Agent UML: a formalism for specifying multiagent software systems

Bernhard Bauer, Jörg P. Müller, James Odell

Angaben zur Veröffentlichung / Publication details:

Bauer, Bernhard, Jörg P. Müller, and James Odell. 2001. "Agent UML: a formalism for specifying multiagent software systems." *Lecture Notes in Computer Science* 1957: 91–103. https://doi.org/10.1007/3-540-44564-1_6.

Agent UML: A Formalism for Specifying Multiagent Software Systems

Bernhard Bauer*, Jörg P. Müller*, James Odell[+]

* Siemens AG, ZT IK 6, D-81730 München, Germany
bernhard.bauer@mchp.siemens.de
joerg.mueller@mchp.siemens.de

[+] James Odell Associates, 3646 W. Huron River Dr,
Ann Arbor, MI 48103 USA
jodell@compuserve.com

Abstract. In the past, research on agent-oriented software engineering had been widely lacking touch with the world of industrial software development. Recently, a cooperation has been established between the Foundation of Intelligent Physical Agents (FIPA) and the Object Management Group (OMG) aiming to increase acceptance of agent technology in industry by relating to de facto standards (object-oriented software development) and supporting the development environment throughout the full system lifecycle. As a first result of this cooperation, we proposed AGENT UML [1; 20], an extension of the Unified Modeling language (UML), a de facto standard for object-oriented analysis and design. In this paper, we describe the heart of AGENT UML, i.e., mechanisms to model protocols for multiagent interaction. Particular UML extensions described in this paper include protocol diagrams, agent roles, multithreaded lifelines, extended UML message semantics, nested and interleaved protocols, and protocol templates.

1 Introduction

For the past decade, research on agent-oriented software engineering had suffered from a lack of touch with the world of industrial software development. Recently, it has been recognized that the use of software agents is unlikely likely to gain wide acceptance in industry unless it relates to de facto standards (object-oriented software development) and supports the development environment throughout the full system lifecycle.

Successfully bringing agent technology to market requires techniques that reduce the perceived risk inherent in any new technology, by presenting the new technology as an incremental extension of known and trusted methods, and by providing explicit engineering tools to support proven methods of technology deployment.

Applied to agents, these insights imply an approach that:

- introduces agents as an extension of active objects*: an agent is an object that can say "go"* (flexible autonomy as the ability to initiate action without external invocation) *and "no"* (flexible autonomy as the ability to refuse or modify an external request)[1];
- promotes the use of standard representations for methods and tools to support the analysis, specification, and design of agent software.

The former aspect of our approach leads us to focus on fairly fine-grained agents. More sophisticated capabilities can also be added where needed, such as mobility, mechanisms for representing and reasoning about knowledge, and explicit modeling of other agents. Such capabilities are extensions to our basic agents—we do not consider them diagnostic of agenthood.

To achieve the latter, three important characteristics of industrial software development should be addressed:

1. The scope of industrial software projects is much larger than typical academic research efforts, involving many more people across a longer period of time. Thus, communication is essential;
2. The skills of developers are focused more on development methodology than on tracking the latest agent techniques. Thus, codifying best practice is essential;
3. Industrial projects have clear success criteria. Thus, traceability between initial requirements and the final deliverable is essential.

The Unified Modeling Language (UML) is gaining wide acceptance for the representation of engineering artifacts in object-oriented software. Our view of agents as the next step beyond objects leads us to explore extensions to UML and idioms within UML to accommodate the distinctive requirements of agents. To pursue this objective, recently a cooperation has been established between the Foundation of Intelligent Physical Agents (FIPA) [7] and the Object Management Group (OMG). As a first result of this cooperation, we analyzed the requirements for such an endeavor and proposed the framework of AGENT UML [1].

In this paper, we describe a core part within AGENT UML, i.e., mechanisms to model protocols for multiagent interaction. This is achieved by introducing a new class of diagrams into UML: *protocol diagrams.* Protocol diagrams extend UML state and sequence diagrams in various ways. Particular extensions in this context include agent roles, multithreaded lifelines, extended message semantics, parameterized nested protocols, and protocol templates.

The model described in this paper has been proposed and accepted for inclusion into the upcoming FIPA'99 standard. It was invited to be submitted as a response to a Request for Information (RFI) issued by the OMG Analysis and Design Task Force for the next release of UML (v2.0).

