UML for Behavior-Oriented Multi-Agent Simulations

Christoph Oechslein, Franziska Klügl, Rainer Herrler, and Frank Puppe

Department for Artificial Intelligence,
University of Würzburg
Am Hubland, 97074 Würzburg

{oechslein,kluegl,herrler,puppe}@informatik.uni-wuerzburg.de

Abstract Developing multi-agent simulations seems to be rather straight forward, as active entities in the original correspond to active agents in the model. Thus plausible behaviors can be produced rather easily. However, for real world applications they must satisfy some requirements concerning verification, validation and reproducibility. Using a standard framework for designing a multi-agent model one can gain further advantages like fast learnability, wide understandability and possible transfer. In this paper we show how UML can be used to specify behavior-oriented multi-agent models. Therefore we focus on activity graphs and the representation of different forms of interactions in these graphs.

1 Motivation

A multi-agent model captures the behavior of a system on the level of the active entities and maps them onto simulated active entities. Thus, on a rather coarse grain level of analysis, it is far more intuitive to describe the model in terms of agents and roles, using goals or activities, than to abstract it into a set of formulas. However, for executable simulations the concrete refinement and implementation exhibits properties that are rather difficult to deal with. The step between the model concept and the running experiment is by no way trivial:

- The modeler has to bridge the gap between top-down analysis and bottom-up implementation.
- Concurrent interactions form a central part of multi-agent systems. However, their concrete implementation is rather tricky.
- Complex agent behavior in sophisticated environments leads to rather large models that are effortful to design and implement in a consistent way.
- A huge amount of parameter has to be fixed for an executable simulation.

The consequence is that tools and frameworks are essential for the applicability of multi-agent simulation as a method for analyzing systems. The ideal framework for modeling would be easy to learn, generates understandable representation, provides all necessary primitives and constructors, and scales for concrete domains. In addition to these methodological points, the representation of a model

based on a framework should be clear and understandable to a wide group of people working in the same domain. This leads to the overall hope that models in multi-agent simulation can also satisfy the strong requirements concerning verification and validation that traditional simulation models are exposed to.

A prominent example for an already established framework is UML ("Unified Modeling Language" [1]). Besides multiple diagram types for the specification of structure and dynamics, UML provides with OCL ("Object Constraint Language") a formal language for attaching additional information, like invariants, constraints, etc. to the graphical specification.

In the following we therefore want to introduce behavior-oriented multi-agent models in the next section. After a short survey of existing frameworks in the area of agent-oriented simulation and software-engineering we are presenting the application of UML for our problem. This paper ends with a conclusion and short sketch of further works.

2 Behavior-Oriented Multi-Agent Simulation

A multi-agent model in general can be seen as a simulated multi-agent system that exists in a simulated environment [2]. This is a natural form of modeling, especially for societies, as active entities in the original system are also interpreted as active ones in the model.

Existing multi-agent models focus on different aspects of a society from the development of dependence networks based on agents beliefs to a huge amount of massively interacting simple entities. Instead of trying to develop a plausible framework for human reasoning in interacting groups or organizations, we restrict ourselves to models that are simple enough to be validate-able, but on the other side capture rather sophisticated behaviors at the agent level. We call these multi-agent models *behavior-oriented* as they do not focus on the internal reasoning processes based on beliefs, goals or desires, but on the agents behavior and the effects of their actions and interactions. This focus is motivated by our primary application domain: the simulation of social insects; However, several other applications show the broad applicability of our approach.

Behavior-oriented multi-agent models follow two main concepts that help to structure the complete multi-agent simulation supporting extensive models with non-trivial behaviors and a huge amount of entities: The agent body as a set of sorted state variables and rule-based behavior representation that is augmented by structures like primitive and abstract activities.

Associating task fulfilment with activities and incorporating organizational concepts like roles additionally facilitates the development of complex society models. Although these concepts are already implemented into a modeling and simulation environment, called SeSAm ("Shell for Simulated Multi-Agent Systems" [3]), this framework is based on a proprietary specification language. This leads to the problem that our work still lacks a more generally applicable method for developing multi-agent simulations. Using a standard framework will have the

additional advantage of generating models that are more likely to be reproducible by a modeler familiar with that standard.

