not executed. The *StateInvariant* notation is shown as a text in curly brackets on the lifeline on which the constraint is applied. A second kind of constraints exists in the specification: the *InteractionConstraint*. This constraint is used in conjunction with *CombinedFragments*. The notation is a text within square brackets.

2.5 CombinedFragment

The *CombinedFragment* class in UML 2.0 represents a concise manner to represent several traces. The semantics of the *CombinedFragment* depends of the *InteractionOperator* used. There are several *InteractionOperators*:

Alternative. This *InteractionOperator* describes that several traces in the interaction are possible but at most one will be executed. The selection is based on guards. At most one guard is satisfied. The associated trace is executed. A default trace *else* can be added if no other trace can be executed.

Option. This *InteractionOperator* contains a single trace. Two situations are possible: if the guard associated to the trace is satisfied, then the trace is executed else nothing happens. This *InteractionOperator* is a particular case of the *InteractionOperator Alternative*.

Break. The *InteractionOperator Break* represents a breaking scenario that stops the current trace execution and executes the trace present in the *CombinedFragment*. The broken current execution will not be resumed.

Parallel. The *InteractionOperator Parallel* describes that several traces can be executed concurrently. The set of traces can be interleaved in any order.

Weak Sequencing. This *InteractionOperator* represents a weak sequencing between the different event occurrences in the *CombinedFragment*. It implies that it is possible to order the Message on a same Lifeline but it is not possible to make any assumption for Message ordering of other Lifelines in the same *CombinedFragment*.

Strict Sequencing. This InteractionOperator refines the InteractionOperator Weak Sequencing by requiring that all Messages in the CombinedFragment are timely ordered.

Negative. This InteractionOperator describes the set of traces that are invalid.

Critical Region. The InteractionOperator Critical Region describes a region on which it is not possible to interleave the set of traces within the Critical region with other messages outside the Critical Region. It corresponds to the critical section in distributed systems.

Ignore/Consider. The InteractionOperator Ignore describes the set of traces that can occur during the communication but has to be considered as insignificant and has to be treated like that. The InteractionOperator Consider is the converse of the InteractionOperator Ignore. It represents the set of messages that has to be considered.

Assertion. The InteractionOperator Assertion describes the only set of traces that is valid. This InteractionOperator is often combined with the InteractionOperators Ignore and Consider.

Loop. The InteractionOperator Loop depicts that the CombinedFragment represents a loop. The guard associated with the iteration is either a range with a lower and an upper bounds, or a boolean expression. The loop is executed as long as the guard is satisfied.

The CombinedFragment notation is a solid-outline rectangle with a pentagon in which the InteractionOperator is written. InteractionOperators are written as follows: *alt* for alternatives, *opt* for options, *break* for break, *par* for parallel, *seq* for weak sequencing, *strict* for strict sequencing, *neg* for negative, *critical* for critical region, *ignore* for ignore, *consider* for consider, *assertion* for assertion and *loop* for loop.

Some InteractionOperators have a special notation: (ignore | consider){<message name> {, message}*} and loop [<min int>, <max int>].

A CombinedFragment example can be found on Figure 11.

2.6 Continuation

A Continuation is a syntactic way to represent continuations of different branches of an Alternative CombinedFragment. Continuations is intuitively similar to labels representing intermediate points in a flow of control. The notation is a rounded-rectangle with a name within. The name corresponds to the label.

2.7 InteractionOccurrence

An InteractionOccurrence corresponds to a call to another interaction diagram. The current interaction diagram is resumed when the called interaction diagram ends. The notation of an InteractionOccurrence is a solid-outline rectangle with a pentagon. The keyword *ref* is written within the pentagon. The name of the called interaction diagram is written within the rectangle.

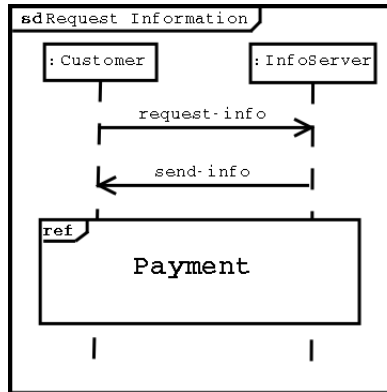


Fig. 3. The UML 2.0 InteractionOccurrence Notation.