The paper is structured as follows: In Section 2, we survey approaches to software specification, including UML. Section 3 specifies the extension of UML by multiagent interaction protocols. Section 4 discusses further details of the extensions. Section 5

[1] See [12], [16] for more comprehensive definitions of agents.

attempts a preliminary evaluation of the concepts, summarizes the results of the paper and discusses future research topics.

2 Software Specification Techniques

AGENT UML is an attempt to bring together research on agent-based software methodologies and emerging standards for object-oriented software development.

2.1 Methodologies for agent -based software development

There is a considerable interest in the agent R&D community in methods and tools for analyzing and designing complex agent-based software systems, including various approaches to formal specification (see [11] for a survey). Since 1996, agent-based software engineering has been a focus of the ATAL workshop series and was the main topic for MAAMAW'99 [9].

Various researchers have reported on methodologies for agent design, touching on representational mechanisms as they support the methodology. Our own report at [22] emphasizes methodology, as does Kinny's work on modeling techniques for BDI agents [14; 15]. The close parallel that we observe between design mechanisms for agents and for objects is shared by a number of authors, for example, [4; 6].

The GAIA methodology [25] includes specific recommendations for notation in support of the high-level summary of a protocol as an atomic unit, a notation that is reflected in our recommendations. The extensive program underway at the Free University of Amsterdam on compositional methodologies for requirements [10], design [3], and verification [13] uses graphical representations with similarities to UML collaboration diagrams, as well as linear (formulaic) notations better suited to alignment with the UML meta-model than with the graphical mechanisms that are our focus.

Our discussion of the compositionality of protocols is anticipated in the work of Burmeister et al. [5]. Dooley graphs [21] facilitate the identification of the *character* that results from an agent playing a specific role (as distinct from the same agent playing a different role).

The wide range of activity in this area is a sign of the increasing impact of agent-based systems, since the demand for methodologies and artifacts reflects the growing commercial importance of agent technology. Our objective is not to compete with any of these efforts, but rather to extend and apply a widely accepted modeling and representational formalism (UML) in a way that harnesses their insights and makes it useful in communicating across a wide range of research groups and development methodologies.

2.2 UML

The Unified Modeling Language (UML) [17] unifies and formalizes the methods of many object-oriented approaches, including Booch, Rumbaugh (OMT), Jacobson, and Odell. It supports the following kinds of models:

- *use cases:* the specification of actions that a system or class can perform by interacting with outside actors. They are commonly used to describe how a customer communicates with a software product.
- *static models:* describe the static semantics of data and messages in a conceptual and implementational way (e.g., class and package diagrams).
- *dynamic models:* include interaction diagrams (i.e., sequence and collaboration diagrams), state charts, and activity diagrams.
- *implementation models:* describe the component distribution on different platforms (e.g., component models and deployment diagrams).
- *object constraint language* (*OCL*): a simple formal language to express more semantics within an UML specification. It can be used to define constraints on the model, invariant, pre- and post-conditions of operations and navigation paths within an object net.

In this paper, we propose agent-based extensions to three following UML representations: packages, templates, and sequence diagrams. This results in a new diagram type, called *protocol diagram*, which we developed within FIPA 1999, and which will be considered for inclusion into UML version 2.0 by OMG. The UML model semantics are represented by a meta-model the structure of which is also formally defined by OCL syntax. Extensions to this meta-model and its constraint language are not addressed by this paper.

2.3 A rationale for AGENT UML

In a previous paper, we have argued that UML provides an insufficient basis for modeling agents and agent-based systems [1], see also [20]. 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.

To employ agent-based programming, a specification technique must support the whole software engineering process—from planning, through analysis and design, and finally to system construction, transition, and maintenance.

A proposal for a full life-cycle specification of agent-based system development is beyond the scope for this paper. Both FIPA and the OMG Agent Work Group are exploring and recommending extensions to UML [1; 18]. In this paper, we will focus on a subset of an agent-based UML extension for the specification of *agent interaction protocols (AIP)*.

This subset was chosen because AIPs are complex enough to illustrate the nontrivial use of and are used commonly enough to make this subset of AGENT UML useful to other researchers. AIPs are a specific class of software design patterns in that they describe problems that occur frequently in multiagent systems and then describe the core of a reusable solution to that problem [8, p. 2].

