From Simulated to Real Environments: How to Use SeSAm for Software Development

Franziska Klügl, Rainer Herrler, and Christoph Oechslein

Lehrstuhl für Künstliche Intelligenz und Angewandte Informatik
Institut für Informatik, Universität Würzburg
Am Hubland, 97074 Würzburg
{kluegl,herrler,oechslein}@informatik.uni-wuerzburg.de

Abstract. In this paper we want to show the possibilities to use agent-based modeling and simulation for software development. Therefore we present the integrated environment SeSAm and recent extensions, that allow creating simulated environments for agent based software as well as actually developing and deploying software agents.

1 Introduction

The development of agent-based software[1] is a quite hard task that incorporates difficulties like for example sophisticated software architectures, synchronization of concurrent processes with distributed control and data as well as unpredictability of interaction patterns [10]. For solving these problems, there exist not only diverse approaches for agent-oriented software engineering, but also lots of tools and frameworks for implementing agent-based software.

In spite of constant progress in agent-based software engineering the gap between specification and implementation is seldom addressed: The question is how to implement the completely designed agent system in a way that its validity and robustness is ensured? The problem here is twofold. On one hand the specification of the agent system has to be translated into executable code, on the other hand this code has to be tested thoroughly for validation and verification. We want to show in this paper, that – based on an existing simulation environment for agent-based models – tools can be provided that not only support the development of agent-based software but also the testing in different useful ways.

After going into more detail about agent-based software development, we continue in the remainder of the paper by presenting the modeling and simulation environment SeSAm, that allows visual agent modeling as well as experimenting and analyzing agent-based simulation experiments. In section 4 and

[1] We distinguish between "agent based software" and "agent based simulation" based on the environment the agent system is existing in. In agent based simulation the environment is also simulated and completely under the control of the modeler. Agent-based software is working in a real-world environment. Nevertheless, also developing agent-based simulations is sophisticated as well.

M. Schillo et al. (Eds.): MATES 2003, LNAI 2831, pp. 13–24, 2003.
© Springer-Verlag Berlin Heidelberg 2003

5 we describe the possibilities for using SeSAm as a tool for supporting agent based software development. In the first place an approach will be proposed for modeling a simulated environment as a testbed for agent systems, afterwards we describe how to transfer automatically agent programs from simulated environments to real-world environments. We terminate with a short conclusion and outlook about further work.

2 Agent-Based Software Development and Simulation

Agent-based methodologies for engineering multi-agent systems try to direct the developers by providing process models for the single phases of development: requirement engineering, analysis and specification. They are mostly proposing graphical means for representing their outcomes. There is a huge amount of proposals that aim at supporting agent systems for distributed problem solving applications, e.g. GAIA [20], other agent-oriented software engineering methodologies are proposed specially for open agent systems, like [11]. A survey about methodologies and the problem in general can be e.g. found in [9].

However, the problem of actually producing code is left to traditional object-oriented procedures, as these methods merely stop at the design level. As another extreme, there are some formal specification frameworks that produce executable code. The earlier ones built upon temporal logic [4], BDI Logic [18] or Z schemes [14]. At least the logic-based frameworks have problems when applied to more complex problems with a lot of different agents.

More practically oriented frameworks were proposed by providing modifications and enhancements of UML specifications. The most famous is Agent UML [15] that was introduced to capture the flexible, high-level interactions between agents. Other specification languages concern class diagrams for specifying social concepts. MESSAGE/UML [2] is providing means for formulating knowledge-level concepts, like goals or roles by extending class and activity diagrams. Flake et al. [5] focus on how to generate code for a multi-agent system from UML diagrams based on a method called CASA.

On the other hand there are lots of tools for supporting the implementation by providing environments for the development of agent-based software with pre-defined building blocks of platforms according to FIPA standards. The *Agent Construction Tools* Site[2] lists (at 2003-05-11) 24 tools for mobile agents, agent infrastructure or development as commercial products and 36 products from research institutes. Another extensive, yet very heterogeneous list can be found on the AgentLink pages. Here only two environments for FIPA compliant platforms, like JADE[3] or FIPA-OS[4] should be mentioned. It seems that at least JADE is not only used by the developing institute.

Whereas there are some approaches for formal verification of agent-based software [21], testing agent-based software mostly refers to debugging issues.