2.8 Gate

A Gate is a connection point to relate a Message outside the Interaction with a Message inside the Interaction. Gates are points on the frame corresponding to the end of Messages, the one outgoing the Interaction and the one incoming the Interaction.

2.9 Termination

The Stop denotes the end of participation of a lifeline in the communication. The Stop is depicted by a cross in the form of an X at the bottom of a Lifeline. An example can be found on Figure 11.

3 Agent UML Interaction Protocol Profile

UML is an extensible language through stereotypes and tagged values. We have defined the Agent UML Interaction diagram specification as a UML profile. We define in this section the modification we have done on the UML Interaction diagram specification to give birth to Agent UML Interaction diagram specification.

3.1 Interaction

There are two kinds of protocols in multiagent systems: protocol templates and instantiated protocols. Template protocols represent reusable patterns for useful protocol instances. The protocol as a whole is treated as an entity in its own right, which can be customized for other problem domains. A protocol template is not a directly usable protocol because it has unbound parameters. Its parameters

must be bound to actual values to create a bound form that is an instantiated protocol. Protocol templates refer to abstract classes in object-oriented theory.

A protocol template is depicted by an Interaction where the pentagon contains the keyword `<<template>>` between the keyword `sd` and the protocol name. An instantiated protocol has parameters that are depicted as a Note linked to the Interaction via a dashed line. The first element in the Note is the keyword `<<parameters>>`. Frequent parameters are the ontology, the content language and the agent communication language as shown on Figure 4. These three parameters are introduced by the keywords *ontology*, *CL* and *ACL*. As a consequence, parameters in UML 2.0 Interaction are removed and replaced by these ones. Parameters can be written without referring to an unbound parameter as long as there is no confusion.

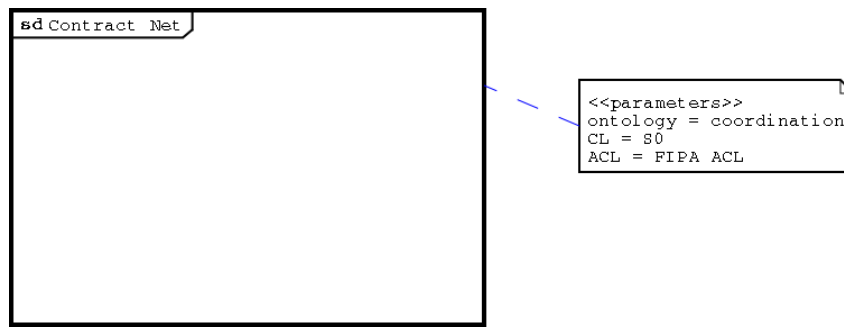


Fig. 4. The Agent UML Interaction Notation.

3.2 Lifeline

Lifelines are the elements that were the most modified. First, Lifelines in Agent UML can represent a set of agents and not a unique agent as sketched in UML 2.0. Agent UML proposes to add roles in order to reduce the size of interaction diagrams and group agents that have the same behavior in this interaction. Actually, an agent can be described with its role or with its role and its group. There are five possible ways to represent the content of the head of the lifeline:

1. an agent identity denotes an agent instance, for instance Smith,
2. an agent identity with a role denotes an agent instance playing a specific role, for instance Smith:Employee,
3. an agent identity with a role and a group denotes an agent instance playing a specific role in a particular group, for instance Smith:Employee/ACME,
4. a role denotes a role regardless of the agents playing this role and,
5. a role and a group denotes a role in a group regardless of the agents playing this role.

A role is prefixed by a colon. A group is prefixed by a slash. A cardinality can be added to represent the number of agents playing a specific role. The cardinality can be an exact number, a range or a logic formula or a condition. Figure 5 summarizes the different notation.

