Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Technical Memo
TM-00-01

Specifying Agent Interaction Protocols with UML Activity Diagrams

Jürgen Lind
German Research Center for AI (DFKI)
Im Stadtwald B36
D-66123 Saarbrücken, Germany
lind@dfki.de

March 2000

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210
E-Mail: info@dfki.uni-kl.de

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341
E-Mail: info@dfki.de

WWW: http://www.dfki.de

Deutsches Forschungszentrum für Künstliche Intelligenz
DFKI GmbH
German Research Center for Artificial Intelligence

Founded in 1988, DFKI today is one of the largest nonprofit contract research institutes in the field of innovative software technology based on Artificial Intelligence (AI) methods. DFKI is focusing on the complete cycle of innovation — from world-class basic research and technology development through leading-edge demonstrators and prototypes to product functions and commercialization.

Based in Kaiserslautern and Saarbrücken, the German Research Center for Artificial Intelligence ranks among the important "Centers of Excellence" worldwide.

An important element of DFKI's mission is to move innovations as quickly as possible from the lab into the marketplace. Only by maintaining research projects at the forefront of science can DFKI have the strength to meet its technology transfer goals.

DFKI has about 115 full-time employees, including 95 research scientists with advanced degrees. There are also around 120 part-time research assistants.

Revenues for DFKI were about 24 million DM in 1997, half from government contract work and half from commercial clients. The annual increase in contracts from commercial clients was greater than 37% during the last three years.

At DFKI, all work is organized in the form of clearly focused research or development projects with planned deliverables, various milestones, and a duration from several months up to three years.

DFKI benefits from interaction with the faculty of the Universities of Saarbrücken and Kaiserslautern and in turn provides opportunities for research and Ph.D. thesis supervision to students from these universities, which have an outstanding reputation in Computer Science.

The key directors of DFKI are Prof. Wolfgang Wahlster (CEO) and Dr. Walter Olthoff (CFO).

DFKI's six research departments are directed by internationally recognized research scientists:

- ❏ Information Management and Document Analysis (Director: Prof. A. Dengel)
- ❏ Intelligent Visualization and Simulation Systems (Director: Prof. H. Hagen)
- ❏ Deduction and Multiagent Systems (Director: Prof. J. Siekmann)
- ❏ Programming Systems (Director: Prof. G. Smolka)
- ❏ Language Technology (Director: Prof. H. Uszkoreit)
- ❏ Intelligent User Interfaces (Director: Prof. W. Wahlster)

In this series, DFKI publishes research reports, technical memos, documents (eg. workshop proceedings), and final project reports. The aim is to make new results, ideas, and software available as quickly as possible.

Prof. Wolfgang Wahlster
Director

Specifying Agent Interaction Protocols with UML Activity Diagrams

Jürgen Lind
German Research Center for AI (DFKI)
Im Stadtwald B36
D-66123 Saarbrücken, Germany
lind@dfki.de

DFKI-TM-00-01

This work has been supported by a grant from The Federal Ministry of Education, Science, Research, and Technology (FKZ ITW-).

ISSN 0946-0071

Specifying Agent Interaction Protocols with UML Activity Diagrams

Jürgen Lind
German Research Center for AI (DFKI)
Im Stadtwald B36
D-66123 Saarbrücken, Germany
lind@dfki.de

March 23, 2000

Abstract

In this paper, we will demonstrate how the Unified Modeling Language (UML) can be used to describe agent interaction protocols. The approach that is presented in this paper does not propose major enhancements or completely new diagrams but instead it relies on existing UML elements that are part of the standard. This conformity with the base UML is a major advantage of the idea as it prevents a diversification of the UML into different potentially incompatible dialects. The practical use of the method is demonstrated in the specification of a realistic agent interaction protocol.

1 Introduction

Interaction is one of the core concepts of Multiagent Systems as it lays out the foundation for cooperative or competitive behavior among several autonomous agents. Before interaction can take place, however, some technical and conceptual difficulties must be solved. Firstly, the agents must be able to understand each other. This mutual understanding is achieved by relying of accepted formal or informal standards where the de-facto standard of todays agent applications seems to be KQML (FININ AND FRITZSON, 1994), others can be found in (BUSSMANN AND MÜLLER, 1993) or (FIPA, 1996). Secondly, the agents must know which messages they can expect in a particular situation and what the are supposed to do (e.g. sending a reply message) when a certain message arrives (or does not arrive for a given period of time). This part of the interaction process is controlled by *interaction protocols* (or simply protocols).

