A Generic Simulation Service for Distributed Multi-Agent Systems

L. Braubach, A. Pokahr, W. Lamersdorf
Distributed Systems and
Information Systems
University of Hamburg, Germany
{braubach, pokahr, lamersd}
@informatik.uni-hamburg.de

K.-H. Krempels
Communication and
Distributed Systems
RWTH Aachen, Germany
krempels@i4.informatik.rwth-aachen.de

P.-O. Woelk
Institute of Production Engineering
and Machine Tools
University of Hannover, Germany
woelk@ifw.uni-hannover.de

Abstract

Multi-agent systems are well suited for building large software systems. A great deal of these complex systems includes process flows that are concerned with time or are even time-critical. The activities of these process flows are often executed in distributed autonomous subsystems that have to be synchronized with respect to the superordinated task execution. To be able to build such systems and test their behaviour adequately, it is often advantageous and sometimes necessary to simulate them in the run-up to their practical use. Testing and simulation of process flows within multi-agent systems requires synchronization of the participating agents with respect to the global simulation time. In this paper, a design proposal and a service implementation for testing and simulation is presented, which takes care of the special requirements imposed by multi-agent settings. This so called time service is implemented as a FIPA-compliant agent, and can be used to couple heterogeneous subsystems implemented on different agent platforms.

1 Introduction

Complex enterprise software systems consist of several more or less tightly coupled subsystems that are organized in a hierarchical fashion (i.e., the various subsystems contain other subsystems). It is argued that multi-agent systems (MAS) are well suited to reduce the complexity of building such software systems [Jennings, 2001]. MAS are able to capture the structure and hierarchical organization, as each subsystem can be represented by an autonomous agent, which may itself be a multi-agent system containing other agents. Two of these "multi-multi agent systems" (MMAS) are developed in the Agent.Enterprise and Agent.Hospital initiatives of the German priority research programme DFG-SPP-1083 "Intelligent Agents in Real-World Business Applications".[1]

A problem when developing large-scale systems is to coordinate the timely execution of activities inside the single subsystems, as well as activities which require coordination between different subsystems. This is even more true for agent-based systems, which highly emphasize the autonomy of the individual components. Only carefully elaborated design and extensive testing induces successful operation under all conditions. To solve this problem and to test the developed systems, it is useful to simulate them in the run-up to their practical use. This paper describes a standard-compliant middleware *time service* component enabling the simulation of process-flows in distributed MAS. The design and implementation of the time service is based on techniques from the field of simulation. It has the purpose to allow the timed synchronization among the participants and will control the progress of the over-all process flow. Further components necessary for a complete simulation environment, e.g. responsible for generating external events or for result analysis are not considered.

In the next section, the foundations of multi-agent based simulation are described. Thereafter, the context of this work is presented in Section 3. In Section 4, the design and implementation of the time service with respect to the special properties of multi-agent systems will be described. In Section 5, some related work is presented and, finally, some concluding remarks are given.

2 Multi-Agent Based Simulation

Complex software systems are still extremely difficult to design and implement, and prototypes have to be tested extensively before they are used in practice. To be able to test and understand the runtime behaviour of a distributed time-based system, simulation techniques for multi-agent systems can be utilized. The multi-agent based simulation (MABS) differs from conventional simulation in that the entities constituting the system are agents. The topic is influenced by existing research areas such as *parallel and distributed discrete event simulation* [Fujimoto, 1999] and *object oriented simulation* (OOS) [Page, 1991]. Below, some basic ideas of these subjects are discussed, for a detailed survey about MABS see [Davidsson, 2000]. Note that in this contribution, unlike traditional distributed simulation, distribution is not introduced to increase the performance of simulation. Rather, simulation is used as a technique to test an inherently distributed system. However, the applied concepts are the same.

In discrete event simulation (DES) it is assumed that state changes in the world occur at distinct points in time and are caused by events. It can be seen as the counterpart

[1] `http://www.realagents.org`

to continuous simulation where state changes occur continuously over time. DES can be further subdivided into event-driven and time-driven approaches. Event-driven approaches utilize an ordered event list where time stamped events are stored. Progress is achieved by removing the earliest entry from this list, advancing the simulation time to the timestamp of that event and executing the event. This may lead to further entries of new events in the event list. Time driven approaches advance the time in constant time steps. In each step the clock is adjusted and the participants are informed about the new time. They now can check, if an activity has to be executed at this point in time.

