Extending Agent UML Protocol Diagrams

Marc-Philippe Huget

Agent ART Group
University of Liverpool
LIVERPOOL L69 7ZF
United Kingdom
M.P.Huget@csc.liv.ac.uk

1 Introduction

Agents in multiagent systems use interaction in order to perform their tasks either by cooperation or by coordination. Interaction is driven by interaction protocols. These protocols inherit from communication protocols and as a consequence use formal description techniques employed by these one: finite state machines [14], Petri nets [7], the language Z [10], the language LOTOS [21], the language SDL [20] or temporal logic [13].

All these formalisms are efficient to represent interaction protocols but one point is missing quoted by Bauer et al: "In a previous paper, we have argued that UML provides an insufficient basis for modeling agents and agent-based systems [4]. Basically, this is due to two reasons: *Firstly*, compared to objects, agents are active because they can take the initiative and have control over whether and how they process external requests. *Secondly*, agents do not only act in isolation but in cooperation or coordination with other agents. Multiagent systems are social communities of interdependent members that act individually."

A new class of formal description techniques appears in order to tackle this point. FLBC [9], COOL [3] and AgenTalk [22] are some examples belonging to this class. A new graphical modeling language emerges recently. This is Agent UML [24] which is an extension of UML. UML is not used directly since it is too weak to represent the agent-based features described above.

This new language is now the modeling language of the association FIPA [11] instead of UAML They replace the UML sequence diagrams.

Protocol diagrams and Agent UML are two standards which still evolve (see our different proposals in [18, 15, 17]). The aim of this paper is to present some new features that we propose. Several features deal with reliability such as triggering actions and exception handling. These new features are in fact proposed due to our work in electronic commerce and in supply chain management [16]. As much as possible, we apply these new features to needs in the supply chain management example.

The remainder of the paper is defined as follows. In the first section, we present the Agent UML protocol diagrams. The presentation is inspired from the paper [5] which is the reference for protocol diagrams. Section 3 presents the new features we want to add to the current specification. Then, we give the example of the English Auction Protocol in Section 4. Section 5 concludes this paper and presents future work.

2 Protocol Diagrams in Agent UML

Several notions are encompassed in protocol diagrams: agents or agents' roles, messages, constraints on messages and the protocol unfolding. All these notions are explained in this section.

Agents and Agents' Roles

Agents on protocol diagrams can be represented by different manners: (1) they are represented by their identity called *instance* in Agent UML, (2) they are represented by their role in the protocol. For instance, in auction protocols, one has usually two roles: the *Seller* and the participants called *Auctioneer*. (3) finally, they can be represented by both their identity and their role.

The agent class can be given for completing the definition of the roles. Using roles reduce the size of the diagram. Actually, one does not need several identities but just one role if they all share the same role.

Agents are described on protocol diagram by a box and a text defined as follows: instance "/" role ":" class. For instance, for auction protocols, one could have Smith/Auctioneer. Figure 1 gives two roles: *Initiator* and *Participant*. The vertical bar specifies messages which are sent from or are sent to this role.

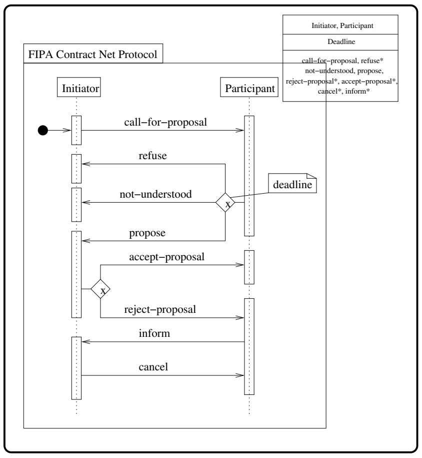

Fig. 1. Contract Net Protocol in Agent UML

Lifelines

Agents are represented by a lifeline which corresponds to their participation in the protocol. When a lifeline is created for a role, this role becomes active for the protocol. This lifeline is present as long as the role has actions in the protocol. Lifelines are defined by a vertical bar drawn on the vertical bar of roles or linked to the vertical bar (see Figure 11 with the messages *not-understood*). Lifelines can be splitted or merged when several alternatives are available in the protocol or several paths are merged in one. For instance, it is the case on Figure 1 where several answers are available: *propose*, *refuse* or *not-understood*.