Our overall goal is hereby to support the construction of models that satisfy the high standards of quality – concerning validation and verification – in standard simulation models. Using a standard framework for the step between concept and implementation is a major part of a viable solution.

3 Related Work: Modeling Frameworks and Formalisms

There are already lots of frameworks, tools, and languages for multi agent simulation. Approaches range from schemata and systems originating in the simulation area to languages for modeling multi-agent systems developed in Distributed Artificial Intelligence (for a more complete survey, see [2]).

3.1 Existing Frameworks Developed in the Simulation Area

Multi-agent simulation can be seen as an enhancement of standard individual oriented simulation: Agents can be taken as equipped with more sophisticated behavior in comparison to the individuals in process-oriented models or cellular automatons. In the line of this interpretation the approaches from agent-oriented simulation tackle more the management of the internal processes and internal agent structures. Frameworks like AgedDEVS ("agent-oriented DEVS" [4]) or PECS ("Physis, Emotion, Cognition, Social Status" [5]) provide schemata for the internal state structure of an agent. They are comparable to "traditional" agent architectures [6], but refer to sets of state variables. Another level of support can be found in modeling and simulation environments: e.g. in AgentSheets [7] (*www.agentsheets.com*) and its visual programming or in Swarm [8] (*www.swarm.org*) and its focus on animation and evaluation tools. The latter tackle just the concrete implementation problem and not the gap between concept and implementation.

3.2 From Agent-Oriented Software Engineering

Frameworks for building multi-agent systems are also a central issue in the area of agent-oriented software-engineering. Existing schemata provide support on different levels and for different stages of development. Methods like GAIA [9] or MAS-CommonKADS [10] concentrate on the analysis and design task by providing means for capturing views onto and models of the agent system in a top-down manner. The actual implementation is not tackled; the user is directed to use standard frameworks, like UML.

On the other hand, declarative languages and schemata for the specification of multi-agent systems exist. These start bottom-up from the agents and their behavior. If these specifications are executable – like in ConcurrentMetateM [11] – these languages can be used for simulation as well. In the area of social simulation, an analogous parallel rule-interpreting system, SDML ("Strictly Declarative Modeling Language" [12]), was developed. However, the problem is that

these frameworks operate just on the complete agent level – there is no support for structuring actions and interactions. The behavior of the agents itself is formulated using an unstructured set of rules.

A specification language that is meant to bridge the gap between theoretical models and implemented systems is the Z-Framework of M. Luck et al. [13]. The Z schemata of "object", "agent" or "autonomous agents" are more and more extended to capture different forms and components of agent architectures or organizational structures like dependence networks [14]. The major criticism of this approach is its drawbacks in the formulation of dynamic aspects. However, the major advantage is the direct transformation into object-oriented class structures for the single agents.

Agent-oriented software engineering in general can be seen as an extension of object-oriented analysis, design and implementation. Thus the usage or even enhancement of standard object-oriented techniques is a proximate idea. In the area of social simulation a suggestion was made for a general framework for multi-agent simulation in social science, called Massi*f* [15]. This framework uses (standard) UML sequence diagrams for describing interactions. *Agent UML* [16] on the other side focusses on the enhancement of these sequence diagrams as they are not apt for representing more sophisticated interaction protocols due to their message-based structure. *Agent UML* provides additional means for specifying negotiations with multiple, concurrently active participants. But for the purpose of designing behavior-oriented multi-agent simulations standard UML is almost satisfying, as we want to show in the following sections.

4 Using UML for Multi-Agent Simulations

4.1 Class Diagrams

A class diagram for an agent class shows the external view on this kind of agent. The external view is divided into the accessible part of the agent memory, i.e. the state variables that are perceivable for other agents, and the interaction methods. The methods are named and contain its interaction partner, which for example is the special class *Environment*, as arguments. Methods specifying interactions can also be documented using the activity graph associated with that agent class. For completely describing the interaction situation the diagrams of the other agents, that participate in the interaction situation, are needed.