The definition of interaction protocols is part of the specification of the dynamical model of an agent system. In UML, this model is captured by interaction diagrams, state diagrams and activity diagrams.

- *Interaction diagrams*, i.e. sequence diagrams and collaboration diagrams are used to define the behavior of groups of objects. Usually, one interaction diagram captures the behavior of one use case. These diagrams are mainly used to define basic interactions between objects at the level of method invocation; they are not well-suited for describing the types of complex social interaction as they occur in multiagent systems.
- *State diagrams* are used to model the behavior of a complete system. They define all possible states an object can reach and how an object's state changes depending on messages sent to the object. They are well suited for defining the behavior of one single object in different use cases. However, they are not appropriate to describe the behavior of a group of cooperating objects.
- *Activity diagrams* are used to define courses of events / actions for several objects and use cases. The work reported in this paper does not suggest modifications of activity diagrams.

3 AGENT UML Interaction Protocols

The definition of an agent interaction protocol (AIP) describes

- a communication pattern, with
  - an allowed sequence of messages between agents having different roles,
  - constraints on the content of the messages, and
- a semantics that is consistent with the communicative acts (CAs) within a communication pattern.

Messages must satisfy standardized communicative (speech) acts which define the type and the content of the messages (e.g. the FIPA agent communication language (ACL), or KQML). Protocols constrain the parameters of message exchange, e.g., their order or types, according to relationships between the agents or the intention of the communication.

The new diagram type introduced in this paper are *Protocol Diagrams*. Since interaction protocols, i.e. the definition of cooperation between software agents, define the exact behavior of a group of cooperating agents, we combine sequence diagrams with the notation of state diagrams for the specification of interaction protocols.

As an introductory example let us consider a surplus ticket market for flights. The example is taken from the PTA application (see Section 5). The auctioning of such tickets can be performed using, e.g. the FIPA English-Auction Protocol as shown in Figure 1. The auctioneer initially proposes a price lower than the expected market price, and then gradually raises the price. The auctioneer informs all participants that the auction has started (represented by the messages *inform(start-auction, departure, arrival)* in Figure 1) and announces the details of the flight. Each time a new price is announced (represented by *cfp(intial-price)* and *cfp(new-price)*), the auctioneer waits until a given *deadline* to see if any participants signal their willingness to pay the proposed price (*propose*) for the ticket. If a participant does not understand the ontology or syntax of the *cfp* it replies a *not-understood* communicative act. The diamond symbol with the 'x' in it indicates a decision resulting in zero or more communications being sent (see Section 4.2)). As soon as one participant indicates that it will accept the price, the auctioneer issues a new call for bids (*cfp(new-price)*) with an incremented price. The auction continues until no auction participants are prepared to pay the proposed price, at which point the auction ends. If the last price accepted by a buyer exceeds the auctioneer's reservation price, the ticket is sold to that participant for the agreed price (otherwise the auction fails). The participants are *informed* about the end of the auction and the buyer is *requested* to pay the price for the ticket.

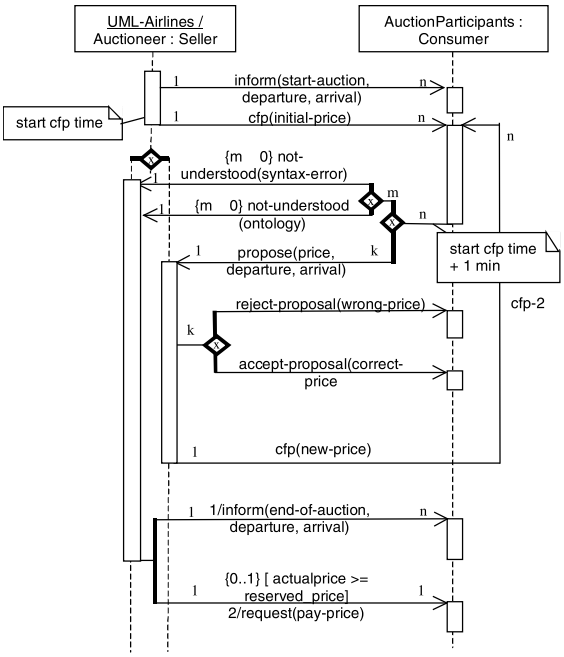

Figure 1. English-Auction protocol for surplus flight tickets

The diagram in Figure 1 provided a basic specification for a English Auction protocol. In [20] we have shown how such a specification can be gradually refined until the problem has been specified adequately to develop or generate code. Each level can express *intra-agent* or *inter-agent* activity.

