Proceedings of the 2009 Winter Simulation Conference
M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin and R. G. Ingalls, eds.

SCALABILITY IN DISTRIBUTED SIMULATIONS OF AGENT-BASED MODELS

Dirk Pawlaszczyk
Steffen Strassburger

School of Economic Sciences
Ilmenau University of Technology
Helmholtzplatz 3
98693 Ilmenau, GERMANY

ABSTRACT

Research on systems of autonomous agents, called multiagent systems (MAS), has received much interest in the domain of (distributed) artificial intelligence in recent years. MAS are most suitable for the development of distributed applications within an uncertain and dynamically changing environment (Logan 2005). For validation of such systems agent based simulation is a new modeling paradigm not limited to systems which qualify as MAS by default. The focus of the work presented here is on scalability aspects of simulation environments for agent based simulations. Scalable solutions are required, as complex models require the capability to simulate hundreds or more complex deliberative agents. This is a capability which is often lacking in existing simulation environments for agents. We investigate different aspects which influence scalability and present a solution for enabling a scalable and efficient distributed simulation of agent-based models based on an adapted optimistic synchronization protocol which limits the level of optimism by using knowledge about agent interaction patterns.

1 INTRODUCTION

Simulation in general is the imitation of a systems behavior and structure in an experimental model to reach findings which are transferable to reality. In multiagent-based simulation (MABS) real world systems are modeled using multiple agents. A software agent is typically defined as a program that acts autonomously, communicates with other agents, is goal-oriented (pro-active) and uses explicit knowledge. The modeled system emerges by interaction of the individual agents as well as their collective behavior. Agents typically send messages with respect to some communication protocol, e.g. as defined by the standards of the Foundation for Intelligent Physical Agents (FIPA) discussed in section 2.1.

Agent-based modeling and simulation is an appropriate tool for domains characterized by discrete decisions and distributed local decision makers. MABS is a valid method if we wish to understand the evolution of a distributed system affected by non-linear dynamics. As we wish to resemble such complex real world systems, there is an increasing demand to simulate large models with many agents. With growing complexity and increasing number of simulated entities, the scalability of a simulation environment becomes a crucial measure of its ability to cope with the complexity of the modeled system.

With the objective of industrial application of MABS it becomes demanding to investigate scalability issues. Only a few contributions exist that deal with the problem of scalable agent-based simulation. The objective of the research presented here is to discuss an approach for scalable simulation which takes advantage of specifics of the agent-based simulation approach effectively. In specific, we introduce an optimistic time synchronization protocol for MABS which effectively reduces the risk of "too much optimism". This is achieved by leveraging external information contained in the interaction protocols used for communication between agents.

The remainder of this paper is structured as follows: We first discuss some foundations of agent-based simulation (section 2). We then outline commonly used notions of scalability and examine what scalability means within the context of agent-based simulation (section 3). We also motivate why we are convinced that only distributed simulation offers a solution for scalability. In section 4 we introduce the concept of our scalable time synchronization service for distributed agent based simulations which explicitly leverages agent specific interaction patterns. With respect to the experimental results given in section 5 it is argued that parallel and distributed simulation can help to ensure scalable simulation of agent based models. Section 6 summarizes the paper.

2 FOUNDATIONS OF AGENT-BASED SIMULATION

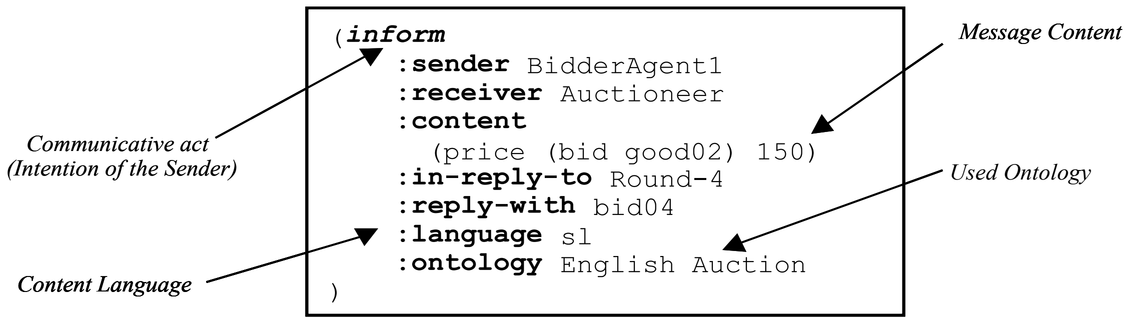
2.1 Agent Technology and Standards

The term “agent” can be defined in many ways depending on the application context. A very general definition describes an agent as “anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors” (Russel and Norvig 2003).