Very interesting with respect to MABS is the distributed discrete event simulation, because the basic entities in distributed simulation, called logical processes (LP), represent active objects with a control flow of their own. They are therefore to some extent comparable to autonomous agents. In *synchronous* process simulation, a centralized or decentralized global clock is used to coordinate the LPs. All processes of the same simulated system share the same time, called simulation time. This simulation time is set to a predefined value at the beginning of the simulation run and is only advanced during the simulation proceeds. At each point in time defined by a process, it gains the possibility to execute. This can include further communication with other processes that were not activated for this simulation time. On the contrary *asynchronous* process simulation allows the presence of events occurring at different simulated times that do not affect one another. The problem that is associated with asynchronous LP simulation is the chance of causality errors. Approaches that allow the advancement of time even though causality errors may happen are called *optimistic* in contrast to *conservative* methods executing only conflict free events at any point in time. Anyhow, the techniques from distributed simulation cannot be used without adaptation to the special requirements of MABS, e.g., Theodorpoulos and Logan say "[...] conventional distributed simulation techniques cannot easily be applied to the problem of modelling the interaction of a system of autonomous components" [Theodoropoulos and Logan, 1999, p. 1]. Ferscha provides an excellent overview of distributed simulation mechanisms [Ferscha and Chiola, 1994].

In OOS the participating entities are objects and for reasons of simplicity it is in many cases sufficient to have a single control flow in the simulation system. Considering these characteristics it becomes comprehensible that utilizing simple non-distributed DES methods for OO simulation is adequate. Besides this important difference some further conspicuous distinctions to OOS can be noticed when analyzing the protagonists. Agents are autonomous, pro-active acting entities that use message passing as communication paradigm and are modelled in terms of mentalistic attributes whereas objects are purely reactive, use method invocation and are modelled in terms of attributes and methods.

After having argued why agent-based simulation is important and which different paradigms form the basis of MABS, the context of this paper is presented.

3 The Agent.Enterprise and Agent.Hospital Scenarios

The SPP 1083 is a research programme supporting research in the field of agent technology for business applications. Two domain areas are covered by the participating projects: Manufacturing logistics and hospital logistics. Initially the projects have developed stand-alone prototype multi-agent systems focussing on different problems in the field of supply chain management and hospital logistics respectively. In the last year two initiatives have been formed to bring together the results of the single projects and integrate the developed prototypes for the creation of large-scale distributed systems. The Agent.Enterprise [Frey *et al.*, 2003] and Agent.Hospital [Kirn *et al.*, 2003] initiatives make up the motivation for the work on MMAS. Two scenarios have been developed to integrate the independent projects into two superordinated systems using generalized process flows.

The Agent.Enterprise scenario is an inter-enterprise multi-level supply-chain scenario. Currently, five research projects are involved in the development of Agent.Enterprise. Two of these research projects focus on supply chain aspects: SCM scheduling as well as tracking and tracing of supply chains. Three projects deal with application of agent technology in the participating enterprises of the supply chain with a special focus on integration of process planning, reliability and robustness, and batch production of semiconductors. Necessary processes in supply chains were analysed and modelled. Additionally the interfaces between involved systems were designed.

In the context of the hospital logistics working group with six participating projects an extensive empirical funded model called Agent.Hospital is being developed. This model is an open framework with numerous different healthcare actors and consists of detailed partial models of the healthcare domain. It enables the examination of modeling methods, configuration problems as well as agent-based negotiation strategies and coordination algorithms. The working group also deals with the integration of different partial hospital logistics models created by the participating projects. One important step for the integration was the definition of numerous different gateways between all these models and the merge into a conceptual overall scenario consisting of the basic process flows. Relevant organizational structures, processes and necessary data models were analysed, formalized and modelled at several hospitals.