For an example of an interaction protocol, consider an English auction. There, an auctioneer offers a product at a particular price to a group of bidders. Each of the bidders individually decides to accept that price or to decline the offer. If one of the bidders accepts the current price, the auctioneer raises the price by a fixed rate and asks the group of bidders again if any of them accepts the new price. If this is the case, the price is raised again and the cycle repeats until none of the bidders is willing to pay the current price. Then, the last bidder who accepted the previous price is given the product.

In this example, we can identify the major elements of interaction protocols. Firstly, we can distinguish the participating agents into several groups. In this case, we have two groups: the auctioneer and the bidders. Each group has a set of associated incoming and outgoing messages an internal functions that decide about their next action. We will refer to the set of messages and behaviors that are associated with a group of agents as a *role* that can be played by an agent. Please note that agents are not limited to a single role, e.g. the auctioneer in the previous example can be a bidder in another auction at the same time. The second important aspect of an interaction protocol besides the participating roles is the temporal ordering of function evaluation and the messages that are exchanged. For example, it would not make sense or would be impossible for the bidder to decide on an offer and to decline it before it has even received the offer. Therefore, the interaction protocol determines the flow of control within each role as well as between different roles.

It is precisely the dualism mentioned in the previous paragraph that makes protocol design a difficult task. There are not only role-internal aspects to consider during the design process, but also external effect induced by the other roles. Even worse, there is currently only little Software Engineering support for the design of interaction protocols. A number of protocol specification languages have been proposed ranging from specification languages for low level communication protocols (THE INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 1997), (HOLZMANN, 1991) up to high level specification languages for multiagent applications (BURMEISTER ET AL., 1995), (KOLB, 1995). Up to now, however, none – perhaps except for Estelle – of these languages has gained widespread acceptance.

One reason for this lack of acceptance is probably the fact that the above languages all provide text-based representations for the interaction protocols. This makes it hard, especially for complex protocols, to understand the flow of control within the protocol. An alternative for these text-based languages are therefore graphical languages that make the described protocols more accessible for the reader. One of the currently most popular graphical design languages is the UML (BOOCH ET AL., 1999). The UML combines various ideas from other graphical design languages as well as some original ideas into a coherent framework that allows the Software engineer to specify almost all aspects of a software system. The key term in the last sentence, however, is "almost all" and it is – among others – the field interaction protocols that is not treated adequately by the UML.

In (LIND, 2000) we have used the UML as a design tool and developed a way to describe interaction diagrams using a standard diagram type provided by the UML. Only minor changes to the proposed standard elements of this diagram type were necessary to provide the required expressiveness. Thus, we use UML activity diagrams that are enhanced by a concept that allows the modeling of the message exchange process between the roles within an interaction protocol. The major advantage of this approach is that it does not introduce a completely new diagram type that is potentially difficult to learn and probably even more difficult to establish in the UML community. Instead, we propose to use an existing diagram type whose semantics can be quickly understood by anybody familiar with the standard UML.