4 Elements of Protocol diagrams

In the last chapter we gave an example how interaction protocols can be specified using the UML extension. In this chapter we will have a closer look at the different extensions.

4.1 Agent roles

In UML, *role* is an instance focused term. In the framework of agent oriented programming by *agent-role* a set of agents satisfying distinguished properties, interfaces, service descriptions or having a distinguished behavior are meant.

UML distinguishes between *multiple classification* (e.g., a retailer agent acts as a buyer *and* a seller agent at the same time), and *dynamic classification*, where an agent can change its classification during its existence.

Agents can perform various roles within one interaction protocol. E.g., in an auction between an airline and potential ticket buyers, the airline has the role of a seller and the participants have the role of buyers. But at the same time, a buyer in this auction can act as a seller in another auction. I.e., agents satisfying a distinguished role can support multiple classification and dynamic classification.

Therefore, the implementation of an agent can satisfy different roles. An agent role describes two variations, which can apply within a protocol definition. A protocol can be defined at the level of concrete agent instances or for a set of agents satisfying a distinguished role and/or class. An agent satisfying a distinguished agent role and class is called *agent of a given agent role* and *class*, respectively. The general form of describing agent roles in AGENT UML is

`instance-1 ... instance-n / role-1 ... role-m : class`

denoting a distinguished set of agent instances *instance-1*,..., *instance-n* satisfying the agent roles *role-1*,..., *role-m* with n, m 0 and *class* it belongs to. Instances, roles or class can be omitted, in the case that the instances are omitted the roles and class are not underlined. In Fig. 1 the auctioneer is a concrete instance of an agent named *UML-Airlines* playing the role of an *Auctioneer* being of class *Seller*. The participants of the auctions are agents of role *AuctionParticipants* which are familiar with auctions and of class *Consumer*.

4.2 Agent Lifelines and Threads of Interaction

The agent lifeline in protocol diagrams defines the time period during which an agent exists, represented by dotted vertical lines. The lifeline starts when the agent of a given agent role is created and ends when it is destroyed. For example, a user agent is created when a user logs on to the system and the user agent is destroyed when the user logs off. The lifeline may split up into two or more lifelines to show AND and OR parallelism and decisions, corresponding to branches in the message flow. Lifelines may merge at some subsequent point. In Figure 1 the lifeline splits in order to describe the different reaction of the agent depending on the incoming messages, here to handle

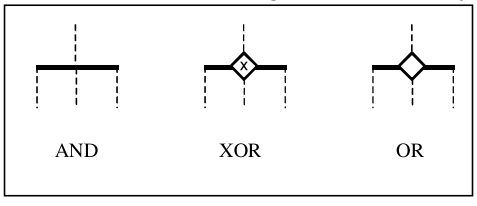

Figure 2. Connector types

proposals and not-understoods respectively. Figure 2 shows the graphical representations for the logical connectors AND, XOR, and OR.

The XOR can abbreviated by interrupting the *threads of interaction* as shown also in Figure 3 (right). The thread of interaction, i.e. the processing of incoming messages, is split up into different threads of interaction, since the behavior of an agent role depends on the incoming message. The lifeline of an agent role is split accordingly and the thread of interaction defines the reaction to different kinds of received messages.

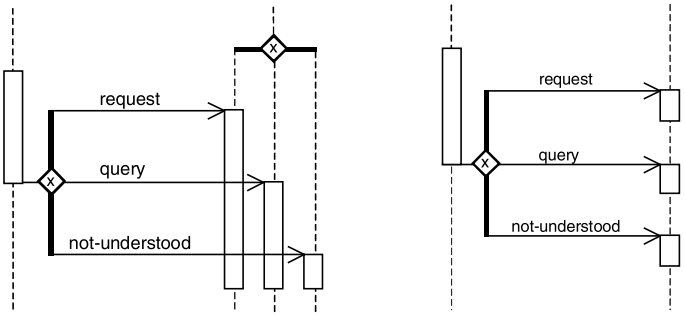

Figure 3. Full and abbreviated notation of XOR connection

The thread of interaction shows the period during which an agent role is performing some task as a reaction to an incoming message. It only represents the duration of the action, but not the control relationship between the sender of the message and its receiver. A thread of interaction is always associated with the lifeline of an agent role. Supporting concurrent threads of interaction is another recommended extension to UML .