To establish the connection between the different prototypes of the two scenarios, each involved MAS has a gateway agent which is able to communicate with agents of its own MAS as well as other gateway agents within the Agent.Enterprise or Agent.Hospital scenario [Krempels *et al.*, 2003]. Communication between gateway agents is based on standard FIPA protocols. While a common supply chain ontology tailored for the Agent.Enterprise scenario has been developed, the Agent.Hospital scenario currently uses different ontologies for communication between different MAS. Since the systems reside on agents platforms all across Germany, Agent.Enterprise and

Agent.Hospital utilise the Agentcities[2] infrastructure including directory facilitator and exchange of FIPA ACL messages using the HTTP MTP. Current developments are dealing with enhancements in synchronisation of distributed MAS and interfaces to further MAS in order to make Agent.Enterprise and Agent.Hospital open platforms e.g. to test MAS in manufacturing or MAS in healthcare as part of the Agentcities network. It is important to point out, that it was and still is an essential development goal of Agent.Enterprise and Agent.Hospital, to support an open and extensible agent-infrastructure for the manufacturing and healthcare domain.

From this work the following general requirements of MABS regarding large-scale agent systems are derived. Firstly, the participants of the simulation are not necessarily agents of one platform, because different subprojects may decide in their own responsibility what agent platform is best suited for their needs. Therefore the communication among the participants and the simulation system has to be standardized in some way with the different agent platforms. Secondly, the subsystems deal with different cut outs of the superordinated system resulting in events irregularly distributed over time. The simulation should be able to handle this distribution effectively. Thirdly, as a lot of work has already been done to implement the participating MAS, it should be possible to adapt existing systems to the simulation with little effort.

4 Realization

In this section the design and implementation of the mechanism for synchronization in the distributed architecture of Agent.Hospital and Agent.Enterprise is presented. Apart from the general requirements mentioned in Section 3, the design has to consider the following technical properties of the distributed multi-agent architecture:

- Agents may be newly created and destroyed. Therefore the number of participants in the synchronization mechanism may change dynamically. When a new agent is created, the synchronization mechanism will not know about its existence, before it has announced its interest in taking part in the synchronization. It has to be made sure, that subsequent activities are not triggered, until all interested parties have been given opportunity to appropriately announce their scheduled activities.
- Inside each agent there may be a number of parallel executing tasks. Each task may independently require synchronization with other agents. Therefore, an agent may require a separate time point for each of its tasks and the mechanism has to provide ways for an agent to announce more than a single time point.
- Agents may receive messages from other agents while waiting for time points to occur. Therefore a task of such a notified agent must have the possibility to change its next wakeup time, due to new information contained in the message.

[2] `http://www.agentcities.net`

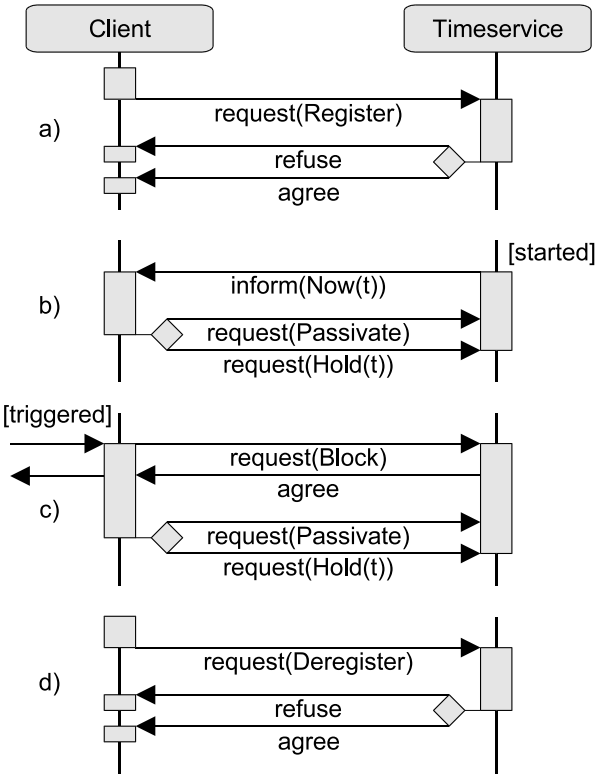

Figure 1: Time Service Protocol

4.1 Design

The above mentioned requirements are best met by process-oriented simulation, using a global list of time points for scheduled activities. In synchronous process-oriented simulation, the processes (agent tasks in this case) send to the coordinator a *Passivate* message to indicate that they have no scheduled activities, or a *Hold(t)* message with the time of the next activity that needs to be scheduled. These messages tell the coordinator, that the process has finished its actual activity and the next activity can be scheduled. Therefore the coordinator iteratively removes the next entry from this list, advances the clock, and informs the corresponding process that the time point is reached. Then the coordinator will wait, until that process has answered again.