Agent UML Connectors

Interaction protocol designers have three logical connectors for describing alternatives and choices (see Figure 2). The connector AND (see Figure 2a) means that messages have to be sent concurrently. The connectors OR (see Figure 2b) and XOR (see Figure 2c) mean that a choice between several messages has to be done. The connector OR means that zero or several messages can be chosen and the connector XOR means that one message has to be chosen. The term CA refers to communicative acts [2].

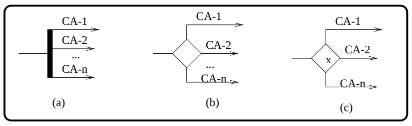

Fig. 2. Agent UML Connectors

Various Notions

Several other notions are present in protocol diagrams:

Conditions: It is possible to put some conditions on sending messages. For instance, for the English auction protocol (see Figure 11), a condition is given for the messages *not-understood*. The variable m must have a value greater than 0 if one wants to use this message. Conditions are given inside curly brackets.

Cardinality: Designers have to have the ability to send several copies of the same message (for advertising the opening of an auction for instance). In protocol diagrams, the multiple sending of messages has the meaning of multicast. Two numbers are defined: the first one is close to the vertical bar of the role which sends the message. The number is 1. The second number is close to the vertical bar of the role which receives the message. The value corresponds to the number of copies.

Type of message: Normally, messages are sent asynchronously (see Figure 3a). It is also possible to sent messages synchronously (see Figure 3b). The last possibility is when messages are not received instantaneously but after some delay (see Figure 3c).

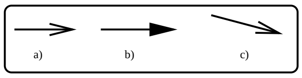

Fig. 3. Type of Messages in Agent UML

Comments: It is possible to insert some comments on diagrams, for instance *start cfp time* on the diagram of the Figure 11.

Repetition: One example of repetition is given on Figure 11 for the message requesting the payment. The curly brackets contain two numbers which represent how many times the message has to be sent. Here, the values range from 0 to 1 since the current price might be less than the reserved price, thus no payment is requested.

Nested and interleaved protocols: Since protocols are reusable, it is possible to link protocols together. Two solutions are available in Agent UML protocol diagram: nested protocols and interleaved protocols. Nested protocols correspond to protocols which are within another protocols. This case is used when one needs to repeat several times a protocol. Nested protocols continue while conditions hold.
Interleaved protocols correspond to protocols which are called during the execution of a second protocol. Bauer et al. give the example of an auction participant who requests some information about her/his bank account [6].

3 The proposed new features

Our work in the domain of reliability, electronic commerce and supply chain management [16] needs some features which are not present in Agent UML. Here is the list of these features:

1. broadcast
2. synchronization
3. triggering actions
4. exception handling
5. time management
6. atomic transactions
7. message sending until receiver acknowledges receipt

All these features are explained in detail in this section. We also describe how they are included in Agent UML.

Broadcast

The term *broadcast* means one message is sent to an undetermined set of agents. The sender is unable to quote all the agents since this message is sent to all agents on the network. One does not mistake the broadcast mechanism for the multicast one. Indeed, in multicast, agents are specified by the sender (even if the set of agents is numerous) and this mechanism is already present in Agent UML protocol diagrams. Designers have just to insert the number of agents on the arrow. The broadcast is represented by an arrow where the arrowhead is a circle (see Figure 4). Broadcast can also be applied to a structure such as groups, hierarchies or communities. In this case, the message is only sent to this structure. The name of the structure is adorned close to the arrowhead.

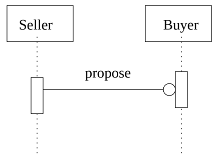

Fig. 4. Broadcast in protocol diagrams

The broadcast could be used when the manager of the task in the Contract Net protocol [8] announces the task. A second example is when the auctioneer informs agents that the auction begins or when it proposes a new item to sell. Another example is when agents are looking for services as Jini does it [1].

Synchronization

One example of synchronization is when agents in Sian's protocol [26] have to decide if one assumption is true, false or if the assumption needs to be modified. The synchronization allows agents to define a meeting point during interaction. All agents have to reach this point if agents want to follow the interaction. The synchronization is represented by an arrow where the arrowhead is crossed out by a vertical line (see Figure 5). The synchronization is realized on the message which labels this arrow.

Triggering actions

The triggering action management presents some similarity with exception management since in the two cases, they represent situations which are not the normal path in the interaction. However, a difference is noted between exception and triggering action. In the former, we represent abnormal situations and in the latter, situations occurring which need to send a particular message or to do some specific actions. One example of triggering action is during an ascending price

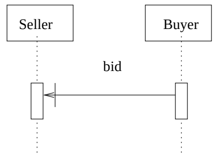

Fig. 5. Synchronization in protocol diagrams

auction when an agent sends a proposal which is lower than the current price. In this case, the auctioneer warns the agent of this error by sending a message. It is not an exception since this message does not endanger the auction.