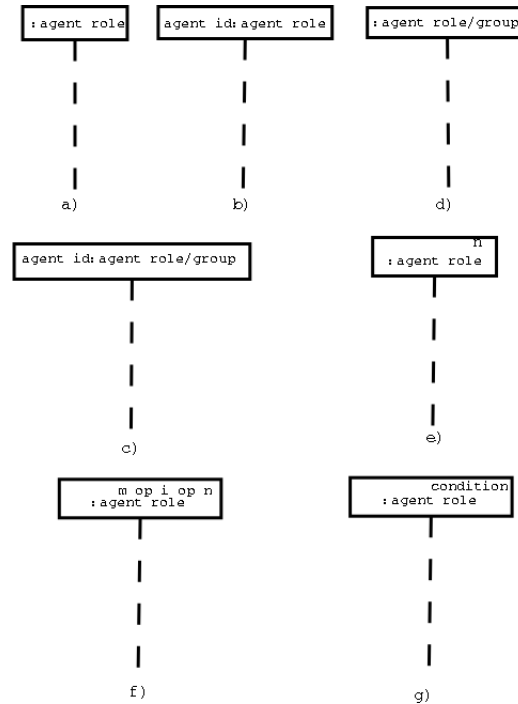


Fig. 5. The Agent UML Lifeline Notation.

The second important modification is the ability to write the role dynamics. An agent is able to add a new role or to change from one role to another one. We add two new stereotypes for this role dynamics: `<<add role>>` and `<<change role>>`. Changing or adding a role consists in drawing a directed line with an open arrow head from the current agent instance with the roles it plays to the new situation. The stereotype is written on the directed line as shown on Figure 6.

3.3 Message

We add new stereotypes in Agent UML Message in order to take account of roles, communication on the same lifeline and message delay.

Agent UML interaction diagrams can address a specific agent instance or a set of agents denoting by their role. As a consequence, it is required to write on

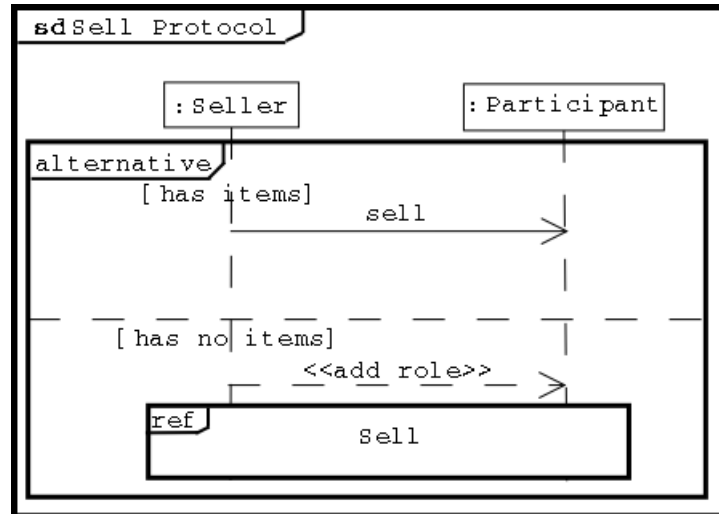


Fig. 6. The Agent UML Role Dynamics Notation.

message if it addresses a specific agent when the Lifeline only refers a role. The notation is to write the name of the agent instance on the Message close to the receiver Lifeline. A second point is issued from the use of roles. It is possible that a portion of the agent instances playing this role is concerned by this Message. A cardinality is adorned on the message to denote the number of agents that will receive this Message.

Since Agent UML considers roles, it is possible that a Message is sent from one agent instance in this role to another agents in this same role. It is important to address the case whether the sender wants to receive the Message as well. We add one new stereotype to cope the situation where the sender does not want to receive the Message in an asynchronous communication. It is not possible to have the sender in the list of receivers if the communication is synchronous since in this case, the sender will be deadlocked.

Figure 7 summarizes the Agent UML Message notation. The notation *a* corresponds to an asynchronous message sending. The notation *b* corresponds to a synchronous message. The notation *c* gives the cardinality for both sender and receiver. The notation *d* depicts the cardinality for the receiver in terms of a range. The notation *e* denotes the cardinality for the receiver in terms of a condition. The notation *f* denotes that this Message is sent to a specific agent instance when the Lifeline only refers to a role. The notation *g* corresponds to sending asynchronously the Message to the same Lifeline, the sender will receive a copy of the Message. The notation *h* depicts sending asynchronously the Message to the same Lifeline and the sender will not receive a copy of the Message. The notation *i* denotes to sending synchronously the Message to the same Lifeline, the sender will not receive a copy of the Message.