The coordinator managing the global list of time points can be implemented as an autonomous agent and therefore fits naturally into the MAS world. While the drawback of a synchronous approach is maybe less efficiency, it "considerably simplifies the implementation of correct simulations by avoiding problems of deadlock and possibly overwhelming message traffic" [Ferscha and Chiola, 1994].

The client/server protocol of the time service agent is depicted in Fig. 1. The four different interaction possibilities are denoted by the characters *a* to *d*. In the initialization phase (*a*), agents (more precisely agent tasks) may *Register* and will receive an *agree* message, if they are not already registered. When a participant terminates (e.g. when a patient leaves the hospital) it requests a *Deregister* (*d*), receiving a *refuse* if the participant was not registered. The other two interactions (*b* and *c*) may happen repeatedly

during simulation runs.

When the simulation run is started, the initial time (*Now*) is sent to all participants (*b*). New participants that register while the simulation is running will immediately receive the *Now* message with the current simulation time. While the simulation is running a participant will continuously get these so called wake-up calls whenever its registered point in time is reached. The participant is now supposed to execute its actual activity and subsequently send back a message when it is finished. Either the process announces the point in time (t) for its next activity by submitting a *Hold(t)* request, or it currently has no activities to be scheduled and therefore submits a *Passivate* request.

Another case that can occur during the simulation is that the active process communicates with another process (*c*). The new information that arrives at the waiting process may lead to new activities in this process and a new activation time. Therefore the triggered process can decide to stop the advancement of the global time by sending a *Block* request to the time service. The service removes the time entry of the process and acknowledges this by sending an *agree* message. Now the coordinator has to wait until both processes (the triggered and the triggering) declare that they are finished by sending a *Passivate* or *Hold(t)* request. It is noteworthy that this design differs considerably from the ordinary process oriented simulation where it only depends on the active process when the coordinator may advance the time. Therefore a triggered process has to notify the calling process when it has finished its activities. This has to be done is a nested way, when the triggered process sends messages to further processes. In an agent scenario this traditional kind of design is undesirable, because it violates the agent's autonomy concept and leads to awkward interaction protocols that resemble method calls.

In addition to the client/server protocol the time service implements an administrative interface with commands that can be submitted via the FIPA-Request interaction protocol [Foundation for Intelligent Physical Agents, 2002]. The simulation is initialized with the *Init* command by providing the start time. It is started with the *Start* command or alternatively in single-step or slow-motion mode with the *Step* or *Slow* commands, where for slow-motion an optional speed argument can be supplied. During the execution, the simulation can be paused with the *Pause* command or switched to single-step or slow-motion mode. The *Continue* command puts it back to normal mode. The simulation is terminated by the *Stop* command. The *Pause* and *Stop* commands accept an optional argument indicating the simulation time at which the command should take effect. Otherwise the commands are executed immediately.

4.2 Implementation

For the communication between the time service and the processes, a so called TimeService ontology has been conceived, which contains on the one hand information about the agent actions, what an agent wants another agent to perform, and on the other hand the domain concepts. The agent actions correspond to the above explained client requests and administrative commands and the modelled domain concepts describe things like dates and points in time. The ontology was defined in the ontology modelling tool Protégé [Grosso *et al.*, 1999].

```
<goals>

 <achievegoal name="ts_register">
  <parameter name="participant" class="String.class"/>
  <parameter name="timeservice" class="AID.class" optional="true"/>
  <parameter name="leasetime" class="Duration.class" optional="true"/>
 </achievegoal>

 <achievegoal name="ts_hold">
  <parameter name="participant" class="String.class"/>
  <parameter name="nexttime" class="AbsoluteTimepoint.class"/>
 </achievegoal>

 <achievegoal name="ts_passivate">
  <parameter name="participant" class="String.class"/>
 </achievegoal>

 <achievegoal name="ts_block">
  <parameter name="participant" class="String.class"/>
 </achievegoal>

</goals>
```

Figure 2: Client Goals (digest)