Other definitions require agents to show properties like reactivity, pro-activity, autonomy, and social skills. Further characteristics may include rationality, learning aptitude, and mobility (Wooldridge and Jennings 1995). Especially the capabilities for rational behavior and learning aptitude are often seen as prerequisites for intelligent behavior. Agents which adopt at least one of these concepts are often referred to as intelligent or cognitive agents. Further classifications for agents exists (e.g. reactive, deliberative, hybrid), but shall not be subject of this discussion. The focus of our considerations are deliberative agents that have a complex behavior and require extensive computation power.

When using agent-based design approaches it is often the interplay between multiple agents, which one is interested in. This leads to the term of multi-agent systems (MAS). An MAS is generally considered a system composed of multiple autonomous agents which can interact with each other. MAS can be used to solve or describe problems which are difficult or impossible to solve/describe with individual agents or a monolithic system. Examples include online trading and modelling social structures. The collective behaviour of an MAS is often a subject of investigation.

Agents are only capable of exchanging knowledge and interacting with each other when they use a common language spoken and understood by all agents. The Foundation for Intelligent Physical Agents (FIPA) is an organisation founded with the objective of creating a framework architecture for the interaction of heterogeneous agent systems. A central point of this effort is to suggest standards for the message-based communication of agents and multi-agent systems. The structure of a message is defined by the Agent Communication Language (FIPA 2002). Figure 1 gives a simple example for an ACL message which is part of an auction protocol.



```
(inform
   :sender BidderAgent1
   :receiver Auctioneer
   :content
     (price (bid good02) 150)
   :in-reply-to Round-4
   :reply-with bid04
   :language sl
   :ontology English Auction
)
```

Figure 1: Example for an ACL-Message

Most importantly for the further discussions, FIPA also defines specifications for standard interaction protocols (FIPA 2002a). These interaction protocols define typical sequences of messages or patters, how agents may interact. The resulting dialogs between agents always follow this same pattern. A simple example of this is shown in Figure 2. The figure depicts the possible lines of communication between an auctioneer agent and multiple bidder agents. The auctioneer first has to request bids from the participating bidders, before he can accept a proposal and inform the successful bidder. The possible message types exchanged within this interaction protocol and their sequence is independent from the actual message content. It is, for instance, completely irrelevant whether the auction concerns a pencil or a car.

Currently 13 different protocols are distinguished. By using a certain protocol the agent commits to react to requests of the protocol in the predefined format. Please note that FIPA also defines further aspects of the communication within and between multi-agent systems which are beyond the scope of this discussion.

2.2 Agent-based Simulation

The adoption of agent based approaches for modeling and simulation promises greater flexibility and better abstraction capabilities when describing the behavior of systems with many active (or “intelligent”) components. The term agent-based simulation (ABS) describes the modeling and simulation of real systems with the help of agents, which interact within a simulation model. In agent-based simulations agents can represent human actors, machines, transport carriers, etc.

As there is no standardized terminology, there are also other terms, e.g. “multi agent based simulation”, which are used synonymously. It should also be noted that agents are not only used as a metaphor for model building. Often they are also the actual subject of the simulation, i.e., the actual entity / system under investigation (Uhrmacher and Gugler 2000). In that case the simulation serves the purpose of analysing and designing multi-agent systems.

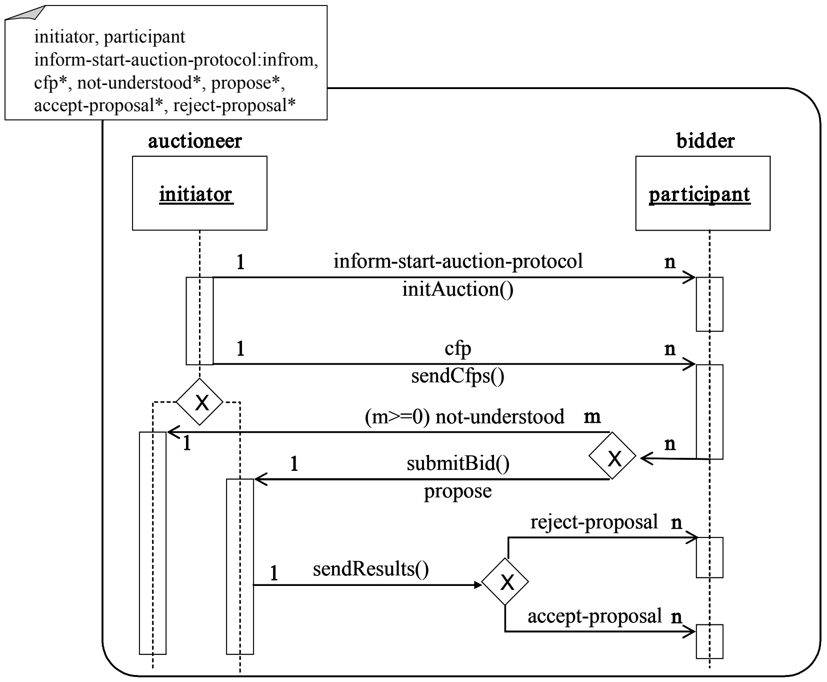


Figure 2: Example for an auction protocol

ABS is not a completely new approach (Davidsson 2000). Rather, it is influenced and builds upon existing paradigms like discrete event simulation (esp. in its process-oriented world view flavor) and object-oriented simulation.