4.3 Nested and Interleaved Protocols

Because protocols can be codified as recognizable patterns of agent interaction, they become reusable modules of processing that can be treated as first-class notions. For example, Figure 4 depicts two kinds of protocol patterns. The left part defines a nested protocol, i.e. a protocol within another protocol, and the right part defines an interleaved protocol, e.g. if the participant of the auction requests some information about his/her bank account before bidding. Additionally nested protocols are used for the definition of repetition of a nested protocol according to guards and constraints. The semantics of a nested protocol is the semantics of the protocol. If the nested protocol is marked with some guard then the semantics of the nested protocol is the semantics of the protocol under the assumption that the guard evaluates to true, otherwise the semantics is the semantics of an empty protocol, i.e. nothing is specified.

If the nested protocol is marked with some constraints the nested protocol is repeated as long as the constraints evaluate to true. In addition to the constraint-condition used in UML the description n..m, denoting that the nested protocol is

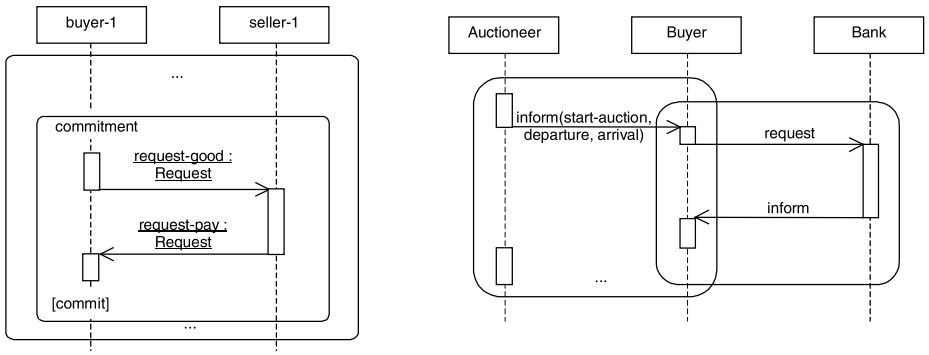

Figure 4. nested protocol and interleaved protocol

repeated n up to m times with n , m { * }, the asterisk denotes arbitrary times, is used as a constraint condition.

4.4 Extended Semantics of UML Messages

The main purpose of protocols is the definition of communicative patterns, i.e., patterns of messages sent from one agent role to another. This is described by various parameters, such as different cardinalities, depending on some constraints, or using AND / OR parallelism and decisions.

Sending a communicative act from one agent to another that conveys information and entails the sender's expectation that the receiver react according to the semantics of the communicative act. The specification of the protocol says nothing about how this reaction is implemented.

An asynchronous message is drawn as ⟶[2]. It shows the sending of the message without yielding control. A synchronous message is shown as ⟶. It shows the yielding of the thread of control (wait semantics), i.e. the agent role waits until an answer message is received and nothing else can be processed. Normally message arrows are drawn horizontally. This indicates the duration required to send the message is "atomic", i.e. it is brief compared to the granularity of the interaction and that nothing else can "happen" during the message transmission. If the messages requires some time to arrive, e.g. for mobile communication, during which something else can occur then the message arrow is shown as ↘. The repetition of a part of a protocol is represented by an arrow or one of its variations usually marked with some guards or constraints ending at a thread of interaction which is, according to the time axis, before or after the actual time point, like the *cfp(new-price)* in Fig. 1. This repetition is another extension to UML messages

Each arrow is labeled with a message label[3]. The message label consists of the following parts, which can also be found in Fig. 1. The communicative act which is sent from one agent to another, like *cfp(initial-price)* with a list of arguments representing additional information for the characterization of the communicative act. The cardinality defines that a message is sent from one agent to n agents, like in the *cfp(new-price)* case. Constraints and guards, like {m >= 0 } and [actualprice >= reservedprice] respectively, can be added to define the condition when a message is sent. In addition to the constraint-condition used in UML the description n..m, denoting that the message is repeated n up to m times with n , m { * }, the asterisk denotes arbitrary times, is used as a constraint condition.