The time service has been implemented as a FIPA-compliant agent using the JADE agent framework [Bellifemine *et al.*, 1999] and the BDI-extension Jadex [Pokahr *et al.*, 2003]. JavaBeans, used in JADE to construct the ACL-message contents, have been generated directly from the ontology using a code generator tool [van Aart *et al.*, 2002]. For time service clients (agent tasks) a generic pluggable agent module has been implemented, that handles the communication with the time service. This client module provides among other things several typed goals (see Fig. 2) that can be used to easily initiate actions with the time service. These actions comply to the needed client tasks taken from the time service protocol such as register, hold, passivate and block. To use the time service functionality a client only has to instantiate such a goal, provide it with the needed parameters, dispatch it to the BDI-system and wait for the result.

As the service is FIPA-compliant, non-JADE agents can also participate in the simulation as long as they conform to the FIPA standard [Dale and Mamdani, 2001]. For the ease of use a graphical user interface has been developed (see Fig. 3). This interface allows the service to be monitored and controlled manually. Additionally, the visual representation of the global time history and the list of participants with current activation points is useful for testing and debugging purposes.

4.3 Practical Application

The time service has already been integrated into the MedPAge project [Paulussen *et al.*, 2003; 2004], which is a subproject of the Agent.Hospital initiative and addresses the topic of treatment scheduling for patients in hospitals. The time service allows to test the MedPAge hospital model with a number of different functional units, and different arrival rates for patients. In the MePAge scenario the arrival of new patients, represented as patient agents, and the execution of treatments, performed by the resource agents, needs to be synchronized. Practical explorations have shown no difficulties simulating a reasonably sized hospital model with several wards. Resource agents always synchronize their activities, while patient agents only

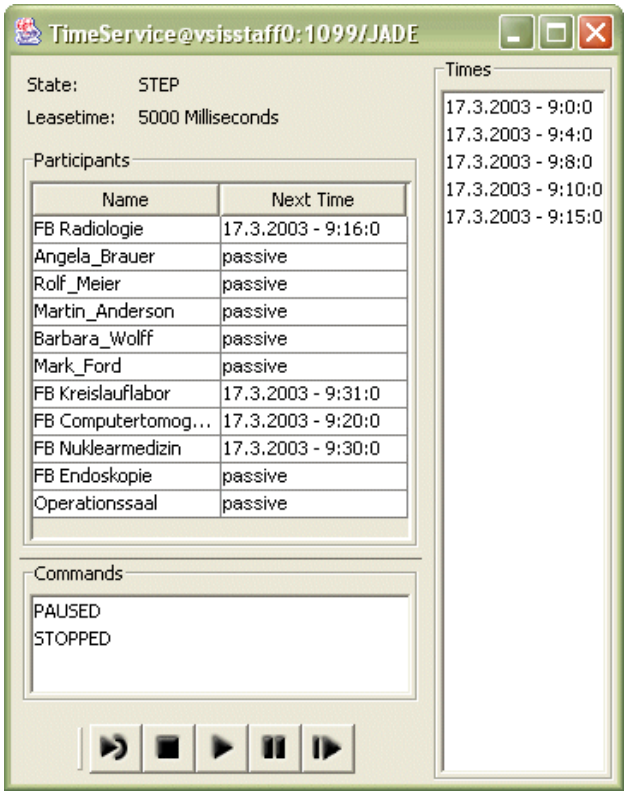

Figure 3: Time Service GUI

need to synchronize when they are actively negotiating. A first scalability analysis using a profiler tool shows that even when creating a large amount of patients at once (e.g. 50), which instantly engage in concurrent negotiations, the synchronization cost does not increase significantly but remains approximately at 5-10% of the overall execution time.

5 Related Work

The term simulation with respect to multi-agent systems is not defined unambiguously. There are at least two different research areas that are further explained by outlining some of their typical representatives.