There is a wide variety of development environments for agent-based simulations, mostly from academic sources. Current examples include Cougar, Farm, James II, Repast, Samas, Sassy, and many more. Most of them have an event based mechanism for advancing logical simulation time. The majority offers the option to run on a distributed architecture and implements conservative or optimistic time synchronization protocols. A small number of the distributed environments applies real-time driven time-advancement and synchronization protocols.

2.3 Distributed Simulation

Parallel and distributed simulation is often used as a paradigm for implementation ABS. However, agent-based modeling and simulation environments do not necessarily have to be build in a distributed fashion. Many implementations of agent-based models even lack typical properties generally associated with agents, like autonomy and pro-activity. Often, a simulation model using an agent-based model design is not in fact implemented with agents. Rather, these "agents" are often simulated on a serial machine, i.e., using a traditional sequential simulation algorithm (Uhrmacher and Gugler 2000). This may be useful up to a certain extend where agent-based metaphors are used for describing new modelling paradigms and agents exhibit a very simplistic behaviour. In that case the lacking scalability of a serial machine is not an issue which really limits the number of agents which can be simulated.

However, when it comes to the potential that a parallel execution of agents might actually offer, e.g. in terms of performance advantages of agents with complex planning or learning algorithms, or in the implementation of truly autonomous behaviour, these approaches fall short of reaching the actual potential offered by the agent-based modelling paradigm. Drougoul et. al (2002) have addressed this divergence in their controversial paper "Where are the agents?".

In the light of these facts, we want to discuss the basics, the potential and the benefits of using parallel or distributed architectures for agent-based simulation. Distributed simulation (DS) is a technology that enables a simulation program to be executed on distributed computer systems (Fujimoto 2000). There can be many reasons and objectives for using distributed simulation. They include the desire to obtain s*peed-up* (i.e. a reduction in the execution time of the simulation) as well as the need to couple heterogeneous and possibly geographically distributed simulation components. It may also be the case that the complexity of a model (e.g., its memory and computing requirements) simply make the model unsuited for monolithic model execution (Mustafee et. al 2006). In the following we will argue that the latter may constitute a very important reason for using distributed simulation when simulating extremely large agent-based models (i.e. models with thousands of deliberative agents or more).

Research in parallel and distributed simulation has a long history. One of its main fields of investigation has always been the problem of time synchronization, which arises when multiple simulation models are executed in a distributed fashion. In that case, each simulation model has its own simulation clock and time advances of these local simulation clocks have to be synchronized. The synchronization protocol types developed as a result can be classified into the two main categories of conservative and optimistic protocols. For in-depth background information, please consult Fujimoto (2000).

Conservative protocols implement mechanisms that prevent a member of a distributed simulation from processing messages out of time stamp order, thus maintaining strict causality. Conservative synchronization protocols are rather easy to use and implement, but their performance depends highly on a value called Lookahead. Lookahead is a guarantee that a simulation will not generate any messages with a time stamp smaller than its current time plus the value of Lookahead. If the current time of a simulation is T, and its Lookahead is L, any message generated by the simulation must have a time stamp of at least T+L.

Lookahead is hard to extract and always depends on application context. For agent-based simulations, the situation comes close to the worst case scenario, as their interaction protocols often require a Lookahead of zero. In this case, conservative synchronization protocols almost completely inhibit any parallel execution within the distributed simulation.

Optimistic protocols do not impose the requirement to process events in strict time stamp order. Simulations using optimistic synchronization can process received messages although there may be future messages with a smaller time stamp. To maintain causality, these approaches detect and recover from causality errors, which may be introduced by processing events before it is safe to proceed. The major advantage of these approaches is that they allow the exploitation of parallelism in situations where it is possible that causality errors might occur, but in fact they do not occur. The Time Warp protocol is an example of an optimistic synchronization mechanism.

Optimistic protocols are more complex to implement than conservative protocols, but the following statement of Fujimoto (2000, p. 174) certainly holds much truth: “If one’s goal is to develop a general purpose simulation executive that provides robust performance across a wide range of models […] optimistic synchronization offers greater hope.”

3 SCALABILITY OF SIMULATIONS

3.1 General Considerations

To simulate large scale, distributed systems of deliberative software agents is a complex endeavor for hardware and software. With growing complexity, the scalability of a simulation environment becomes a crucial measure (Chaturvedi 2004). Scalability issues are not solved just because a system is executed on a distributed computing architecture (Law 1998). It is a common assumption that the inherent distribution of multi-agent systems can also be used for scale-up respectively speed-up of the simulation. However, practical experience shows that this is not necessarily true.