The triggering actions are located on arrows inside parentheses (see Figure 6). The conditions which trigger some actions are written there and the message which is sent when conditions become true. Conditions can be written with OCL [25] for a formal representation or with a text.

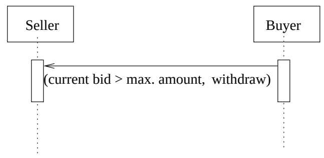

Fig. 6. Triggering actions in protocol diagrams

The triggering actions could be applied to the Supply Chain Management when the agent workers do not finish their tasks at time. Then, a message can be sent to the dispatcher agent informing of this situation [16].

Exception management

In our opinion, the notion of exception is barely used in interaction protocols and the situation seems to be the same in communication protocols. We can only cite the article of Moore [23]. According to us, the notion of exception in protocols are important since it allows designers to get rid off abnormal situations coming from either an improper use of protocols or when system performance slows down.

The notion of exception refers to a behavior which is not the expected behavior. It is for example the case when an agent answers in the interaction with a performative which does not belong to the list of allowed performatives for this state of interaction. The receiver can fire an exception and send a *not-understood* message to the sender.

A second use case is when the system performance drops and answer time increases. It might be interesting to insert an exception dealing with the delay between two messages. If the delay between two messages is greater than the allowed delay, one exception can be fired. Exceptions allow interactions to stop which can otherwise not terminate. It is, for instance, the case if the receiver of the previous message is disconnected or the network is down.

The Figure 7 gives how exceptions are represented in protocol diagrams. Designers have to insert the keyword *exception* followed by the exception. Designers insert the keyword *not* before exceptions for deleting them. Exceptions are written textually.

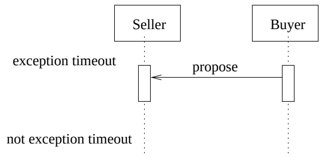

Fig. 7. Exceptions in protocol diagrams

An example of exception in the Supply Chain Management is when the client agent cancels the order which is negotiated at this moment. This cancellation stops definitively the protocol for placing an order.

Time management

The current proposal in protocol diagrams for representing delay and deadlines is to describe deadlines with a note on arrows, for instance, the example of the comment *deadline* on the Contract Net protocol when receiving the proposals (see Figure 1). Our proposal is to insert the time management at two levels:

1. One application domain of multiagent systems and interaction protocols is electronic commerce and negotiation. It could be interesting having a deadline on the whole interaction, for instance, when the interaction is stalled since one agent is disconnected or the network is down. A second case is when the negotiation is no longer progressing. The use of exception allows negotiations to stop which will otherwise not finish.
   This deadline is inserted at the beginning of the protocol diagram above the agents' roles since it is applied to the whole interaction. This deadline is given by the keyword *time* and a delay which is the allowed delay for the interaction (see Figure 8a).
2. It is necessary to give some delay between two messages as Agent UML proposes it with comments. This time, we give the delay inside parentheses under the arrow and preceded by the keyword "d:" (see Figure 8b).

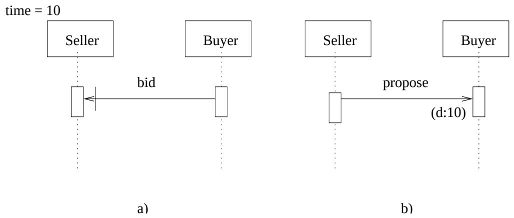

Fig. 8. Time management in protocol diagrams

Time management and exceptions seem to be a convenient way to deal with overrunning delays. In this case, agents can trigger an exception and do something in consequence.

Atomic transactions

The atomicity of transactions is a term coming from electronic commerce (and databases). It deals with security problems. Actually, atomicity of transactions means it is not possible to realize such a transaction if other linked transactions fail. For instance, the following situation must be forbidden: the financial transactions (credit and debit) are done but the delivery fails or the opposite.

The insertion of the atomicity in protocol diagrams is done by inserting all the messages in a package (see Figure 9). All the messages must be sent properly and if one of them fails, the other messages must be canceled. One example is for meeting scheduling. If one person can not receive the message, the sender has to cancel the other ones. The cancellation is performed by a new message sent to the receivers of this message.