[2] `http://www.agentbuilder.com/AgentTools/`
[3] `http://sharon.cselt.it/projects/jade/` or [17]
[4] `http://fipa-os.sourceforge.net/`

There are no tools for systematic testing as e.g. are used in general modeling and simulation area for determining the sensitivity of a program to environmental changes. This is of course due to the fact that the environment in which the agent-based software will be deployed is not part of the implementation environment. If the tools provide a simulated environment in necessary detail testing agent based software would be facilitated. However, modeling and simulating the environment and also simulating the behavior of the agents in reaction to their abstracted, potentially simplified environment seems to be useful but seldomly manageable as it might require more than double work - implementing the real agent system, the simulated environment and the simulated agent system in this environment. Uhrmacher et al. propose in [19] to use reflection for adapting the agents behavior for acting in a simulated environment.

Our UML extensions (called SeSAmUML see [16]) focus on behavior-based specification of agents based on activity diagrams.It therefore share some similarities with some other specification languages or tools by also using the uml activity graph or the uml state graph (see [3,6,7]). Our approach for bridging the gap between specification and implementation is to provide a (complete) visual modeling tool that allows to build a specification based on analogous graphical primitives and afterwards implement this specification by adding additional information to render more precisely. The approach described here is different from Jade [17] or JACK [8] as it supports the simulation and the analysis of the simulation results of the implemented agent models. It also supports the visual modelling of behavior based agent type models in contrast to Jade (only a Java-framework with low-level predefined agent types) or JACK (which support BDI agent types but not continuous visual).

3 SeSAm

SeSAm stands for "Shell for Simulated Agent Systems". It provides a generic environment for modeling and experimenting with agent-based simulations. Its development was specially focused on providing a tool for the easy construction of complex models, as the main vision behind it was a tool for researchers without actual programming background [13].

In the following we want to present some important properties of SeSAm. We are currently working on a methodology for performing simulations, but have to evaluate and maybe modify it in the context of multi agent software development. Therefore we will present our results in addition to a deep evaluation and a long case study in a forthcoming paper. More information about SeSAm can obtained from `http://www.simsesam.de`.

3.1 SeSAmUML and Visual Agent Programming

SeSAm-Agents consist of a body that contains a set of state variables and a behavior that is implemented in form of UML-like activity diagrams. Activities

(the nodes of an activity graph) are seen as scripts that are initiated and terminated by firing rules. Based on this simple concept both, a specification language and a software environment for modeling and simulation were developed.

The specification framework focuses on the representation of agent behavior especially in relation to other agents, resources or the general environment [16]. An example can be seen in figure 1. This is a part of a specification of a model about the evolution of sociality in ants. It shows the different agent classes, their behavior and interaction. Behavior graphs are augmented with class diagrams

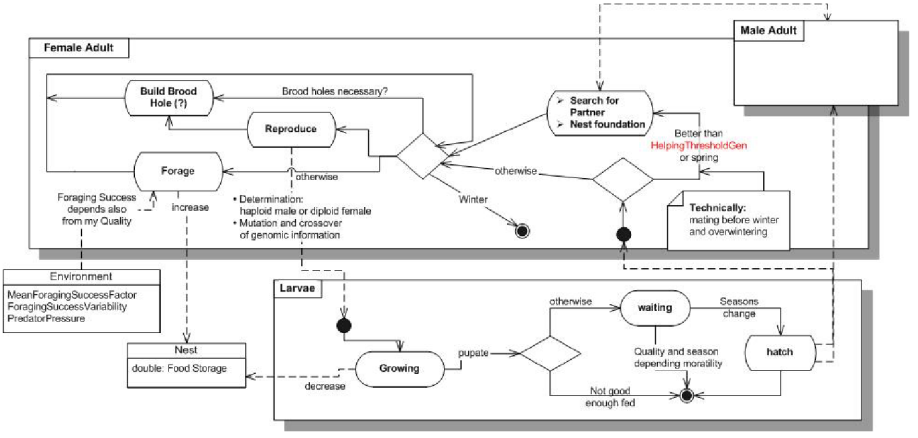


Fig. 1. Example specification of a biological model with interactions between different agents (Female, Male Adult and Larvae) and resources (Nest). Also some parameters of the general environment influence certain activities.

for the agents internal structure. Basically there is a predefined distinction between agents (active entities), resources (passive entities) and the environment (global surrounding). Only in simple models no protocol-type specification of the agents interaction is necessary. In these cases, most interaction consists mainly of implicit coordination: One agent changes the environment more or less deliberatively. Any other agent may perceive the modified surroundings and adapts its behavior according to it. More complex agent based models require a detailed specification of interaction sequences.