Messages may be sent in parallel or exactly one message out of a set of different messages should be sent. E.g., in Figure 1, exclusive sending is denoted as for the *reject-proposal* and *accept-proposal*. *inform(end-of-auction, departure, arrival)* and *request(pay-price)* are sent in parallel but *inform* is sent first (*1/inform-2*) and the

---

[2] Notation of UML v1.3.

[3] The message label is a special case of the message label presented in the UML 1.1 specification section 8.9.2.

request is sent as the second message (*2/request*). The request is also sent zero or one time {0..1}, depending on whether the reservation price was reached or not.

4.5 Input and Output Parameters for Nested Protocols

Nested Protocols can be defined either within or outside a protocol diagram where it is used or outside another protocol diagram. The input parameters of nested protocols are threads of interaction which are carried on in the nested protocol and messages which are received from other protocols.

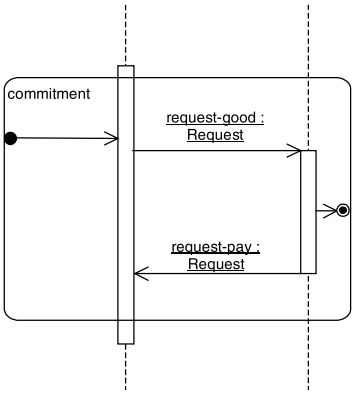

Figure 5. Input/output of nested protocols

The output parameters are the threads of interaction which are started within the nested protocol and are carried on outside the nested protocol and the messages which are sent from inside the nested protocol to agent roles not involved in the actual nested protocol. A message or thread of interaction ending at an input or starting at an output parameter of a nested protocol describes the connection of a whole protocol diagram with the embedded nested protocol.

The input and output parameters for the threads of interaction of a nested protocol are shown as in Figure 4 which is drawn over the top line and bottom line of the nested protocol rectangle, respectively. The input and output message parameters are shown as ⟶ and ⟶◉, respectively.

The message arrows can be marked like usual messages. In this context the predecessor denotes the number of the input / output parameter. The input / output thread of interaction can be marked with natural numbers to define the exact number of the parameter.

4.6 Protocol Templates

The purpose of protocol templates is to create reausable patterns for useful protocol instances. E.g., Figure 6 shows a template for the FIPA-English-Auction Protocol from Figure 1. It introduces two new concepts represented at the top