In general, scalability can be described as the property of a system compared to some other property of the same system. For example, we could observe the runtime of a certain simulation run compared to the number of simulated entities. Several researchers tried to prove the scalability of a particular application by simulating more entities than some other application, to draw the conclusion, that their system is more “scalable”. In fact, such studies do not have much expressiveness, since scalability is always a relative value dedicated to a certain system. According to Nicol we define scalability as “…how the performance of a certain application behaves as the application problem size increases and the parallel architecture executing it increases” (Nicol 1998, p. 7). Hence, if we wish to decide whether a simulation model will scale or not, we have to consider the architectural as well as the algorithmic side of the simulation system.

We can choose different performance metrics to measure the capability of our simulation model to scale. To reduce problem complexity, we normally concentrate on a single facet of the system architecture or a certain software aspect to decide whether a system scales or not. The usage of distributed simulation to reach scalability requires a model to be partitioned among the available processors and simulated with a parallel algorithm that synchronizes the different partitions. To find a partition that balances workload efficiently - even if the simulation model grows - requires a suitable partitioning strategy as well as adaptive load balancing schemes that yield scalable behaviour. Accordingly, the effectiveness of a distributed simulation again generally depends on three elements: partitioning, load balancing and synchronization. Nevertheless, the distribution of the simulation model leads to an additional challenge: Correctness of experimental results generated by a distributed simulation run mainly depends on the accuracy of the underlying synchronization mechanism. At the same time, synchronization overhead may affect scalability negatively.

Although there are other factors which influence performance and scalability (e.g. effective data distribution mechanisms), we limit the considerations of this paper to synchronization issues, as they the key factor for our applications.

3.2 Related Work

The use of PDES for agent-based modelling is a relatively new idea (see for instance (Logan et. al 1999), (Uhrmacher 2000), and (Riley 2003)). A focus with this has been to model situated agents like robots moving about a spatial 2- or 3-dimensional environment and to address questions about spatial representation and management of shared space. Our approach is somewhat different, in that it is not meant to give an explicit support to spatially situated agents. From our point of view the environment of an agent consists of other agents and explicit environmental entities like resources, whereby interaction is the central point. To simulate the process flow within multi-agent systems we use a message-passing approach. Agents communicate directly. Since we need to synchronize participating agents with respect to global simulation time each message event refers to some send and receive time, just like in classic PDES simulation.

Our literature survey outlined in this section indicates that only a small number research efforts have explicitly focused on the area of scalable agent-based simulation so far. MACE3J was one of the first implementations of Java-based MAS (Gasser et. al 2004). It is reported that MACE3J has been run with up to 5000 agents on a shared memory system to prove scalability; admittedly, agents have not exchanged any messages in this test. For the SESAM agent testbed, a test with 10.000 agents on a single computer attests acceptable performance (Oechslein 2004). Nevertheless, SESAM does not support distributed simulation. Accordingly, its capability to scale is limited by default. Theodoropoulos and Logan (1999) propose a framework to speed up simulation of MAS by adopting techniques from the parallel simulation community such as optimistic synchronization. Most of their work focuses on questions about spatial representation and management of shared space.

Another simulation toolkit that is designed for large scale testing of distributed multi-agent systems is the JAMES agent testbed (Uhrmacher 2000). It uses a centralized server architecture for tracking and tracing of simulated agents using a centralized environment model. Again, there are no concrete statements concerning scalability of this system. Popov et. al (2003) describe a parallel sequential simulation approach to simulate 10^6 agents to capture the behavior of web users. This vast number of simulated agents is reached by keeping the agent implementation static. Furthermore they are using a relatively weak notion of agency, and do not consider deliberative agent structures.

Furthermore, there are some contributions in place concerning distributed agent-based simulation using the High Level Architecture (HLA). HLA_AGENT for example introduces support for distributed simulation to the SIMAGENT toolkit (Lees et. al 2003). These efforts do not focus on scalability and rely on the options given by HLA.

To summarize, although ABS has received a lot of attention in recent years there are only a few contributions that deal with the problem of simulating large scale agent-based systems required for industrial applications.

4 OPTIMISTIC SYNCHRONISATION OF DISTRIBUTED AGENT-BASED SIMULATIONS

In our research we have developed a time service to enable efficient simulation of large-scale agent based models. To do so, we have added PDES-functionalities (in specific: a time service) on top of an existing agent middleware to get support for optimistic simulation. The used middleware is the Java Agent Development Environment (JADE), a generic framework for development of agent based applications (JADE 09). JADE is widely used in academia. Since JADE is compliant to the FIPA standard (see section 2.1), a high degree of interoperability is guaranteed. Moreover, the JADE messaging sub system scales well, even for heaviest message traffic (Vitaglione et. al 2002). JADE provides many built in features like remote method invocation (RMI), serialization, and agent management tools that allow a distributed model to be easily developed. Because the program is written in Java the implementation is portable across a wide range of platforms.