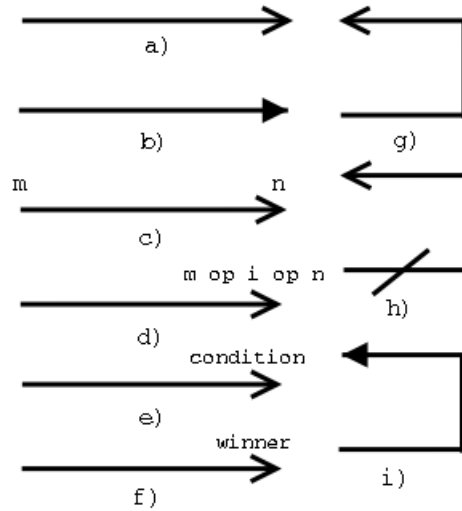


Fig. 7. The Agent UML Message Notation.

Finally, a Message can be delayed due to network congestion. It is then required to add that a specific message sent before another one can arrive after it. A new stereotype is added to denote this message switching. The delayed Message is represented as a multi-directed line and its point on the receiver Lifeline is after the Message sent after it. The Message that arrives before the Message sent before it is represented with a bridge on the directed line to avoid line crossing as shown on Figure 8.

3.4 Constraint

Two kinds of constraints are considered in Agent UML interaction diagrams: constraints and timing constraints. Timing constraints are conformed to the one defined in UML 2.0. Constraints are refined into two categories: blocking constraints and non-blocking constraints. Non-blocking constraints correspond to the constraints defined in UML 2.0. If the constraints are satisfied then the InteractionFragment is executed. In the case of blocking constraints, agents are blocked as long as the constraints are not satisfied. If a blocking constraint is applied to a Lifeline where a role is declared without agent instances, all the agent instances playing this role are blocked as long as the constraints are not satisfied. The stereotype `<<blocking>>` is added to represent blocking constraints as shown on Figure 9.

3.5 Protocol Template

The purpose of protocol templates is to create reusable patterns for useful protocol instances. The protocol as a whole is treated as an entity in its own right,

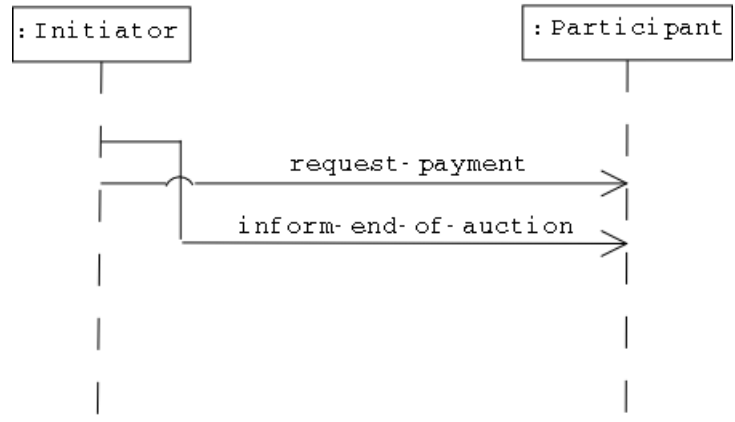


Fig. 8. The Agent UML Delayed Message Notation.