Such a specification of an agent-based simulation model can be directly implemented using SeSAm, if the description provides sufficient details down to primitive actions and perception predicates. With every activity actions are associated which are executed when the agent is in this activity. Rules are responsible for termination and selection of activities. Actions of activities and conditions of rules can be composed from an extensive number of primitive building blocks. Figure 2 shows a screenshot of a part of the implementation of the specification shown in figure 1, namely central behavior of an agent of the class *female adult*.

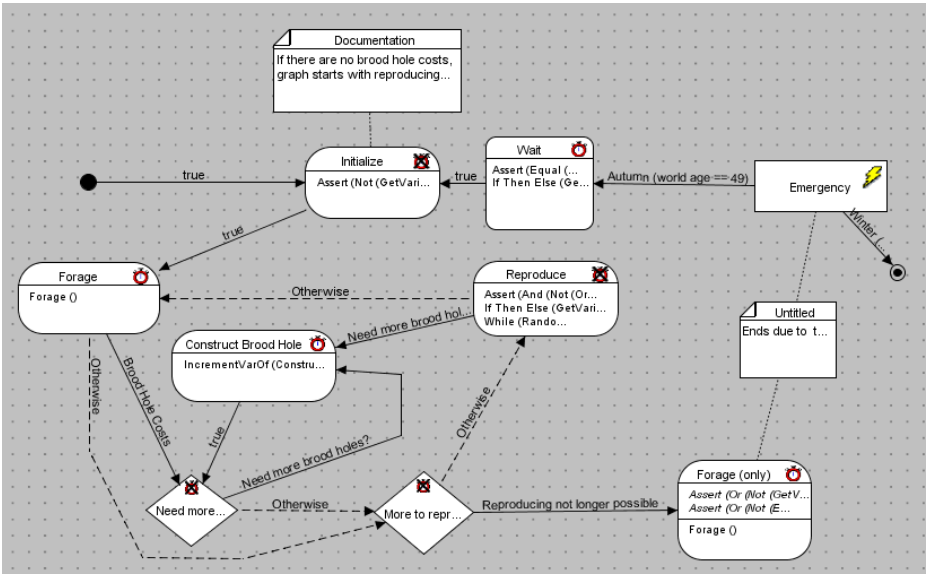


Fig. 2. Screenshot of an implemented SeSAm-model: Partial behavior of an agent of the adult female class as specified in figure 1.

In addition to these normal activity nodes and rule arrows, also special forms of nodes like input or output nodes were introduce. These describe activities where the agent interacts with other agents and thus are marked for denoting this special purpose. Processing its activity graph initiates at the start node and terminates with the end node. The lifetime of an activity graph is over when the end node is selected. Activity graphs can be hierarchically organized in a way that a rule selects an activity graph instead of a simple activity. Additionally several activity graphs may determine the behavior of an agent in parallel. The agent itself is terminated when one of its activity graphs ends. Thus the implementation of rather complex behavior is possible that can even resemble the functionality of some reactive planning routines, like RAP.

As indicated in figure 2, a user is able to design visually the behavior of an agent. Analogous mechanisms are provided for specifying the static structures of an agent, an agent system or the environmental elements and their configuration. Thus a complete agent-based simulation model can be implemented visually without programming in a traditional programming language.

3.2 Modularization

Graphical modeling is built upon a set of predefined primitive elements – action primitive, perception primitive or operator primitive. They can be combined to function calls visualized in form of a tree. Such an expression can be constructed every time when needed, or can be composed into user functions with arguments.

They may represent more abstract building blocks or subroutines. Possible forms of combination are recursion, selection or composition.

The primitive actions, perceptions and operators form the interface to the underlying programming language Java. For integrating such a Java method, it has to "declared" with additional information that describe its arguments, functionality, etc. This declaration is interpreted for the visual programming integration.