4.2 Activity Graphs

General Components An activity graph in UML shows the behavior depicted as activity flow of objects in an object-oriented program. The main nodes in an activity graph are actions or activities. The latter are compositions of actions. Additionally there exist state nodes that are known from the UML state graph. State graphs can be seen as a specialization of an activity graph and are used in UML for describing state changes of a single object in contrast to activity changes

concerning many objects in an activity graph. Other nodes are signal receiving and sending nodes. These nodes are important for modeling interactions and are explained later in more detail.

Transitions between nodes are defined based any form of conditions. One frequently used condition for a transition from an activity consists of the termination of this activity together with the activation of another state or activity. The modeler may use a rhombus node to show that there is an explicit decision in determining the next state or activity.

Activity Graphs for Multi-Agent Models As in the design of a multi-agent system in general, one can use activity graphs for describing the activity flow of one or more simulated agent classes. The transitions are conditions on external perceptions and internal state values (also goals, desires, and intensions if the agent architecture uses such concepts).

If there is more than one transition from a node, but the associated conditions form a crucial part of the model, then an explicit decision node should be used. As we observed in most of our applications, domain experts are using state-like behaviors (e.g. the IDLE "behavior") in their model to capture e.g. waiting. These can be better represented by state nodes.

There exist two special nodes: the start node and the end node. The agent starts its behavior at the start node, which entails no actions or activities and no incoming transitions. A filled circle depicts this start node. In an analogous way the end node is a dead end for the behavior of the agent, i.e. it has no outgoing transitions and can be interpreted as the terminating state of the agent. The end node is a filled circle surrounded by another unfilled circle.
This usage of UML activity graphs is straightforward, not only for describing the modeled behavior but also for generating the executable code of this agent. However, relationships and interaction between agents have to be explicitly shown. These involve agent-agent and agent-environment interactions. Thus a combination of activity graphs is necessary. How this is be managed is shown in the following.

Interaction Types Until now relationships between the behavior of several agents are hidden in activities in different graphs. In the following we are examining possible interaction types and suggest how these interactions can be expressed in (standard) activity graphs:

In general possible interaction forms are direct or indirect (via the environment) interactions. For combining activity graphs a further distinctions is necessary: whether the receiver may be another instance of the same agent class or an instance of another agent class or a combination of these. Thus an interaction may have one of the following forms:

a) *Object flow* is the prominent method for indirect interactions. One agent creates a resource object, that is perceived by other agents, that in consequence adapt their behavior due to the modified environmental configuration. If a pheromone trail is implemented by pheromone resource objects,

then depositing and following of trails corresponds to exactly this kind of interaction.

b) *Agent creation* is an important component of dynamic, variable structure model. An example is the production of offsprings frequently used in biological models.

c) *Manipulation of public instance variables* is the most technical form. It exhibits three subtypes: The sending agent modifies values either of itself, of a direct receiver or an intermediary. The first kind represents some sort of indirect broadcast, the sender changes its shape for the agents that are able to perceive it. Manipulating the variables of a receiver corresponds to directed interaction. If the changes concern variables of the environment we can observe a special case. It can be both directed and broadcast-like depending whether the environment serves as a communication medium or not.

d) The standard form of interaction is the synchronous and directed *sending and receiving of messages*. This can be part of complex negotiation protocols. A special form is broadcast communication in an explizit and synchronous form.