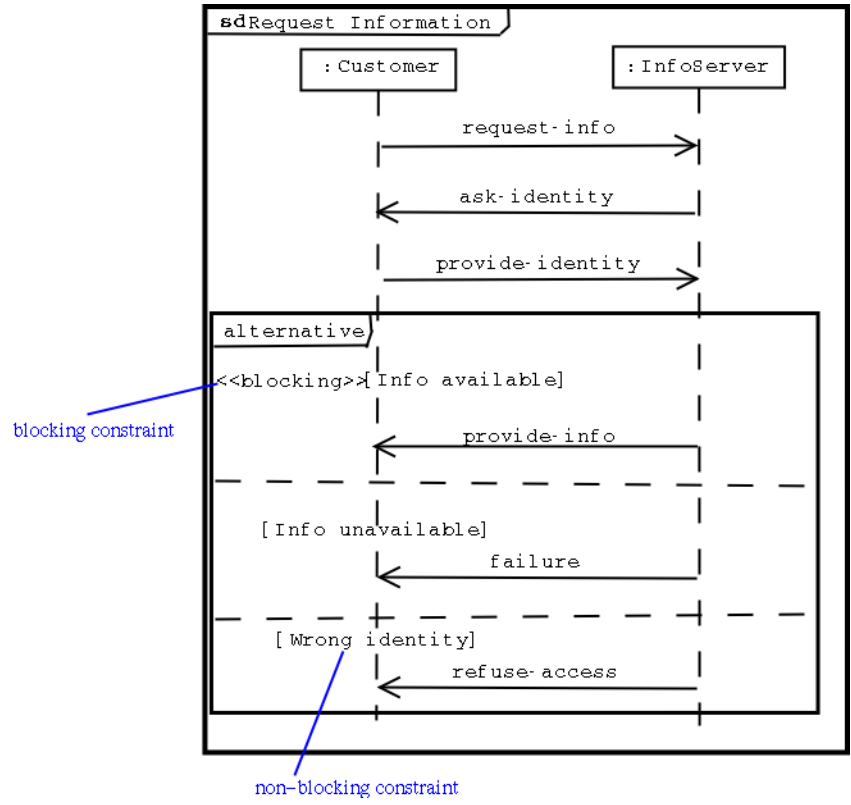


Fig. 9. The Agent UML Blocking Constraint Notation.

which can be customized for other problem domains. A parameterized protocol is not a directly usable protocol because it has unbound parameters. Unbound parameters are distinguished from bound parameters by the stereotype «*unbound*». Unbound parameters will be bound in instantiated protocols via the note linked to the interaction diagram as shown on Figure 4.

3.6 Action

Sending and receiving messages imply performing actions within agents. For instance, if we refer to FIPA ACL, sending a inform message implies verifying that the sender believes the content of the message; and receiving the inform message entails that the receiver believes the message content. An action is depicted as a round-cornered rectangle linked to the message that triggers it by an association. The action is written within the round-cornered rectangle. The action is written as a text independent of any programming language. The executable language from Mellor and Balcer is a possible language to represent actions in Agent UML [4].

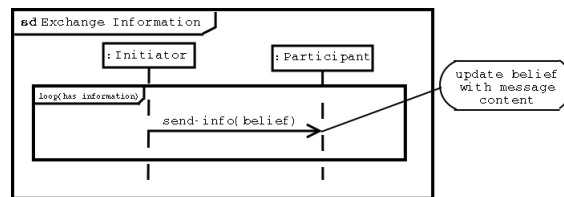


Fig. 10. The Agent UML Action Notation.

4 Agent UML Sequence Diagram Example

We take the example of the FIPA Request When protocol to exemplify some classes, we described in Section 3. The FIPA Request When protocol allows an agent to request that the receiver performs some action at the time a given precondition becomes true. The Agent UML representation of this protocol is given on Figure 11. The protocol is composed of two roles: *Initiator* and *Participant* denoted by the two Lifelines. Since there are Lifelines with roles regardless of the agent instances, each Lifeline can represent several agents. However, a piece of information may be added on the first Lifeline that there is one and only one Initiator.

All the Messages are asynchronous. The first one is sent from the Initiator role to the Participant role. After this sending, the receiver role can answer either with a refuse Message or with an agree Message denoted by the CombinedFragment with the InteractionOperator Alternative. We do not add an *else* clause since the

two conditions are opposite. The conditions are written as InteractionConstraint within square brackets. Here, the conditions are written as text but it is also possible to use some logic formulae. In case, the Participant refuses to answer to the request of the Initiator, it replies by the refuse Message. Just after, the Interaction stops since there has a Stop. If the Participant accepts, it replies by the accept Message and the Interaction continues. We add a blocking constraint. As long as the precondition holds, the Interaction is blocked and the agent instances as well. When the preconditions are no longer satisfied, the Participant has three different answers, either a failure Message if it fails performing the action, an inform-done Message to inform that the action is performed or an inform-result Message to give the result of the action. Since there is no more Messages in the Interaction, the Interaction stops.

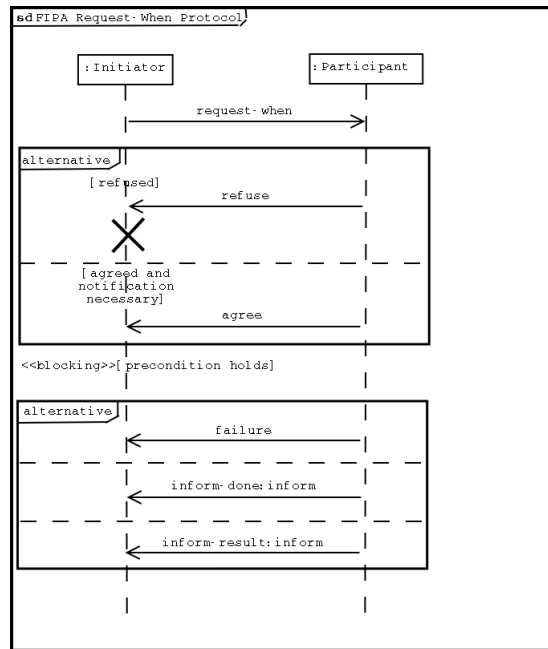


Fig. 11. The FIPA Request When Protocol.