On the one hand there exist simulation systems that claim to be generic and are used in most cases to evaluate artificial intelligence aspects of agents. E.g., James [Uhrmacher and Schattenberg, 2001] is a sophisticated Java-based simulation tool designed with the objective to support the flexible construction of experimental frames for multi-agent systems. The system was hitherto especially used to compare different planning strategies under the premise that the agents planning time is tracked by the system [Schattenberg and Uhrmacher, 2001]. Another simulation middleware is MPADES [Riley, 2003] which is a hybrid system: Discrete events are used for the agent's interactions with the world, while the underlying world model is assumed to be continuous. The system focuses on the consideration of latencies for the agent's internal sense, think, and act processes. It is independent from a definite world model, agent architecture and programming language. Its objective it is similar to the more basic MESS system [Anderson, 1997], which tracks the computation of agents at the level of LISP instructions.

On the other hand artificial life facets are investigated by simulation systems. They allow the simulation of complex societies, e.g., for biological, economical or sociological experiments. For example SeSAm [Klügl, 2001] is a highly developed Java-based simulation system that allows the definition of agent behaviours in a graphical UML-statechart [UML 1.4, 2001] like fashion. Another platform is Swarm [Swa, 2000], which concentrates on the simulation of complex adaptive systems. In contrast to other systems the basic unit of simulation is a swarm, a collection of agents executing a schedule of actions.

So why can't we take one of the above mentioned systems to easily synchronize agent actions in a given multi-agent system? All the systems (except SeSAm) are not capable of exchanging FIPA-compliant messages, what is essential when considering the fact, that different multi-agent platforms have to be integrated into one simulation. Furthermore nearly all systems (except MPADES) constitute a definite intra-agent architecture and force the agents to be implemented on a special platform. Even worse, some of the tools restrict the agent implementation to a given proprietary programming language. In addition, systems from the artificial life category often use a time-sliced simulation model, which is inefficient and inappropriate to synchronize agents that exhibit irregularly distributed points in time.

6 Concluding Remarks

The objective of this paper is to describe a middleware service component for the simulation of process flows in coupled heterogeneous multi-agent systems. Under consideration of the special requirements imposed by the Agent.Hospital and Agent.Enterprise scenarios, a centralized approach with one coordinator process that holds the current simulation time and the global list of time points was chosen. Although a centralized coordinator might be a bottleneck or single point of failure, it allows the participating projects to easily integrate their existing solutions, independently from the chosen agent platform. Furthermore, the approach helps the system designer to understand the systems runtime behaviour because the conservative approach only generates correctly ordered wake-up calls. The various administrative commands like slow and step mode that are available through a user interface support debugging as well.

Looking back at the three requirements derived from the scenarios in Section 3, one can see how they are addressed by design and implementation choices. First, the communication is FIPA-compliant and therefore client and service can be easily decoupled and implemented on different platforms. Secondly, the use of discrete event-based simulation techniques allows the events to be arbitrarily distributed over time without performance loss. This design decision is different to most existing agent-based simulation frameworks that utilize time-driven approaches to avoid the complexity of a distributed environment. And thirdly, the basis for an easy integration process was provided on the design level by choosing a conservative cen-

tralized simulation approach and on the implementation level by providing a pluggable agent module to handle timing aspects.

First experiments in the MedPAge project show that the design and implementation of the time service is well suited for synchronizing activities in distributed multi-agent systems. To extend this work in order to simulate and test the superordinated process flows of the Agent.Enterprise and Agent.Hospital scenarios, the interactions between the gateway agents have to be revised, and the remaining projects have to integrate the synchronization mechanism into their prototypes. This integration process will demand the usage of time service clients implemented on different platforms and will hopefully lead to further results regarding the service reliability and scalability when used in the large.

Acknowledgements

This work is funded by the DFG German priority research programme 1083 *Intelligent Agents in Real-World Business Applications.*

References

[Anderson, 1997] S. D. Anderson. Simulation of multiple time-pressured agents. In S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, editors, *Proceedings of the Winter Simulation Conference*, pages 397–404, 1997.

[Bellifemine *et al.*, 1999] F. Bellifemine, G. Rimassa, and A. Poggi. JADE – A FIPA-compliant agent framework. In *4th International Conference on the Practical Applications of Agents and Multi-Agent Systems (PAAM-99)*, pages 97–108, December 1999.

[Dale and Mamdani, 2001] J. Dale and E. Mamdani. Open Standards for Interoperating Agent-Based Systems. *Software Focus*, 1(2), 2001.

[Davidsson, 2000] P. Davidsson. Multi agent based simulation: Beyond social simulation. In *Multi Agent Based Simulation*. Springer Verlag LNCS series, Vol. 1979, 2000.