Applying conventional PDES models and techniques to MABS is non-trivial, since we have to consider specifics of agent technology like a particularly high communication demand and dynamic topologies, without degrading performance. Our time service employs some novel techniques. At the heart of the simulation executive there is a new optimistic synchronization algorithm that provides a solution to cases where the traditional optimistic synchronization fails (or poorly performs) because of too much optimism. This is a typical situation in agent-based simulation. The algorithm allows the usage of external information (from the FIPA interaction protocols) to throttle the degree of optimism in distributed agent based simulations.

Furthermore we have implemented a new and efficient decentralized scalable algorithm for computation of GVT (global virtual time) taking into account transient messages without using blocking barriers. Other basic features of the simulator include automatic state saving, repeatability of simulation runs using a tie-breaking algorithm and simulation support for FIPA compliant agent models. All PDES-functionalities were added to the JADE middleware as platform services using the standard plug-in concept of the agent-framework. This enabled us to test the feasibility of the newly developed algorithms.

Programming simulation models with the simulator is straightforward. Individual agents are programmed by the application developer using the standard agent-based sense-think-act paradigm. Agent processes are autonomous in the sense that they hold and manage their own events, and are optimistic in their event processing. State saving and synchronization of agent processes is done in the background.

4.1 Optimistic Processing

As discussed in section 3, a scalable architecture by default requires it to be composed of multiple processors/computers, so that the number of used resources can grow as the problem size increases. This justifies the application of distributed simulation for our agent-based simulations. Searching for a general purpose synchronization technique we have disqualified conservative approaches because of the inherent zero lookahead requirements in many agent interaction protocols. Hence, optimistic synchronization is the most promising approach, because it (at least theoretically) provides enough performance for a wide range of models. However, within pure Time Warp implementations, a common problem can be caused by rollback cascades as there are no constraints on the relative distance between logical processes (LPs). One can call this a problem of "too much optimism". Every LP processes events almost independently of the progress of other LPs' timelines. Consequently, the probability of incorrect computations (i.e. causality errors) is very high. If the time to perform a rollback is high, i.e. many states have to be rolled back, the performance of the simulation rapidly decreases.

The straight-forward solution that our approach suggests is to limit the level of optimism in the applied time warp protocol by using extra-knowledge extracted from the used agent interaction protocols. Communication following these interaction protocols is one of the key features in agent technology.

Messages are sent out from a sender to one or more receiver(s). Messages are encoded in an *Agent Communication Language (ACL)*, an external language that defines the intended meaning of a message by using performatives. A series of messages produces a dialog. A dialog normally follows a predefined structure – the *Interaction Protocol* (IP) (see section 2). The FIPA *Request Interaction Protocol* for example allows an agent to request another agent to perform some action. The responding side needs to decide whether to accept or to refuse the request. In any case, the message receiver has to respond. Even if the receiver cannot interpret a message, the specification prescribes to send at least a not-understood message.

We exploit this characteristic: the communication almost always follows a known sequence of messages. In normal Time Warp every new event is immediately sent over the network to its corresponding receiver where it can be executed immediately. Our approach leverages delays on the event execution based on information about the current state of the interaction protocol. Instead of immediately processing every incoming event message the event is delayed using an adaptive rule:

Definition 1 (wait for rule): Given agent a_1 which has sent a message m_1 to agent a_2, and assuming that there is at least one valid required reply m_2 for m_1, the rule "wait for" is defined as follows: If the expected reply message was not received yet, agent a_1 must wait for this particular message, before going on to process the next message.

This rule basically requires agents to wait for a reply if a communication as part of an interaction protocol has been started. In the *FIPA-request-protocol* for example, if an agent has agreed to do something he automatically commits himself to send an *inform-done-message* as soon as he has finished the task. If he did not succeed he has to send a *refuse-message*. In any case the agent always has to reply. Accordingly, for every message m_i which is received while agent a is waiting for message m_k from a sender different to the sender of m_i, this message is buffered. The execution of m_i is delayed.

In figure 3 an example is given, to demonstrate the effect of delayed execution. The events in this example have distinguishable time stamps (expressed as indices) only for reasons of clarity. The suggested synchronization protocol can handle simultaneous events just in the same way.