A set of attributes and user functions may be defined independently and grouped to a so called "feature" representing some notion of additional non-basic abilities that can be assigned to agents (or the environmental classes). They may be implemented visually by the modeler supporting modularization. Predefined features are for example spatial representation and functions that capture movement, or gene attributes with evolutionary operators.

Another important issue is the question about the basic data types available for agents static structure and behavior implementation. In addition to standard data types, like boolean, numeric, string values there are also composed data types like hashtables or lists (with index-based access). Besides the built-in (or via features added) types a modeler can specify new types. They support sophisticated high-level agent structures and behaviors. There are two types of user types: enumerations and compositions. The range of first consists of a collection of predefined symbols, for example "female" and "male" for a data typ "gender". The value of a variable of this type is one element out of this collection, here e.g. "female". The second form of user defined data types is composition where an arbitrary number of attributes with other data types (also user defined ones) may be aggregated.

3.3 Database Interface and Ontology Import

For large input data it is neither comfortable nor useful to implement and fill complex data structures for simulation, since databases are available for the organization and storage of large data sets. Therefore a special feature provides an online interface to an JDBC database (mySQL). Attributes of agents (or the environment) can be connected to database entities. Thus, agents may read and write from and to these database entities. With this additional components real world data, e.g. weather data or task structures, may be integrated into an agent-based simulation. On the other hand, this is no one way connection. SeSAm may be used to to provide an dynamic testbed for intelligent systems that use information from the database e.g. for planning.

For complex (open) agent systems the first step for ensuring interaction abilities is to define an ontology that clarifies the common terms, relations. For certain domains, established ontologies exist. Thus grounding a simulation model on an already existing ontology increases readability and transparency of the modelled structures. This supports not only communication about the model, but also later enhancements of it. Therefore we developed an plugin for importing large parts of any ontology developed in Protege into SeSAm-models. The structures are directly converted to user data structures from where they can

be converted using standard refactoring tools of SeSAm to agent or resource classes, attributes, etc.

3.4 Simulation and Experimentation Support

All information about and parts of a simulation model – from agent structure and behavior to relevant features and ready-to-run simulation run specification – are stored into a XML file based on a XML schema. Based on this model data format we developed a converter for a printable html visualization of a complete model.

Support for simulation cannot end when the model is implemented completely. Testing and executing experiments are also very demanding and effortful tasks. As there are freely configurable instruments for gathering data and scripting options for constructing simulation experiments on several computers concurrently, SeSAm is a highly valuable tool for agent based simulations.

However, due to fact, that SeSAm is made for agent based *simulations*, the programming facilities may only be used for agents in a simulated world that is completely under the control of the modeler. This does not work when aiming at agent-based software. Therefore we aim at extending the usability of SeSAm by adding new enhancements for relating SeSAm agents to real world agents. This will be presented in the following sections.

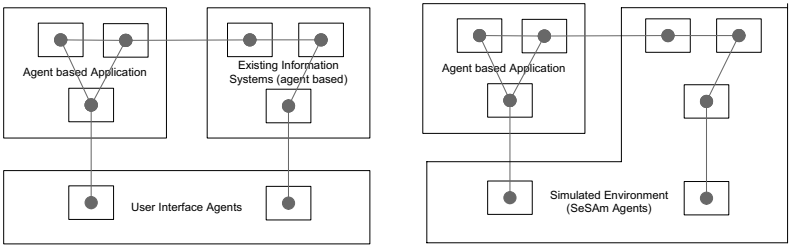
4 Simulation-Controlled MAS

Agent-based applications are part of an environment containing real world agents like users and other systems (possibly agent systems). In agent software development isolated tests are usually very difficult, instead tests have to be made under the conditions of this environment. As mentioned above, our idea was to extend SeSAm in a way that it can be used as a testbed for agent based software without re-modeling the agent for interacting with the simulated environment. Then, creating models of the global system becomes rather intuitive.

Figure 3 shows an sketch of two possible related scenarios. An agent based application is integrated in an environment consisting of additional information systems and users, that can also be seen as agents. The mapping of this scenario to a simulation controlled MAS is depicted in figure 3(b). All agents from the real-world environment are replaced by simulated agents in SeSAm. The level of detail of the simulation is depending on the requirements of the agent system. Due to the representation of the real-world in the simulated testbed the developed components can be tested both, under realistic as well as under extreme conditions. This usually cannot be done in the real world for reasons of cost or security.