These four kind of interactions can be represented in activity graphs based on sending and receiving nodes:

Specification of Interaction in Activity Graphs UML provides a facility for describing these interactions via signal sending and receiving nodes in activity graphs. Inserting these nodes into an activity graph is straightforward, but additional information concerning the concrete interaction might be necessary. Adding this enhanced specification to the sending-receiving node combination is sophisticated and depends on the interaction type. To clarify the sender and the receiver we draw a link between these two nodes and attach an additional node to this link. This is an optional procedure in UML but we want to restrict ourselves to establish a transparent representation. In detail the form of this additional node depends on the interaction types (see figure 1 for an illustration):

a) Object flow in general is specified by signal sending and receiving nodes and a link that is annotated by the class diagram of the "exchanged" objects.

b) Agent creation is represented by a link between the sender node and the start node of the new agent's activity graph. This is a special case since no receiver node is addressable for the created agent.

c) For variable manipulation we propose to augment the link with a documentation node that contains the necessary information about the sender, the receiver and the involved variable. Unfortunately a documentation node in standard UML just contains of unstructured text. An additional node type may be more appropriate. As we are restricting ourself here to standard UML, we postpone suggestions about additional node types to other papers.

d) Interaction via direct messages can be specified by both object flow links or documentation annotated links. This depends on the complexity of the message. Details of this kind of interaction are better specified using sequence diagrams.

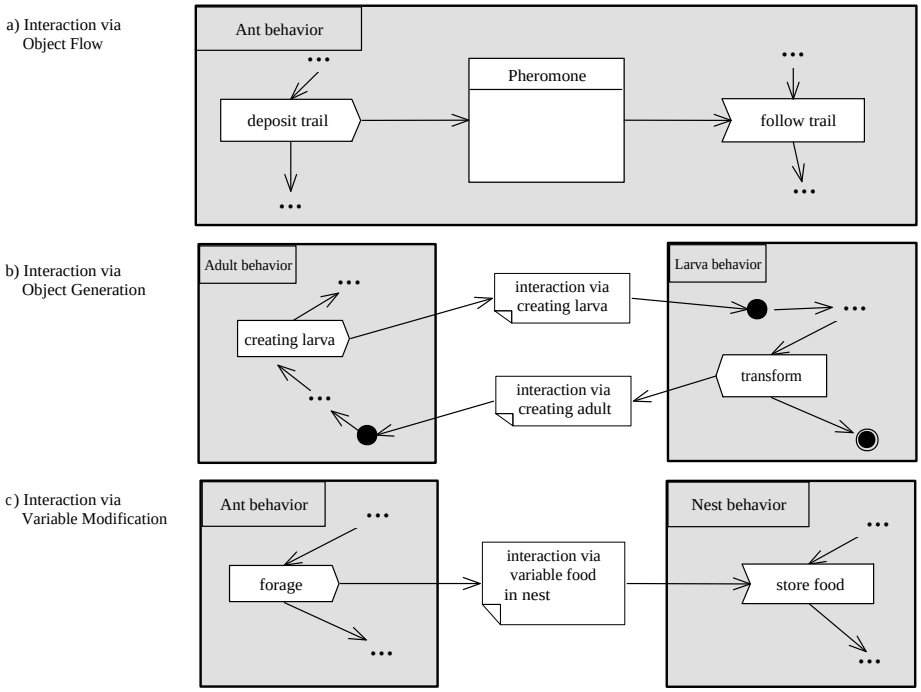

Figure 1. Specifying Different Interactions Types in UML.

4.3 Sequence Diagram

In sequence diagram the message passing between objects is described. On the x-axis the participants of the interaction situation are listed, the y-axis is the timeline. For direct interactions, *Agent UML* [16] proposes enhancements to UML for sending a message to a few concurrently active agents. The overall aim of Agent UML is to provide a framework for sophisticated (reusable) negotiations. In behavior-oriented multi-agent simulations this kind of interactions may be rarely found due to the focus on agent behavior and not on sophisticated internal reasoning processes which are prerequisites for complex negotiation abilities.

Moreover, most of the interactions in such simulations take the form of indirect ones, that means the happen via coordinated changes in the environment ("stigmercy" [17]). Asynchrony and undirectedness are therefore important features of this kind of interaction. Thus sequence diagram are not well suited. Nevertheless, for documenting a typical path of interaction they are useful. But, a modeler using sequence diagrams in this case should keep in mind that both, participants and the concrete sequence of primitive interactions, are just examples and not fixed and complete specifications of protocols.