2 Related Work

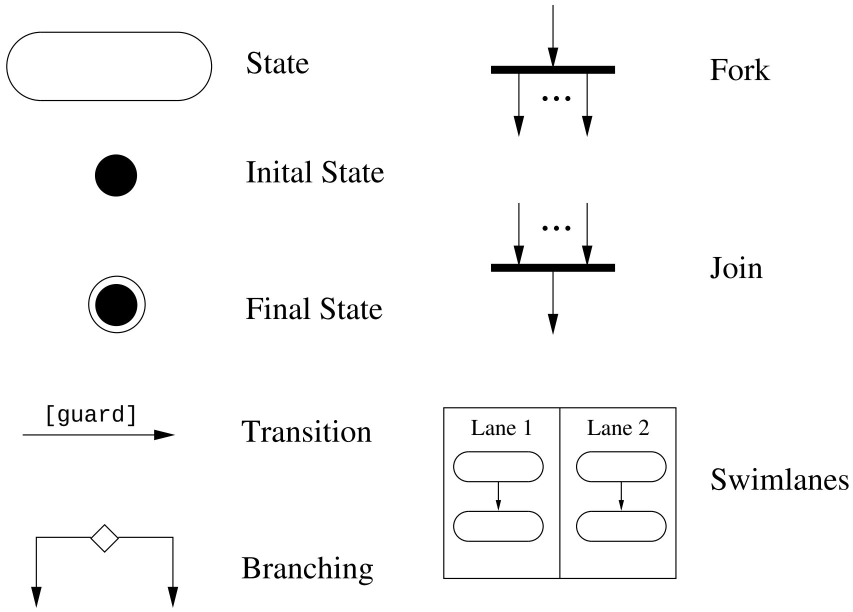
In the previous section, we have already mentioned some protocol specification languages that have been proposed. Estelle (THE INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 1997), for example, is a specification language for service description and system behavior in telecommunications that uses extended finite automata to describe the intended behavior. Extended finite state machines are normal finite state machines plus (typed) variables. The state in the finite state machine has a set of associated variables that can be queried and/or manipulated in the transition specifications. In Estelle, a protocol is a collection of several distinct automata where each automaton can have an arbitrary number of interaction points with other automata. These interaction points are called *channels* and they control the message exchange between different automata. Estelle is a very powerful language that was mainly developed for the specification of low level protocols. It is therefore not directly suitable for the use in multiagent applications.
The Protoz (PHILIPPS AND LIND, 1999) protocol specification environment features a specification language that is related to Estelle (THE INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 1997) and that is based on a similar computational concept. However, due to the focus on multiagent specific aspects, Protoz provides a more accessible interface to protocol design. The main tool of the protocol environment is a compiler that generates Oz code (PROGRAMMING SYSTEMS LAB, 1999) from a given protocol specification, a graphical notation is currently not available. In the Protoz environment, a protocol is defined by a collection of roles where each of these roles is specified as an extended finite state machine. The state machine transitions fire upon incoming messages; messages can stem from other agents or from internal procedures. These internal procedures implement the connection to the application and allow for a uniform modeling of internal and external communication.
The ZEUS development environment (NWANA ET AL., 1999) from British Telecom is a a design method and tool collection for the engineering of dis-

tributed multiagent applications. The ZEUS tools all encompass the direct-manipulation metaphor and allow the designer to use drag-and-drop technology to assemble the application from pre-defined components. The tool-kit allows the designer to specify models for different types of agents, for the organizational structure of agent societies and for negotiation models. The negotiation models are either pre-defined or the can be build by the designer if no appropriate pre-defined model is available for a particular task. In (COLLINS AND NDUMU, 1998) a notation for role models is presented that originates from UML class diagram notation and that contains also elements from UML interaction diagrams (e.g. message sequencing). The ZEUS role models capture structural (static) relationships between roles as well as communicative acts that describe the dynamic aspects of inter-agent communication. The pre-defined role models that are provided by the ZEUS environment include various protocols from the trading domain as well as business processes such as supply chain management.
As part of the AGENT UML in the FIPA standard, (BAUER ET AL., 1999) suggests an extension of the UML by a completely new diagram type called *protocol diagrams*. These diagrams combine elements of UML interaction diagrams and state diagrams to model the roles that can be played by an agent in the course of interacting with other agents. The new diagram type allows for the specification of multiple threads within an interaction protocol and supports protocol nesting and protocol templates based on generic protocol descriptions.

3 UML Activity Diagrams

Activity diagrams in UML models provide a number of structural elements as shown in in Figure 1 to describe algorithms in a flowchart like manner. To this end, each computation is expressed in terms of *states* and the progression through these states. In order to allow for a hierarchical modeling, the UML distinguishes between two classes of states. *Action states* are atomic entities that cannot be decomposed and that relate to atomic statements in a programming language, eg. variable assignment. *Activity states*, on the other hand, represent a collection of atomic states and can thus be decomposed into these atomic states. Furthermore, the execution of an activity can be interrupted between any two subsequent states. In terms of programming languages, actions relate to statements and activities relate to subroutines.
The states of an activity diagram are linked with each other through *transitions* that indicate the control flow within the activity diagram. Each transition can have a *guard* condition that controls the flow of control in that it only allows a transition to fire if the guard condition is true. Because of the basic requirement that each transition must have at least one start and one end point, special states are introduced that represent the beginning and the end of an activity diagram, respectively.