In the example, the immediate execution of event e_{19} from a_3 normally would cause the agent process a_1 to rollback when message e_{12} is received at some later point in time. With the new policy in place this situation can easily be avoided. Instead of immediately processing message e_{19}, agent a_1 has to wait for the reply message from agent a_2. This is because there is an external knowledge that the active interaction protocol will sooner or later require a_2 to send a message "FIPA: inform-done" (expressed as e_{12}). The wait-for-rule basically formalizes this behaviour. Therefore event e_{19} has to be buffered thus preserving the relative event order. This approach minimizes the potential for incorrect execution of events as far as knowledge from the interaction protocols can be extracted.

The fact that the execution of some message/event is delayed for some time does not mean that the agent process is completely blocked. Rather, it can still receive other messages and perform other tasks which do not advance simulation time beyond the message that is being waited for. The process of waiting for a certain message could be interpreted as a conflict to the asynchronous nature of agent execution, but in fact, it is not a contraction, rather, it is a natural approach common in PDES to maintain causality. Agents in our simulations must behave consistently with the interaction protocol specification. If agent a_1 in the example above executed e_{19} immediately upon reception, a rollback would be unavoidable when e_{12} is received. Since the agent has the external knowledge that e_{12} will in fact occur, it is straight-forward to avoid this situation. It should also be noted, that the result of the simulation is not affected in any way by this approach.

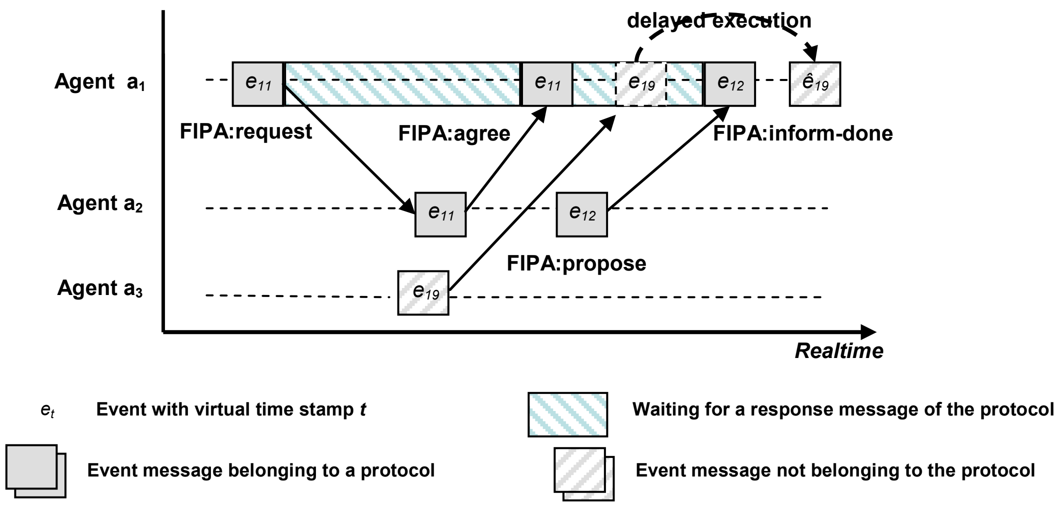

delayed execution
Agent a1
Agent a2
Agent a3
e11
e11
e19
e12
ê19
e11
e12
e19
FIPA:request
FIPA:agree
FIPA:inform-done
FIPA:propose
Realtime
e_t Event with virtual time stamp t
Waiting for a response message of the protocol
Event message belonging to a protocol
Event message not belonging to the protocol

Figure 3: Delayed event execution based on protocol information. Agent a_1 receives a proposal from Agent a_3, while he is waiting for an inform-done message of Agent a_2. Instead of immediately processing the incoming message, the execution is delayed. Thus, event order is preserved and still valid.

Further to what was stated before we also have to consider some special cases, where an agent exceptionally is allowed to execute events, even if it is waiting for a reply message. For example we have to consider *deadlock* situations. Assuming a situation where agent a_1 waits for agent a_2, and at the same time agent a_2 waits for agent a_1. This may be the case since both independently have sent a message to each other. Both agents would then become blocked as they are both waiting for a message event which will never occur. In this case an agent must also process a message from its opponent even if it is not the message content it was waiting for. Another exception are rollback situations. Whenever a message event arrives at a process with a time indicator lower than the local time of the process, we have to quit from wait-state as well. Finally we have to consider *cyclic dependencies*. Although not very likely, there may be situations where an agent receives a request within the same conversation, from a new communication partner different from its original opponent. This may be the case for example in multi-staged-protocols. In this situation, this new message has to be processed by the waiting agent before he can go to wait state again. Since every message is marked with a conversation-id that uniquely identifies to which conversation thread this message belongs, we can easily identify messages accordingly.

The proposed policy certainly cannot fully prevent rollback situations. However, it minimizes the risk for rollbacks caused by messages which causally belong to a common communication. For agent-based simulations this approach certainly performs better than a pure Time Warp solution. Also, this approach is capable of maintaining an acceptable degree of parallelism required for scalable solutions.