4.1 Technical Base and Realization

To support connecting agent systems to SeSAm a new plugin was developed, that allows to extend SeSAm agents with FIPA compliant communication abili-



(a) Real world scenario (b) Simulated testbed scenario

Fig. 3. Sketch of a possible development scenario where 3(a) describes the original situation and 3(b) represents the simulated environment.

ties. Extended in this way SeSAm agents can interact with other agents running on any kind of FIPA platform. New primitives for registering and deregistering at a directory facilitator as well as for sending and receiving messages can be used in the SeSAm models. This is supported by the ontology feature mentioned above enabling domain modelers to construct agents and message contents from ontology terms and concepts.

Thus, coupling agent applications and environment (either the real or the simulated one) is possible with less effort, since all systems use standardized interfaces.

Technically, the SeSAm FIPA feature is realized by integrating and controling a complete JADE-platform. When SeSAm is started the developer may start the JADE platform with special platform configuration parameters or allow the default start by SeSAm. Since SeSAm allows to make several simulation runs in parallel, each simulation run - when started - is associated with a single agent container within JADE. This concept is used to keep the different worlds separated.

SeSAm agents in a running simulation can decide to create a representation of themselves in the associated JADE agent container. This agent can be seen as some kind of rather passive shadow. State representation and reasoning is still carried out by the SeSAm agent, but all actions regarding the FIPA functionality (mainly communication tasks) are delegated to the JADE shadow. Received messages on the other hand are stored by the shadow in an incoming message queue until the SeSAm agent performs a sensing action. The following primitive actions and functions were added to the SeSAm primitive set:

- `StartJade()`: This function starts a local JADE platform, if it isn't already running. A window showing a view on this platform is opened.
- `CreateShadow()`: The acting agent decides to create his shadow identity. The functions described below can only be called if a shadow has been created before. Not every agent in simulation needs to have a shadow.
- `GetJadeAgentsOfName(name)`: This function returns a list of addresses of agents with a given name. When executed this function makes a query to

the directory service of the platform. The returned address (agent identifier) can be used to contact the agents.
- `GetJadeAgentWithService(servicetype)`: This function returns a list of addresses of agents offering a certain service. The directory service at the local platform returns at matching candidates from the set of registered agents.
- `IsACLMessagePending()`: This boolean function returns true if the shadow agent has received at least one ACL-Messages. This message can then be received using the ReceiveACLMessage primitive.
- `ReceiveACLMessage()`: This function is used to receive a pending message, otherwise the simulation is blocked until a message is received. If such a blocking is not desired, one should use the IsACLMessagePending predicate before. The function returns the content of the message as well as the address of the sender for answering purposes.
- `SendACLMessage(receiver, content)`: This function sends a message to another agent running on any FIPA agent platform.

4.2 Example Application: *Agent.Hospital*

A example of the development of a simulation controlled MAS is the integrated health care scenario *Agent.Hospital* [12]. In *Agent.Hospital* various agent based health care systems offer their services on FIPA compliant platforms. These multi agent systems have been linked together with a joint directory facilitator and have also been integrated into the worldwide network of agent platforms, named *Agentcities*.

Agent.Hospital was developed by a couple of research institutions across germany, working together in a priority research program founded by the german research foundation (DFG). For testing the participants agreed on creating a integrated simulation scenario that may be used by all systems concurrently. The central control of the integrated scenario was modelled with SeSAm. An exemplified patient process was chosen, that required interaction with every projects agent system. The simulated patients were modelled by the domain experts, specifying all possible interactions with the agent applications. One prerequisite for interoperability of SeSAm and the agent applications was the development of a hospital ontology [1] that was built by all the participants in advance. The final result was a simulation model of a virtual hospital, where software agents and simulated agent systems were integrated. Simulation runs can show the overall behavior of the resulting system consisting of many different multi-agent systems in whole. In addition integrating a new agent application into this overall environment shows whether the implemented interfaces are applicable.

This example application can be seen as a first evaluation even if we have to make a more representative evaluation in future. The experiment showed that it is possible and comparatively easy to create simulated testbeds for software agents.