Without a specific protocol for the interactions it is useful to formulate invariants and constraints on the system that contains these interactions. UML also provides a facility: OCL.

4.4 Object Constraint Language (OCL)

An often overlooked feature in UML is the possibility to augment UML diagrams with OCL expressions. OCL ("Object Constraint Language" [18,19]) is a formal language for expressing side effect-free constraints. A modeler can use OCL to specify constraints and other expressions and attach them to his models, with the aim of formalizing necessary information for e.g. testing and verifying his simulations. OCL was introduced and added to UML because a graphical specification, like a class model, is not precise and unambiguous enough for specifying all details. There is a need to describe additional constraints about the objects, even more about the agents in the model. Without OCL constraints have to be written down in natural language with the consequences of its ambiguities and impreciseness. On the other hand, formal languages tend to be hard to read due to their mathematical background. OCL tries to fill this gap: It is a formal language and it is easy to read and to write. More precisely OCL can be used for ...

- specifying invariants for classes or stereotypes,
- describing pre- and post conditions on operations and methods,
- describing guards and event triggers (or transitions),
- tracing associations in object graphs and
- specifying constraints on operations, an operation constraint can be read as a definition for the operation.

These application areas are specially relevant for describing indirect and asynchrone interactions in a multi-agent simulation. Moreover, this kind of declarative and explicit information about the interacting system as a whole and its components is special useful since a procedural definition of interactions is lacking.

One can identify two different kinds of invariants: local and global. An invariant for one class is called a local invariant independent from the number of participating agents. If instances from different class are involved, we call it a global invariant. For an example see figure 2a.

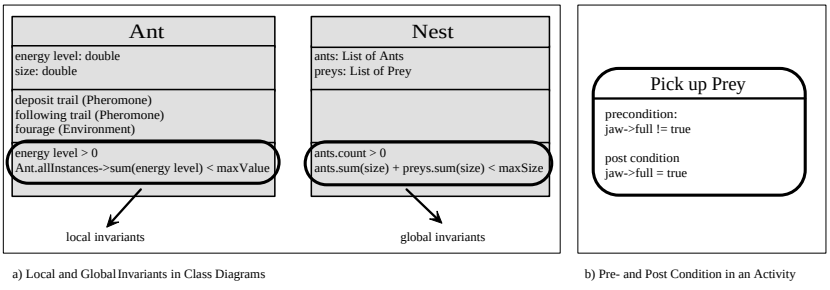

Figure 2. Using OCL for multi-agent model.

Another important application is the assignment of pre- and post conditions to activities. Figure 2b shows an example. Other ways of applying OCL are also useful. OCL is the language for formulating the conditions of the transitions in an activity graphs described above. In an analogous way every side effect-free computation can be specified using OCL.

Although OCL provides no completely precise semantics, it is possible to restrict the power of this language for making it executable. Analyzing the applicability and usefulness in concrete applications of behavior-oriented multi-agent simulations, remains a main part of our future work. We are now working on a environment for designing and implementing executable simulations that is based on the concepts described here.

5 Conclusion and Further Work

In this paper we examined how to use standard UML for specifying behavior-oriented multi-agent simulations. This kind of multi-agent models are characterized by a focus on behavior and interactions of agents in a simulated environment.

Activity graphs are the prominent mean for formalizing behavior of many interacting agents. In general they are used for specifying behavior, here the activity flow of an agent or an agent class. However, in standard UML interactions are not explicitly tackled, since they are mostly hidden in several activity nodes, not directly connected with standard arrows. For multi-agent simulations this is a drawback as interactions form the central part. Based on some not widely used features in UML we presented a viable solution for the representation of interactions. For this we identified four major types of interactions in behavior-oriented multi-agent simulations. Based on sending and receiving nodes with augmented links, all four can be specified. The augmentation consists of additional nodes: single class diagrams or documentation nodes. The latter is a starting point for our future work as instead of using unstructured text information, a structured node would improve the description of the interaction.