The implementation effort of this solution is considerable low. The agent has to be provided with information about the structure of the used protocols at initialization time only. Depending on the protocol length, the policy is applied more frequently. Particular long interaction protocols, like the *fipa-contract-net* are most eligible.

Since this approach is joining optimistic techniques with constrained optimism, it can be classified as a time warp with constraints.

4.2 GVT und Fossil Collection

A good GVT algorithm is critical to the overall performance of an optimistic simulation since memory can be reclaimed with it. Numerous GVT algorithms have been discussed in literature so far. Our implementation differs slightly from known algorithms and relies on a global synchronization, which is achieved by using a butterfly barrier. The processors are numbered 0,1,2 up to N-1. During simulation each processor executes a sequence of log N pair wise synchronization steps with a different processor at each step. To compute a value for the GVT simulated time is divided into a sequence of non-overlapping time slots. Therefore, the simulator broadcasts a special message periodically. Upon the reception of this message each processor will advance the time slot number. When a message is sent by an agent, it is marked with the current time slot of the sender. During each slot the number of sent and received messages is monitored by the processor. If the two numbers are equal, i.e. there are no transient messages for the respective timeslot, GVT can be computed as the lower bound on the local times on all processors and timestamps of all messages that did not reach their destination until now. On the first glance the GVT algorithm looks like a normal LBTS algorithm, but there are some notable differences. Firstly, multiple reductions are normally needed within each time slot. In our approach each processor sends exactly one message per synchronization step. Secondly, processors are normally released from the barrier when they all have completed the last step. Our approach never delays or blocks the processors at the end of a synchronization step. This design helps to ensure an efficient simulation even if many processors are used.

4.3 Repeatability of Simulation

Correctness of results requires that events are executed to preserve *global* timestamp order among all events. Within PDES a model may contain events with the same timestamp. Events that are generated under zero lookahead conditions are often referred to as simultaneous events. The order of execution of such events can have a significant impact on the simulation results. Inconsistent handling of simultaneous events can lead to incorrect simulation behaviour. It may be impossible to reproduce the results of a simulation model across several simulation runs. Because of this, our time service implements a deterministic tie-breaking scheme first described in (Mehl 1992). To ensure repeatability of simulation runs each message contains some extra fields to mark the ancestor message of an event. In this way events are unique whereby ordering of events is done in background without any intervention from the model developer. The algorithm ensures reproducibility for message ordering, even if a message with the same time-stamp but a higher precedence arrives.

5 PERFORMANCE STUDY

In order to test performance of the proposed techniques we used a number of different message-driven simulation models. A first set of experiments was designed to evaluate the efficiency of the proposed time synchronization algorithm We have also tested aspects which are of particular importance for asserting the performance of a time-warp simulation like the rate of inter-processor events, as well as differing workload scenarios. Finally, a set of experiments was designed to demonstrate how the time service scales. Experiments have been performed on two different clusters, in order to evaluate the performance and scalability of the proposed time service. One test environment consisted of a mini-cluster with ten P4 2.8 GHz workstations (512 MB RAM, Windows XP SP2) which were connected by a 100 Mbit switched Ethernet. Some tests were done using an HPC-Cluster with ten 2x1,8 GHz Opteron Processors (2 GByte RAM per Node, Linux Redhat 2.4.19) connected via 2 GBit network interfaces. The simulation executive assigns agents in a round-robin manner to each partition so that each partition contains a more or less equal number of agents and load balancing is achieved. Accordingly, agents are distributed randomly across the processors. On average five independent runs where done for each parameter combination.

5.1 Benchmark Applications

We used four different applications in our performance study. The first is an implementation of the well known PHOLD PDES benchmark. It is a synthetic benchmark, where *n* simulation processes are evenly mapped to available processors. A fixed population of events is generated with random destinations. When an agent receives an event, it schedules a new event into the future to another random destination normally with an exponentially distributed time increment. With probability p-1, the destination is on the same processor as the source. By default the destination LP is chosen with 90% locality (i.e., 10% of events cross processor boundaries). We use a fixed increment with mean 1.0. This is comparable to a lookahead of value 1.0. To determine event destinations a uniform random number generator is used. The PHOLD benchmark is a fine-grained application, i.e. very little computation is performed when processing an event. It therefore represents a worst-case scenario that can reveal runtime overheads of the simulation engine.

A second application covers the simulation of an electronic market with sellers und buyers handling goods. Market participants are represented with agents. By default agents in JADE do not refer to some particular agent architecture. For the given model each agent implements reasoning capabilities using the *Jason* interpreter, an extended version of the Agent-Speak language (Bordini 2007). Since it is written in Java too, Jason can easily be plugged into the body of our JADE agents. To deal with the interaction between agents, communication messages are presented using FIPA-ACL messages. The agents negotiate about randomly created orders using an English auction protocol.