[Ferscha and Chiola, 1994] A. Ferscha and G. Chiola. Accelerating the evaluation of parallel program performance models using distributed simulation. *LNCS 794*, pages 231–252, 1994.

[Foundation for Intelligent Physical Agents, 2002] Foundation for Intelligent Physical Agents. FIPA Request Interaction Protocol Specification. Document no. FIPA00026, December 2002.

[Frey *et al.*, 2003] D. Frey, T. Stockheim, P.-O. Woelk, and R. Zimmermann. Integrated Multi-agent-based Supply Chain Management. In *Proc. of 1st International Workshop on Agent-based Computing for Enterprise Collaboration*, 2003.

[Fujimoto, 1999] R. M. Fujimoto. Parallel and distributed simulation. In *Proceedings of the Winter Simulation Conference*, pages 122–131, 1999.

[Grosso *et al.*, 1999] E. Grosso, H. Eriksson, R. W. Fergerson, J. H. Gennari, S. W. Tu, and M. A. Musen. Knowledge Modeling at the Millennium, 1999.

[Jennings, 2001] N. R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, April 2001.

[Kirn *et al.*, 2003] S. Kirn, C. Heine, R. Herrler, and K.-H. Krempels. Agent.Hospital - agent-based open framework for clinical applications. In *International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 2003.

[Klügl, 2001] F. Klügl. *Multiagentensimulation - Konzepte, Werkzeuge, Anwendung*. Addison Wesley, 2001. (in German).

[Krempels *et al.*, 2003] K.-H. Krempels, J. Nimis, L. Braubach, A. Pokahr, R. Herrler, and T. Scholz. Entwicklung intelligenter Multi-Multiagentensysteme - Werkzeugunterstützung, Lösungen und offene Fragen. In *Informatik 2003 - 33. Jahrestagung der GI*, volume P-34 of *Lecture Notes in Informatics (LNI)*, pages 31–46, 9 2003.

[Page, 1991] B. Page. *Diskrete Simulation: eine Einführung mit Modula-2*. Springer Verlag Berlin, 1991.

[Paulussen *et al.*, 2003] T. O. Paulussen, N. R. Jennings, K. S. Decker, and A. Heinzl. Distributed patient scheduling in hospitals. In *IJCAI'03 Proceedings*, 2003.

[Paulussen *et al.*, 2004] T. O. Paulussen, A Zöller, A. Heinzl, A. Pokahr, L. Braubach, and W. Lamersdorf. Dynamic patient scheduling in hospitals. In *Agent Technology in Business Applications (ATeBA-04)*, 2004.

[Pokahr *et al.*, 2003] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: Implementing a BDI-Infrastructure for JADE Agents. *EXP – in search of innovation*, 3(3):76–85, 2003.

[Riley, 2003] P. Riley. MPADES: Middleware for parallel agent discrete event simulation. In G . Kaminka, P. Lima, and R.Rojas, editors, *RoboCup-2002: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2003.

[Schattenberg and Uhrmacher, 2001] B. Schattenberg and A. Uhrmacher. Planning Agents in James. In *Proceedings of the IEEE, Vol.89, No.2*, pages 158–173, 2001.

[Swa, 2000] Swarm Development Group. *A tutorial introduction to Swarm*, 2000. `http://www.swarm.org/csss-tutorial/frames.html`.

[Theodoropoulos and Logan, 1999] G. Theodoropoulos and B. Logan. A framework for the distributed simulation of agent-based systems. In H. Szczerbicka, editor, *13th European Simulation Multiconference (ESM99), Warsaw, Poland, June 1-4*, pages 58–65, 1999.

[Uhrmacher and Schattenberg, 2001] A. Uhrmacher and B. Schattenberg. Agents in Discrete Event Simulation. In *Proc. of the ESS'98, Nottingham*, pages 129–136. SCS Publications, 2001.

[UML 1.4, 2001] Object Modeling Group. *UML Specification, version 1.4*, September 2001.

[van Aart *et al.*, 2002] C. van Aart, R. Pels, G. Caire, and F. Bergenti. Creating and Using Ontologies in Agent Communication. In *2nd Workshop on Ontologies in Agent Systems*, 2002.