UML provides another very important feature, namely the OCL. The formal expression language is both, precise and unambiguous. Thus it can be used to describe important facets of the model, to reduce the gap between modeling language and concrete implementation.

After an extensive phase of evaluation of this suggestions using real world applications, we want to show in our future work that using UML for multi-agent simulations improves the quality of model description. Thus we hope that it enables a modeler to build and execute multi-agent simulations with the same high standard of validation and verification as it is propagated in standard simulation models.

References

1. G. Booch, I. Jacobson, J. Rumbaugh, and J. Rumbaugh. *The Unified Modeling Language User Guide.* The Addison-Wesley Object Technology Series. Addison Wesley, 1998.

2. F. Klügl. *Multi-Agent Simulation – Concept, Tools, Application (in German)*. Addison Wesley, 2001.
3. F. Klügl. *Activity-based Behavior Modeling and its Support for Multi-Agent Simlation (in German)*. PhD thesis, Universität Würzburg, 2000.
4. A. M. Uhrmacher. Object-oriented and agent-oriented simulation: Implications for social science application. In K. G. Troitzsch, U. Mueller, G. N. Gilbert, and J. E. Doran, editors, *Social Science Microsimulation*, chapter 20, pages 432–447. Springer, 1996.
5. C. Urban. Pecs: A reference model for the simulation of multi-agent systems. In R. Suleiman, K. G. Troitzsch, and G. N. Gilbert, editors, *Tools and Techniques for Social Science Simulation*. Physica-Verlag, Heidelberg, 2000.
6. J. P. Müller. Architectures and applications of intelligent agents: A survey. *Knowledge Engineering Review*, 13(4):353–380, 1999.
7. A. Repenning. Agentsheets: an interactive simulation environment with end-user programmable agents. In *Proceedings of INTERACTION 2000, Tokyo, Japan*, 2000.
8. N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system: A toolkit for building multi-agent simulation. http://www.santafe.edu/projects/swarm/, 1996.
9. M. Wooldridge, N. R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In *Proceedings of the 3rd Internation Conference on Autonomous Agents,1999*. ACM Press, 1999.
10. C. Iglesias, M. Garijo, and J. C. Gonzales. A survey of agent-oriented methodologies. In J. P. Müller, M. Singh, and A. S. Rao, editors, *Intelligent Agents V: Proceedings of the ATAL'98*, volume 1555 of *LNAI*. Springer, 1999.
11. M. Fisher. Representing and executing agent-based systems. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Proceedings of the ATAL'94*, volume 890 of *LNAI*, pages 307–323. Springer, 1995.
12. S. Moss, H. Gaylard, S. Wallis, and B. Edmonds. Sdml: A multi-agent language for organizational modelling. CPM-Report 16, Centre for Policy Modelling, 1997.
13. M. Luck, N. Griffiths, and M. d'Inverno. From agent theory to agent construction. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III (= Proceedings of ATAL'96)*, volume 1193 of *Lecture Notes in Artificial Intelligence*, pages 49–63. Springer, 1997.
14. F. Lopez y Lopez, M. Luck, and M. d'Inverno. A framework for agent architecture, dependence, norms and obligations. In *(Pre-)Proceedings of the MAAMAW'2001*, 2001.
15. E. Mentges. Concepts for an agent-based framework for interdisciplinary social science simulation. *Journal of Artificial Societies and Social Simulation*, 2(2), 1999.
16. J. Odell, H. van Dyke Parunak, and B. Bauer. Extending uml for agents. In *Proceedings of Agent-Oriented Information Systems 2000, Workshop at the AAAI 2000*, 2000.
17. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence - From Natural to Artificial Systems*. Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press, Oxford, 1999.
18. OMG. *Object Constraint Language Specification*. http://www.omg.org/cgi-bin/doc?ad/97-08-08, 1997.
19. J. B. Warmer and A. G. Kleppe. *The Object Constraint Language : Precise Modeling With Uml*. The Addison-Wesley Object Technology Series. Addison Wesley, 1999.