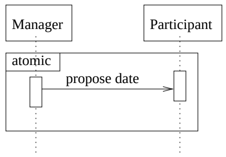

Fig. 9. Atomic transactions in protocol diagrams

Sending messages until delivery

In a utopian version of networks, all messages are sent and delivered to agents without loss. It is definitively not the case on the Internet, especially if the receiving agents can not be reached. The meaning of this new feature is to send the message until the receiving agents send an acknowledgment message. This feature is represented by an arrow where the arrowhead is a plain circle (see Figure 10). This feature needs agents use some acknowledgment process like the hand-shaking protocol.

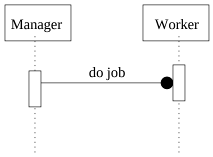

Fig. 10. Sending messages until delivery in protocol diagrams

This feature is interesting when we consider the domain of workflow. The acknowledgment could correspond to the performance of the action associated to this message.

4 One example of protocol in Agent UML

We take the example of the English Auction Protocol [12] in order to exemplify the use of protocol diagrams and some new features. The meaning of the English Auction Protocol is the following: one agent (the Auctioneer) wants to sell an item, it informs all the agents (or just a part of them which can be interested in it) (the Auction Participants) that a new auction begins. The Auctioneer informs the Auction Participants how much the item is. Then, the auction participants can inform the Auctioneer if they accept the bid. If the Auctioneer has more than two Auction Participants who accept the bid, it selects the one which sent first the agreement and informs it that it wins this round. It informs other Auction Participants who sent an offer that their bids are rejected. The auction is opened till it remains only one auction participant who accepts the bid.

When the Auctioneer closes the auction, it compares the current price for this item with the reserved price. If the current price is greater than the reserved price, the item is won by the last agent who accepts the bid. Auction Participants are informed the auction is closed.

The protocol in Agent UML is shown on Figure 11.

The proposed protocol presents some differences with the ones proposed by FIPA [12] and Bauer et al. [5].

The first message is sent to all agents on the network by broadcast whereas FIPA's approach and Bauer's approach consider to send it only to a set of agents represented by n on the protocol. This situation is repeated once for the second message which details the item and the price. The deadline for the bidding is given on the arrow. Bauer et al. propose to insert this deadline as

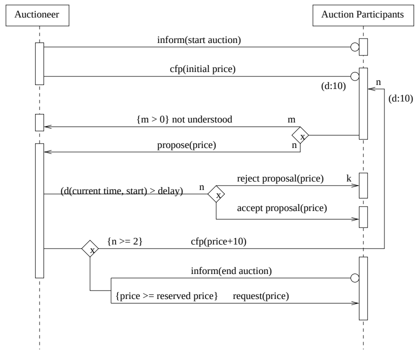

Fig. 11. The English Auction Protocol in Agent UML

a comment. Then, auction participants can answer either by a *not understood*[1] message or by a *propose* message. If auction participants send a *propose* message, they inform the auctioneer they accept the bid. When the deadline is passed (given by d[2](current time, start) > delay), the Auctioneer informs the Auction Participants if they have won this round (by a *accept proposal* message) or if they have lost this round (by a *reject proposal* message). The winner is the auction participant which sent first the acceptance of the bid. If the auctioneer has more than two agents which accept the price, it proposes once again the item at a higher price.

If the auctioneer has only one agent, it closes the auction and informs all the auction participants. If the current price is greater than the reserved price, it informs the last auction participant who accepts the bid that it wins the item.

The main differences with the two other versions are that we use the broadcast, deadlines, triggering actions and conditions on the end of the auction. Bauer et al's model is more complete than the one from FIPA.

5 Conclusion

In this paper, we present a semi-formal modeling language for describing interaction protocols called Agent UML which is an extension of UML. The advantage of such a language is that it is graphical and is based on a language which is well-known to programmers and companies. Programmers will have less difficulty to pass to Agent UML since they are used to developing with UML.

Interaction protocols are described with protocol diagrams in Agent UML. Agents can be described either with their identity or with their role in this inter-

---

[1] FIPA defines in its specifications that all FIPA agents have to be able to send the message *not understood* if they do not understand the content of the message or the message.

[2] The function d means the difference between the two parameters.

action. The unfolding of this interaction is given by the lifelines which represent the different paths in the interaction.

Our work in the domain of reliability, electronic commerce in multiagent systems and Supply Chain Management gives us some features which are not present in the current version of Agent UML such as: broadcast, synchronization, triggering actions, exception, deadline, atomic transactions and sending messages until acknowledgment. This paper shows how to represent protocols using these features in Agent UML. All these features will be submitted for an insertion into the Agent UML specification.