Figure 1 Structural Elements of UML Activity Diagrams

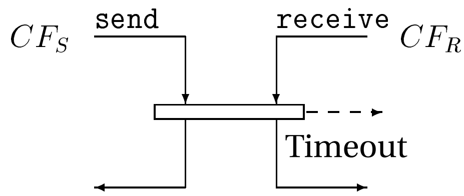


The control flow within an activity diagram is not necessarily linear, otherwise it would be impossible to express anything other then trivial algorithms. Therefore, *branching* elements that represent the decision points within a diagram are provide. Each branching points stands for a boolean decision, i.e. the flow of control can proceed along two different paths.

Many modern programming languages provide some notion for pseudo-parallel program execution within a single operating system process. These light-weight processes – usually referred to as "threads" – can be modeled in UML activity diagrams by using two structural elements. A *fork* operation splits a single thread of execution into two or more threads that are subsequently executed in parallel. Thus, a fork bar has one incoming transition and several outgoing transitions. In order to merge several of these parallel threads into a single thread again, UML activity diagrams provide the *join* element. Thus, a join barrier has several incoming transitions and only a single outgoing transition, it can therefore be used to synchronize several parallel threads of execution. Note that a join barrier waits until *all* incoming threads have arrived at the barrier before proceeding with the single master thread.

Because of the fact that activity diagrams tend to become somewhat confusion with growing in size, UML activity diagrams can contain so-called *swimlanes* that are used to partition an activity diagram into several conceptually related parts. Within an activity diagram, each swimlane must have a unique name and each activity must belong to exactly one swimlane.

Figure 2 Synchronization Point



4 Protocol Specification with Activity Diagrams

In this paper, we propose some slight modifications to the basic elements of UML activity diagrams in order to make them usable to describe agent interaction protocols. First of all, we will extend the idea of swimlanes as a means to distinguish between conceptually related parts of an activity diagram. In our interpretation, these swimlanes are interpreted as physically – as opposed to conceptually – separated flows of control which we will refer to as *Control Flow Spaces* in the rest of this paper. These control flow spaces are linked with each other via explicit communication channels that manage the message exchange between two connected spaces. The message exchange itself is modeled in *synchronization points* that denote the sending and the reception of messages, respectively. The graphical representation of a synchronization point is shown in Figure 2 where CF_S and CF_R denote the the control flow of the sender and the control flow of the receiver, respectively.

Each synchronization point has several incoming transitions out of which exactly one must be labeled with the keyword `send`. The other transitions are the receivers of the respective message. Whenever the control flow of a receiver enters a synchronization point, the receiver suspends until a message has been delivered. This happens whenever the control flow of the sender reaches the synchronization point. After the massage has been delivered, the control flow of the sender and the control flow of the receivers resumes after the synchronization point. In order to prevent the receivers from infinite blocking while waiting for a message that never arrives, an additional *timeout* transition for each receiver can be attached to the synchronization. Whenever the timeout is reached and no message has been delivered, the control flow of the respective receiver resumes at the state pointed to by the timeout transition.

A broader view of activity diagrams in conjuction with agent protocol specification is shown in Figure 3. The round boxes indicate the control flow spaces that are associated with each role within the agent interaction protocol. The control flow of each of these roles is modeled using the structural elements that are provided by standard UML activity diagrams. However, the self-contained control flow spaces are linked via a communication channel that holds one synchronization point that links the activity diagrams of the different roles.