5 SeSAm-Agents Breaking Out

SeSAm development capabilities are not just limited to create environment simulations for software agents. As the language for agent behavior descriptions has the power of a programming language, it can be used for development of agent applications more directly by using the visual programming facilities for rapid agent development.

As described above, an agent behavior representation at specification level is provided, thus programming becomes quite comfortable and seems to be more like a modeling process, without any error-prone gaps between specification and running code. Therefore our idea is to take agents – implemented in SeSAm – out of the simulated environment and apply them in agent based programs for real-world problem solving.

The first step towards this real-world applicability has already been made by enabling SeSAm agents to communicate with FIPA agents. But this is not enough. In the next step, we let agents break out from the simulated environment. There are two possible ways of technical realization:

- **Interpreting Agent Wrapper:** A generic JADE agent was developed, that is able to load and interpret the SeSAm behavior description. This generic agent wrapper works like an adapter around the agent description. We are now evaluating restrictions in SeSAm behavior specification, that have to be meet for external use. Such restrictions for example are that step-wise simulation is not used implicitly for synchronization of agents or that the search for communication partners is done using a directory facilitator.
- **SeSAm to JAVA Compiler:** In some cases developers may prefer creating source code. This is a very common practice especially for building graphical user interface or code generation from UML diagrams. Code generation in SeSAm is not yet available at the moment, however first promising steps have been undertaken.

Both approaches are very similar in their power, but there are some issues for deciding between compilation and interpretation. One might be that compiling restricts round trip engineering, as well as dynamic use during runtime. On the other side compiling might cause faster execution and is more flexible as the compiled result can be modified and extended manually.

However, when used for software development SeSAm has to compete with existing integrated development environments for (non-visual) traditional programming languages. Due to the powerful, yet pre-defined agent structure and abilities, there is a huge potential for providing e.g. online syntax consistency tests, refactoring and debugging functions. For facilitating the development on agent-based simulation models we already offer some of these features.

As a small test example we used a simple supply chain model for bicycle production. Results showed that is it possible to use the XML behavior description outside a simulated environment based on a wrapper for SeSAm agent in JADE. The next steps are integrating interfaces for users and other real-world data sources. Another focus is the general applicability of this approach.

6 Conclusion and Further Work

In this paper we presented several approaches for practically relating agent-based simulation and software development. We showed, how virtual environments for agent based software can be created based on the interoperability features of SeSAm. The feasibility and utility of this approach was shown with the *Agent.Hospital* project. A more visionary proposal focused on the re-use of the powerful visual agent programming facilities of SeSAm for allowing simulated agents to be deployed in real world. The interpretation approach for the transfer of simulated agents to real-word environments is almost available, whereas the development of the compilation approach just has started. For further development of the approaches presented here, more sophisticated application scenarios are necessary and also further improvements in SeSAm concerning implementation support.

Another important point is that agent dynamics in SeSAm are just described in activity graphs, not generated based on goals and planning abilities. This would be necessary for applications of really intelligent agents. Their development would profit a lot from the abstract structures and visual interfaces provided by SeSAm. Enhancement in the SeSAm agent abilities are not only under realization for scheduling, but also planned for goal-oriented behavior.

Acknowledgement. The authors would like to thank Alexander Huber and his students from the TU Erlangen for their implementation of the database interface. The work described in this paper was partially supported by DFG under SFB 554 (Christoph Oechslein) and SPP 1083 (Rainer Herrler).

References

1. Becker, M., Heine, C., Herrler, R. and Krempels, K.H. 2002. *OntHoS - an Ontology for Hospital Scenarios.* In: Technical Report No. 300, Julius-Maximilians-Universität Würzburg, Institut für Informatik, September, 2002
2. Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Leal, F., Chainho, P., Kearney, P., Stark, J., Evans, R. and Ph. Massonet. 2001. *Agent Oriented Analysis Using Message/UML* in. Agent-Oriented Software Engineering, LNCS 2222, pp. 119ff
3. DeLoach, S.: Analysis and Design using MaSE and agentTool. In: *Proceedings of the 12th Midwest Artificial Intelligence and Cognitve Science Conference (MAICS 2001).* Oxford, Ohio, 2001
4. Fisher, M. 1995. Representing and Executing Agent-Based Systems. In Wooldridge, M. and Jennings, N. R., editors, *Intelligent Agents: Proceedings of the ATAL'94*, volume 890 of *LNAI*, pages 307–323. Springer.
5. Flake, S., Geiger, Ch. and J. Küster. 2001. *Towards UML-based Analysis and Design of Multi-Agent Systems* In. International NAISO Symposium on Information Science Innovations in Engineering of Natural and Artificial Intelligent Systems (ENAIS'2001), Dubai March 2001.