As soon as these features will be accepted by Agent UML community, we can insert them in our Agent UML exchange format language [18] and in our model checking process [19]. Finally, most of these features would be used in our Supply Chain Management [16].

References

1. Jini, Sun. http://www.sun.com/jini.
2. J. L. Austin. *How to Do Things with Words.* Clarendon Press, 1962.
3. 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.
4. B. Bauer. Extending UML for the specification of interaction protocols. Submission for the 6th Call for Proposal of FIPA and revised version of FIPA 99, 1999.
5. B. Bauer, J. Müller, and J. Odell. Agent UML: A formalism for specifying multiagent interaction. In P. Ciancarini and M. J. Wooldridge, editors, *Agent-Oriented Software Engineering (AOSE-00)*, 2000.
6. B. Bauer, J. P. Muller, and J. Odell. An extension of UML by protocols for multiagent interaction. In *International Conference on MultiAgent Systems (ICMAS'00)*, pages 207–214, Boston, Massachussetts, july, 10-12 2000.
7. R. S. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng. Modeling agent conversation with colored Petri nets. In J. Bradshaw, editor, *Autonomous Agents'99, Special Workshop on Conversation Policies*, May 1999.
8. R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem-solving. *Artificial Intelligence*, 20:63–109, 1983.
9. F. Dignum. FLBC: From messages to protocols. In F. Dignum and C. Sierra, editors, *European Perspective on Agent Mediated Electronic Commerce.* Springer Verlag, 2000.
10. M. d'Inverno and M. Luck. Formalising the contract net as a goal-directed system. In W. V. de Velde and J. Perram, editors, *Agents Breaking Away, MAAMAW 96*, number 1038 in Lecture Notes in Artificial Intelligence, pages 72–85. Springer-Verlag, 1996.
11. FIPA. *Specification: Agent Communication Language.* Foundation for Intelligent Physical Agents, http://www.fipa.org/spec/fipa99spec.htm, September 1999. Draft-2.
12. FIPA. *Specification.* Foundation for Intelligent Physical Agents, http://www.fipa.org/repository/fipa2000.html, 2000.
13. M. Fisher and M. Wooldridge. Specifying and executing protocols for cooperative action. In *International Working Conference on Cooperating Knowledge-Based Systems (CKBS-94)*, Keele, 1994.

14. A. Haddadi. *Communication and Cooperation in Agent Systems: A Pragmatic Theory*, volume 1056 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.
15. M.-P. Huget. Agent uml class diagrams revisited. Technical Report ULCS-02-013, Department of Computer Science, University of Liverpool, 2002.
16. M.-P. Huget. An application of agent uml to supply chain management. Technical Report ULCS-02-015, Department of Computer Science, University of Liverpool, 2002.
17. M.-P. Huget. Extending agent uml protocol diagrams. Technical Report ULCS-02-014, Department of Computer Science, University of Liverpool, 2002.
18. M.-P. Huget. A language for exchanging Agent UML protocol diagrams. Technical Report ULCS-02-009, Department of Computer Science, University of Liverpool, 2002.
19. M.-P. Huget. Model checking Agent UML protocol diagrams. Technical Report ULCS-02-012, Department of Computer Science, University of Liverpool, 2002.
20. C. Iglesias, M. Garrijo, J. Gonzales, and J. Velasco. Design of multi-agent system using mas-commonkads. In Springer-Verlag, editor, *Proceedings of ATAL 98, Workshop on Agent Theories, Architectures, and Languages*, volume LNAI 1555, pages 163–176, Paris, France, July 1998.
21. J.-L. Koning. Algorithms for translating interaction protocols into a formal description. In K. Ito, editor, *IEEE International Conference on Systems, Man, and Cybernetics Conference (SMC-99)*, Tokyo, Japan, October 1999.
22. K. Kuwabara, T. Ishida, and N. Osato. AgenTalk: Describing multiagent coordination protocols with inheritance. In *Seventh IEEE International Conference on Tools with Artificial Intelligence*, pages 460–465, Herndon, Virginia, November 1995.
23. S. A. Moore. On conversation policies and the need for exceptions. In *Autonomous Agents'99 Special Workshop on Conversation Policies*, 1999.
24. 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.
25. OMG. UML 1.4. Technical report, OMG, 2001.
26. S. S. Sian. Adaptation based on cooperative learning in multi-agent systems. In Y. Demazeau and J.-P. Müller, editors, *Decentralized AI*, volume II, pages 257–272, Amsterdam, The Netherlands, 1991. Elsevier Science Publishers B.V.