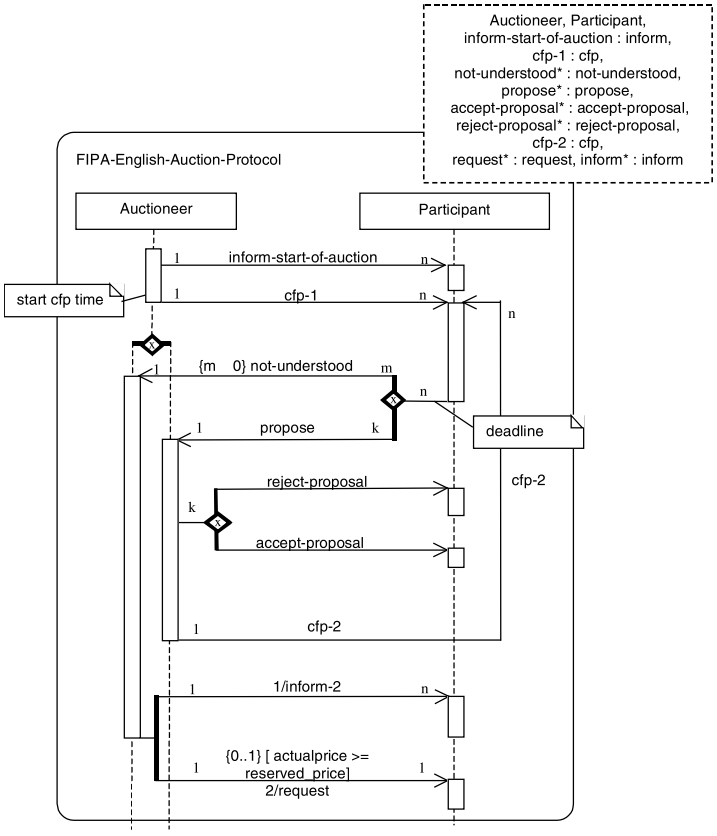

Figure 6. A generic AIP expressed as a template

of the sequence chart. First, the protocol as a whole is treated as an entity in its own right. The protocol can be treated as a pattern that can be customized for other problem domains. The dashed box at the upper right-hand corner declares this pattern as a *template* specification that identifies unbound entities (formal parameters) within the package which need to be bound by actual parameters when instantiating the package. A parameterized protocol 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 a protocol. Communicative acts in the formal parameter list can be marked with an asterisk, denoting different kinds of messages which can alternatively be sent in this context. This template can be instantiated for a special purpose as shown in Figure 1[4]. Figure 7 applies the FIPA English Auction Protocol to a particular scenario involving a specific auctioneer *UML-Airlines* of role *Auctioneer* and Class *Seller* and *AuctionParticipants* of Class *Consumer*. Finally, a specific deadline has been supplied for a response by the seller. In UML terminology, the AIP package serves as a *template*. A template is a parameterized model element whose parameters are bound at model time (i.e., when the new customized model is produced).

FIPA-English-Auction-Protocol <
UML-Airlines / Auctioneer : Seller, AuctionParticipants : Consumer
start cfp time + 1 min
inform(start-auction, departure, arrival),
cfp(initial-price),
not-understood(syntax-error), not-understood(ontology),
propose(pay-price),
reject-proposal(wrong-price), accept-proposal(correct-price),
cfp(increased-price),
inform(end-of-action), request(pay-price, fetch-car)

Figure 7. Instantiation of a template

Wooldridge et al suggest a similar form of definition with their *protocol definitions* [25]. Here, they define packaged templates as “a pattern of interaction that has been formally defined and abstracted away from any particular sequence of execution steps." In contrast to their notation, we suggest a graphical approach that more closely resembles UML, while expressing the same semantics.

5 Evaluation and Conclusion

The artifacts for agent-oriented analysis and design were developed and evaluated in the German research project MOTIV-PTA (Personal Travel Assistant) [2, 23], aiming at providing an agent-based infrastructure for travel assistance in Germany (see www.motiv.de). MOTIV-PTA will run from 1996 to 2000. IT is a large-scale project involving approx. 10 industrial partners, including Siemens, BMW, IBM, DaimlerChrysler, debis, Opel, Bosch, and VW. The core of MOTIV-PTA is a multiagent system to wrap a variety of information services, ranging from multimodal route planning, traffic control information, parking space allocation, hotel reservation, ticket booking and purchasing, meeting scheduling, and entertainment.

From the end user's perspective, the goal is to provide a personal travel assistant, i.e., a software agent that uses information about the users' schedule and preferences in order to assist them in travel, including preparation as well as on-trip support. This requires providing ubiquitous access to assistant functions for the user, in the office, at

[4] This template format is not currently UML-compliant but is a recommendation for future UML extensions.

home, and while on the trip, using PCs, notebooks, information terminals, PDAs, and mobile phones.

From developing PTA (and other projects with corporate partners within Siemens) the requirements for artifacts to support the analysis and design became clear, and the material described in this paper has been developed incrementally, driven by these requirements. So far no empirical tests have been carried out to evaluate the benefits of the AGENT UML framework. However, from our project experience so far, we see two concrete advantages of these extensions: Firstly, they make it easier for users who are familiar with object-oriented software development but new to developing agent systems to understand what multiagent systems are about, and to understand the principles of looking at a system as a society of agents rather than a distributed collection of objects. Secondly, our estimate is that the time spent for design can be reduced by a minor amount, which grows with the number of agent-based projects. However, we expect that as soon as components are provided to support the implementation based on AGENT UML specifications, this will widely enhance the benefit.

Areas of future research include aspects such as

- description of mobility, planning, learning, scenarios, agent societies, ontologies and knowledge
- development of patterns and frameworks
- consideration of events
- real-time-constraints
- support for different agent communication languages and content languages

At the moment we plan to extend the presented framework towards inclusion of these topics. Moreover a project is on the way to refine the specification technique and generate code from such a specification for different agent platforms, e.g. for the MECCA system [2], based on a formal semantics of AGENT UML which is currently being developed.