6. Gervais, M.-P. ; Muscutariu, F.: Towards an ADL for Designing Agent-Based Systems. In: *Proc. of Agent-Oriented Software Engineering (AOSE) 2001, Agents 2001, Montreal*, 2001, S. 49–56
7. Griss, M. ; Fonseca, S. ; Cowan, D. ; Kessler, R.: Using UML State Machine Models for More Precise and Flexible JADE Agent Behaviors. In: *Proceedings of the Third International Workshop on Agent-Oriented Software Engineering (AOSE-02), Bologna, Italy*, 2002
8. Howden, N. ; Rönnquist, R. ; Hodgson, A. ; Lucas, A.: JACK Intelligent Agents — Summary of an Agent Infrastructure. In: *5th International Conference on Autonomous Agents*, URL `http://www.agent-software.com`, 2001
9. Iglesias, C., Garijo, M., and J. C. Gonzales. 1999. A Survey of Agent-Oriented Methodologies. In Müller, J. P., Singh, M., and Rao, A. S., editors, *Intelligent Agents V: Proceedings of the ATAL'98*, volume 1555 of *LNAI*. Springer.
10. Jennings, N. R. 2000 On Agent-Based Software Engineering. *Artificial Intelligence*, 117:277–296, 2000.
11. Juan, T., Pearce, A. and L. Sterling. 2002. *ROADMAP: Extending the Gaia Methodology for Complex Open Systems* Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002), Bologna, Italy, July 2002.
12. Kirn, S., Heine, C., Herrler, R. and K. H. Krempels. 2003. *Agent.Hospital - Agent-Based Open Framework for Clinical Applications* accepted at WETICE, Linz, June 2003.
13. Klügl, F. 2001. *Multi Agent Simulation - Concepts, Tools, Application*(in german), Addison Wesley, München
14. Luck, M., Griffiths, N. and M. d'Inverno. 1997. From Agent Theory to Agent Construction. In Müller, J. P., Wooldridge, M. J., and Jennings, N. R., editors, *Intelligent Agents III (= Proceedings of ATAL'96)*, volume 1193 of *Lecture Notes in Artificial Intelligence*, pages 49–63. Springer.
15. Odell, J., van Dyke Parunak, H., and B. Bauer. 2000. Extending UML for Agents. In *Proceedings of Agent-Oriented Information Systems 2000, Workshop at the AAAI 2000.*
16. Oechslein, C., Klügl, F., Herrler, R. and F. Puppe, 2002. *UML for Behavior-Oriented Multi-Agent Simulations.* In: Dunin-Keplicz, B., Nawarecki, E.: From Theory to Practice in Multi-Agent Systems, CEEMAS 2001 Cracow, Poland, September 26-29, 2001, (= LNCS 2296), Springer, Heidelberg, pp. 217ff
17. Poggi, A. ; Rimassa, R.: JADE - A FIPA-compliant agent framework. In: *Proceedings of PAAM'99, London*, 1999, S. 97–108
18. Rao, A. S. 1996. *AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language* In W. Van de Velde and J. W. Perram, editors, Agents Breaking Away: Proceedings of the 7 th 24 European Workshop on Modeling Autonomous Agents in a Multi-Agent World, (LNAI Volume 1038), 42-55. Springer-Verlag.
19. Uhrmacher, A. M., Kullick, B. and J. Lemcke. 2002. *Reflection in Simulating and Testing Agents.* In 16th European Meeting on Cybernetics and Systems Research (EMCSR 2002), 2002.
20. Wooldridge, M., Jennings, N. R., and D. Kinny. 1999. A Methodology for Agent-Oriented Analysis and Design. In *Proceedings of the 3rd Internation Conference on Autonomous Agents,1999.* ACM Press.
21. Wooldridge, M. 1997. Agent-Based Software Engineering. *IEE Proc. on Software Engineering*, 144(1):26–37, 1997.