A very important feature of UML diagrams is that they provide a powerful struc-

Figure 3 Extended Activity Diagram

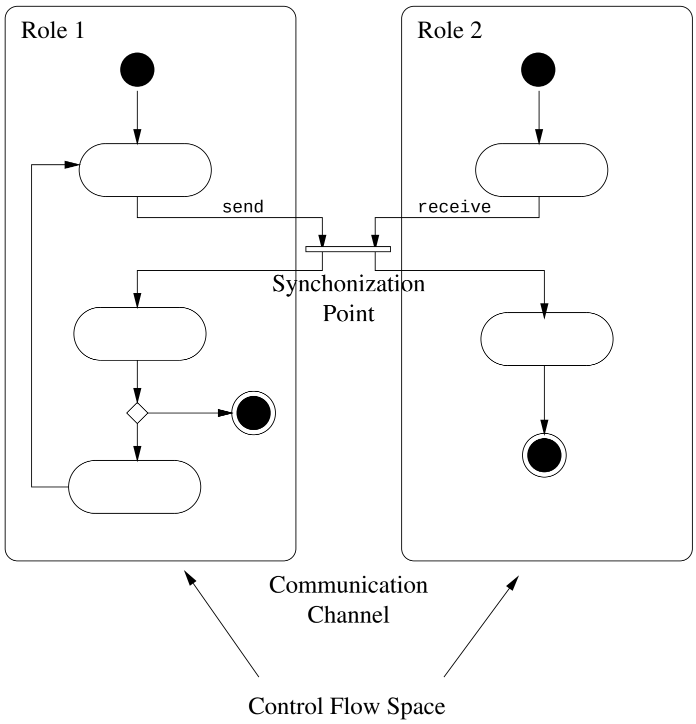
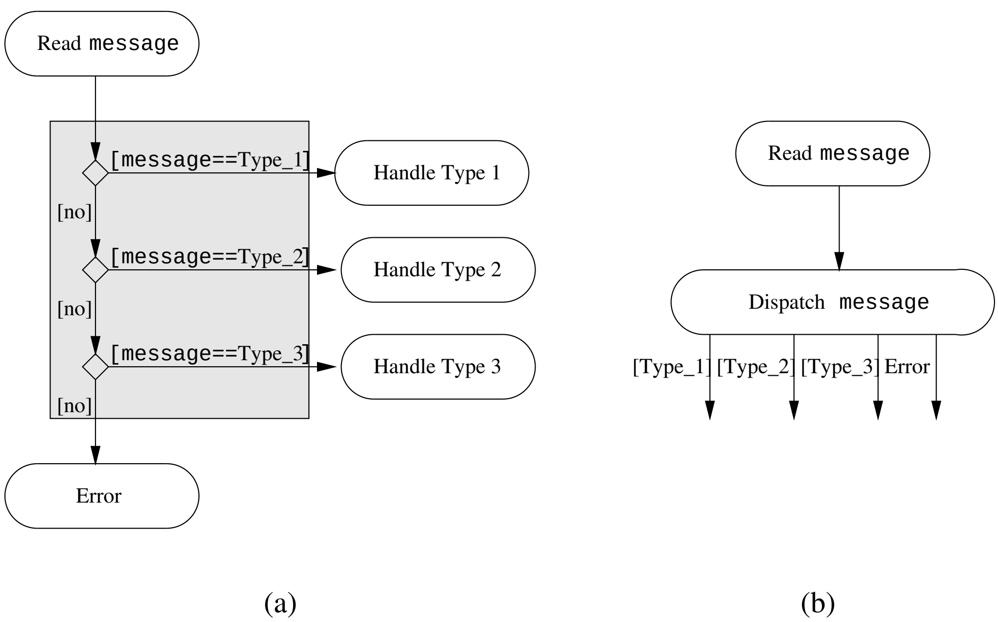