References

[1] B. Bauer. *Extending UML for the Specification of Interaction Protocols*. submission for the 6th Call for Proposal of FIPA and revised version part of FIPA 99, 1999.

[2] B. Bauer, M. Berger: Agent-Based Personal Travel Assistance, submitted to MAMA 2000, 2000.

[3] F. M. T. Brazier, C. M. Jonkers, and J. Treur. *Principles of Compositional Multi-Agent System Development*. Proceedings 15th IFIP World Computer Congress, WCC'98, Conference on Information Technology and Knowledge Systems, IT&KNOWS'98, pages 347-360, Chapman and Hall, 1998.

[4] J. Bryson, and B. McGonigle. *Intelligent Agents IV: Agent Theories, Architectures, and Languages*. Proceedings ATAL 98, ed., Springer, 1998

[5] B. Burmeister, A. Haddadi, and K. Sundermeyer. *Generic, Configurable, Cooperation Protocols for Multi-Agent Systems*. Proceedings Fifth European Workshop on Modelling

Autonomous Agents in a Multi-Agent World, MAAMAW'93, pages 157-171, Springer, 1993.

[6] B. Burmeister. *Models and Methodology for Agent-Oriented Analysis and Design*. ed., 1996.

[7] http://www.fipa.org

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, 1997.

[9] F. J. Garijo, and M. Boman. *Multi-Agent System Engineering.* Proceedings of MAAMAW'99. Springer, ed., 1999.

[10] D. E. Herlea, C. M. Jonker, J. Treur, and N. J. E. Wijngaards. *Specification of Behavioural Requirements within Compositional Multi-Agent System Design.* Proceedings of Ninth European Workshop on Modelling Autonomous Agents in a Multi-Agent World, pages 8-27, Springer, 1999.

[11] C. A. Iglesias, M. Garijo, and J. C. González. *A Survey of Agent-Oriented Methodologies*. Proceedings of Fifth International Workshop on Agent Theories, Architectures, and Languages, pages 185-198, University Pierre et Marie Curie, 1998.

[12] N. R. Jennings, K. Sycara, and M.J.Wooldridge. *A Roadmap of Agent Research and Development.* Journal of Autonomous Agents and Multi-Agent Systems. 1(1), pages 7-36. July 1998.

[13] C. M. Jonker, and J. Treur. *Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness*. Proceedings of International Workshop on Compositionality (COMPOS'97), Springer, 1997.

[14] D. Kinny, and M. Georgeff. *Modelling and Design of Multi-Agent Systems*. Intelligent Agents III, Springer, 1996.

[15] D. Kinny, M. Georgeff, and A. Rao. *A Methodology and Modelling Technique for Systems of BDI Agents.* 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)., pages 56-71. Springer, 1996.

[16] J. P. Müller. The Design of Autonomous Agents : A Layered Approach, volume 1177 of Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1997.

[17] J. Odell and M. Fowler. *Advanced object-oriented analysis and design using UML.* SIGS Books / Cambridge University Press, 1998.

[18] J. Odell. *Agent Technology*, green paper, produced by the OMG Agent Working Group, ed., 1999.

[20] J. Odell, H. v. D. Paranuk, B. Bauer: *Representing Agent Interaction Protocols in UML*, in this volume.

[21] H. V. D. Parunak. *Visualizing Agent Conversations: Using Enhanced Dooley Graphs for Agent Design and Analysis*. Proceedings of Second International Conference on Multi-Agent Systems, pages 275-282, 1996.

[22] H. V. D. Parunak, and J. Odell. *Engineering Artifacts for Multi-Agent Systems*, ERIM CEC, 1999.

[23] Steiner D. MoTiV-PTA: Personal Travel Assistance for Germany, in Proceedings 4th World Congress on Intelligent Transport Systems. Berlin. Germany. October 21-24, 1997.

[25] M. Wooldridge, N. R. Jennings and D. Kinny. *The Gaia Methodology for Agent-Oriented Analysis and Design*. International Journal of Autonomous Agents and Multi-Agent Systems, 3:Forthcoming, 2000.