5 Conclusion

Describing agent interaction protocols is an important task in multiagent system design since a faulty protocol can affect the way agents behave and cooperate. There exist several different formal description techniques available to agent

interaction protocol designers: automata, Petri nets, logic or Agent UML to name a few. Some description techniques fit more or less the agent desiderata and particularly the decision autonomy. To tackle this point, agent interaction protocol designers considered new description techniques and among others, the Agent UML [5]. The Agent UML idea presents several interesting ideas:

1. It is rooted on the well-known and acknowledged modeling language UML
2. Thanks to UML, it eases the gap between industrial concerns and research concerns. It is easier to learn it than a complex or a logic formal description technique
3. Several industrial tools are available
4. It covers the agent needs in terms of interaction

Even if this picture is quite idyllic, the first version of the Agent UML was of limited scope and does not take account of several operators such as the InteractionOperator break or loop. Moreover, the timing constraints were written as a note reducing its use when automatically implementing this protocol. Thanks to a growing effort from the community and particularly via the FIPA Modeling technical committee, the Agent UML takes a second birth. The new Agent UML is now rooted on the UML 2.0 Interaction diagram specification, which offers more possibilities for agent interaction protocols.

As stated in the introduction, the interaction diagram family contains four different diagrams. We only address here the sequence diagram but there are as well the interaction overview diagram, the communication diagram and the timing diagram. Even if we do not speak about them, they can be of interest for agent interaction protocols. The interaction overview diagram can be used to intertwine agent interactions and control flow, and particularly the relationships between different protocols. The communication diagram can help designers during validation to test if agents correctly receive the message and the message ordering. Finally, timing diagrams can be used to check agent design and agent interaction protocol. It is then possible to verify that an agent takes a specific state when sending or receiving a message.

Several directions are already written in the agenda of Agent UML:

1. A comparative study between existing formal description techniques and Agent UML has to be performed in order to check what is missing and what is the expressive power of Agent UML. We particularly think about Petri nets and the study done on the previous version of Agent UML [3],
2. UML presents a main issue, this is a semi-formal language since there is no formal semantics of the Interaction diagrams. As a consequence, ambiguities and misunderstandings are possible. Moreover, such semantics is required when designers want to realize tools and implementation of Agent UML Interaction diagrams. In order to tackle this point, we envisage to describe the Agent UML sequence diagrams via communicating extended finite state machines.
3. Agent UML is a new modeling language and, as a consequence, there is actually no tools for the diagram design, validation or implementation.

4. Agent UML Interaction diagrams are actually only applied to the FIPA Interaction protocol library. We envisage to search for other protocols in order to check if this specification is consistent and answers user needs.

Acknowledgements

Authors would like to thank people who contribute directly or indirectly to this specification via the FIPA Modeling technical committee or via mails.

References

1. M. Barbuceanu and M. S. Fox. COOL : A language for describing coordination in multiagent system. In *First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 17–24, San Francisco, USA, June 1995. AAAI Press.
2. G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, 1991.
3. H. Mazouzi, A. El Fallah Seghrouchni, and S. Haddad. Open protocol design for complex interactions in multi-agent systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, Bologna, Italy, July 2002.
4. S. J. Mellor and M. Balcer. *Executable UML*. Addison-Wesley, 2002.
5. J. Odell, H. V. D. Parunak, and B. Bauer. Extending UML for agents. In G. Wagner, Y. Lesperance, and E. Yu, editors, *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, Austin, Texas, july, 30 2000. ICue Publishing.
6. O. M. G. (OMG). *Unified Modeling Language: Superstructure version 2.0*, 03-04-01 edition, 2003.
7. S. Paurobally. *Rational Agents and the Processes and States of Negotiation*. PhD thesis, Imperial College, University of London, 2003.