Figure 4 Defining Macros



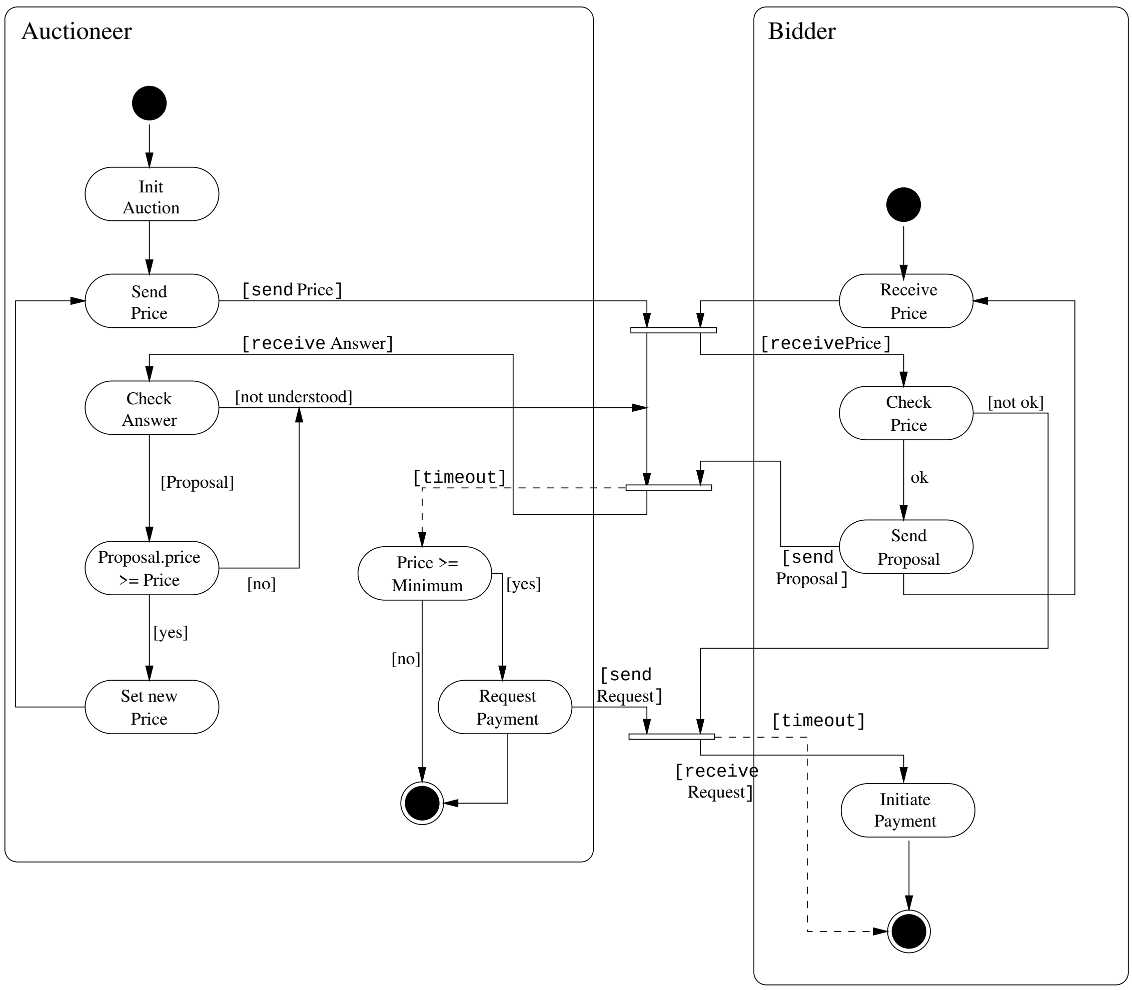
turing mechanism that can be used to make protocol mode readable. Since activity states can represent complete automata, it is straightforward to use them for macro definitions that can be used in interaction protocols. Figure 4 illustrates the idea. Figure 4 (a) shows an activity diagram for dispatching an incoming message according to the message type. Using the UML rule that a state can have several outgoing transitions that are labeled with conditional statements, we can rewrite the shaded part of the original automaton that contains three branching points into a single state as shown in 4 (b)[1]. Collapsing several states into a single macro state has not only the advantage to make a diagram more readable, it is also important that the macro state can be given a speaking name that highlight its purpose. Furthermore, this mechanism can be used to embed protocols into others, allowing for a hierarchical structuring, flexible combination and re-use of protocols.

5 Example

In order to illustrate the use of UML activity diagrams for interaction protocol specification on a realistic example, recall the English Auction that was presented in th introductory section. In Figure 5, we have depicted an interaction protocol that describes the course of actions and message exchanges within the auction more formally.
The first step in the interaction design process is to identify the roles that interact with each other. In the example, we have already identified the *auctioneer* and the *bidder* as the participating roles. Now, we create a control flow space that will later hold the finite automaton that describes the behavior of the agent playing a particular role. It is usually a good idea to develop an initial version of each automaton without considering the other automata, i.e. without switching back and forth between different automata. Thus, for the auctioneer, the auction starts with an initialization of its internal data, e.g. with determining the initial price of the product. Then, the auctioneer sends out a proposal to the bidders and waits for the incoming replies. In order to make the example more realistic, we assume that a bidder can indicate that the proposal was not understood, e.g. because the bidder is not familiar with the ontology used. In that case, the auctioneer simply ignores the message and continuous to wait for further messages. If, on the other hand, the price is accepted by the bidder, the auctioneer raises the price according to a fixed rate and the cycle starts from the beginning. In the offer is not accepted by the bidder, the auctioneer continuous to wait for incoming replies until a fixed timeout. When the timeout has expired and no bidder has accepted the offer, the product is given to the last bidder that has accepted the price (if that price exceeds a previously defined minimal acceptable price). Please note, that the *CheckAnswer* state uses

[1]Note that conditions on the outgoing transitions are abbreviated in the example.

Figure 5 English Auction



the macro mechanism explained earlier to dispatch the incoming messages. Now that the behavior of the auctioneer has been fully specified, we can turn to the bidder role. In the example, the bidder goes into a waiting loop as soon as the protocol execution is started. It leaves this loop when it receives an offer proposed by the auctioneer and checks whether to offered price is acceptable according to its individual goals. If this is the case, the bidder sends out a positive reply and re-iterates the waiting process. If the actual price is not acceptable, the bidder waits for a message from the auctioneer that indicates if the bidder is given the product or nor. Obviously, this can only happen when the bidder has issued a positive reply during the auction. To avoid an infinite blocking of the bidder, a timeout is applied to terminate the waiting process after a finite time. The bidder that receives the positive acknowledgment from the auctioneer, on the other hand, will immediately initiate the payment process to finally receive

the product.
This small example should be sufficient to provide the reader with an impression on how to apply the suggested method to arbitrary agent interaction protocols. The best way to see how the method works in practice is to pick an (preferably easy) protocol from the application domain of interest and then to simply start right away with an iterative modeling process. The value of the diagrams will then quickly become apparent.

6 Conclusion

In this paper, we have demonstrated how UML activity diagrams can be used for the specification of agent interaction protocols. The suggested method uses existing UML concepts as far as possible and requires only little additional elements, therewith making it easy for UML users to understand the interaction protocols without having to learn a completely new type of diagram. The method that was explained in this paper has been used in practical situations and has shown to be a valuable tool for modeling, understanding and communicating agent interaction protocols.

References

BAUER, B., MÜLLER, J. P., AND ODELL, J. (1999). An extension of UML by protocols for multiagent interaction. Submission to the ICMAS2000.

BOOCH, G., RUMBAUGH, J., AND JACOBSON, I. (1999). *The Unified Modeling Language User Guide*. Addison Wesley.

BURMEISTER, B., HADDADI, A., AND SUNDERMEYER, K. (1995). Generic configurable cooperation protocols for multi-agent systems. In CASTELFRANCHI, C. AND MÜLLER, J.-P., editors, *From Reaction to Cognition — 5th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'93)*, volume 957 of *LNAI*, pages 157–171. Springer-Verlag.

BUSSMANN, S. AND MÜLLER, H. J. (1993). A Communication Structure for Cooperating Agents. *Computers and AI*, I.

COLLINS, J. AND NDUMU, D. (1998). The ZEUS Role Modelling Guide. Technical report, BT, Adastral Park, Martlesham Heath.

FININ, T. AND FRITZSON, R. (1994). KQML — a language and protocol for knowledge and information exchange. In *Proceedings of the 13th International Distributed Artificial Intelligence Workshop*, pages 127–136, Seattle, WA, USA.

FIPA (1996). *AgenTalk Reference Manual.* NTT COMMUNICATION SCIENCE LABORATORIES AND ISHIDA LABORATORY, DEPARTMENT OF INFORMATION SCIENCE, KYOTO UNIVERSITY.

HOLZMANN, G. J. (1991). *Design and Validation of Computer Protocols.* Prentice Hall.

KOLB, M. (1995). A cooperation language. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, pages 233–238.

LIND, J. (2000). MASSIVE*: Software Engineering for Multiagent Systems.* PhD thesis, University of the Saarland. to appear.

NWANA, H. S., NDUMU, D. T., LEE, L. C., AND COLLINS, J. C. (1999). ZEUS: A tool-kit for building distributed multi-agent systems. *Applied Artifical Intelligence Journal*, 13(1):129–186.

PHILIPPS, S. AND LIND, J. (1999). Ein System zur Definition und Ausführung von Protokollen für Multi-Agentensystemen. Technical Report RR-99-01, DFKI.

PROGRAMMING SYSTEMS LAB (1999). The mozart programming system. University of the Saarland. http://www.mozart-oz.org.

THE INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (1997). IS-9074 (Information processing systems/Open systems interconnection): Estelle — a formal description technique based on an extended state transition model.

Specifying Agent Interaction Protocols with UML Activity Diagrams

TM-00-01
Technical Memo

Jürgen Lind
German Research Center for AI (DFKI)
Im Stadtwald B36
D-66123 Saarbrücken, Germany
lind@dfki.de