A third example of an agent based model was taken from the simulation of a distributed sensor network. Such sensing systems are composed of sensors and processors spread throughout some area. Sensor networks are configured dynamically. Sensor nodes normally have limited processing capabilities. Each node continuously attempts to find other sensor nodes that can further process the signal features they extract from the raw sensed data. The concrete topology of the net depends on the nodes available, location information and current communication bandwidth.

The last application in our test is an instance of the contract-net-protocol. It is used for distribution of tasks for nodes in a distributed problem solver. During the negotiation process, a conversation between agents with tasks to be executed and agents that may be able to execute those tasks takes place. The participating agents receive a call for proposals from an initiator agent who is asking to execute a concrete task. Contractor agents are able to generate *n* responses.

5.2 Experimental Results

5.2.1 Constrained Optimism

The first set of experiments was performed to evaluate the proposed time synchronization algorithm and was based on the electronic market model. For this test the number of simulated LPs is fixed while the number of processors varies. We track the absolute speedup compared to a sequential simulator as well as the total number of rollbacks. The observed runtime speedup is shown in Figure 4. While both optimistic modes scaled fairly well, our time management policy clearly outperforms the pure time warp solution.

We also compared the results with a sequential discrete event simulator that uses the same simulation engine but with a centralized event list. The overhead of parallelization can be derived from the comparison of the execution time of the optimistic simulation mode with that of the sequential execution when both are executed on one processor. In this setup, the optimistic simulation mode was at least 15% slower than the sequential execution.

Obviously, only when using multiple processors a remarkable speedup was gained compared to the sequential synchronization approach. The results show that our constrained time warp algorithms clearly outperforms the pure time warp implementation. The main reason for these good performance results grounds on the small number of rollbacks within our constrained time-warp algorithm. Within pure Time Warp agents tend to be too far ahead of each other in simulation time. They cause rollbacks more frequently. By deploying delayed message execution we were able to minimize the probability of time consuming rollbacks.

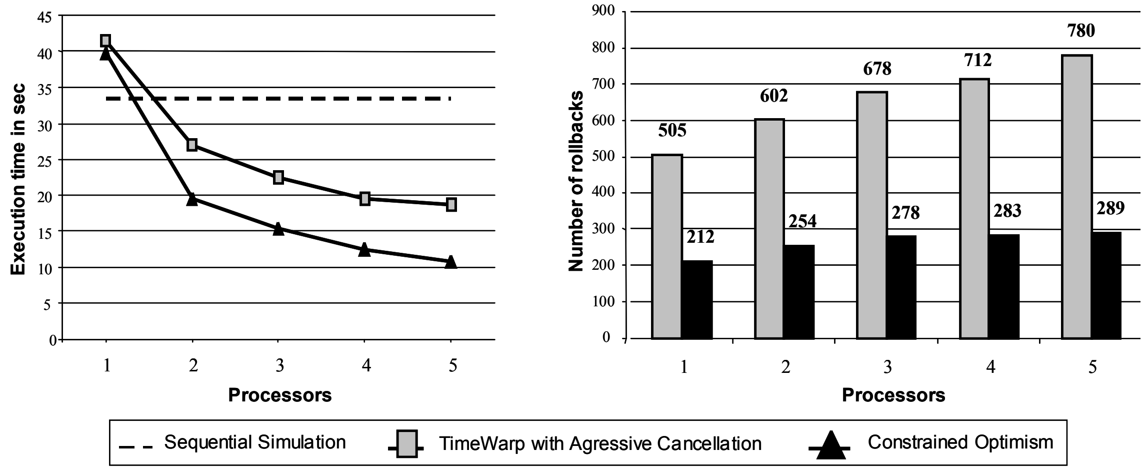


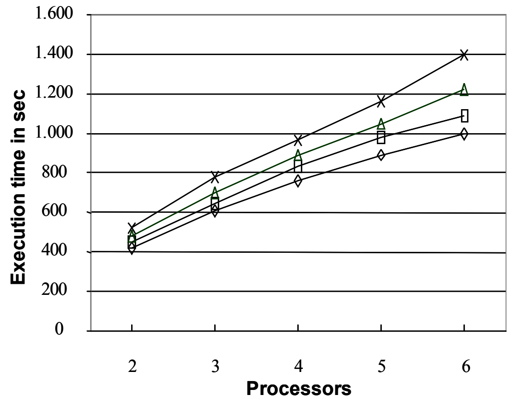
Figure 4: Speedup/Rollback values for electronic market model (100 agents)

5.2.2 With fixed percentage of remote events

The next experiment was designed to demonstrate how the simulator scales with different percentages of remote events. Within the test we used an instance of the contract net model. The problem size is increased linearly to the number of processors. As we can see in Figure 5 the performance drops as the number remote events increases. This is due the higher communication latency, when sending messages along the wire compared to local communication on the same host. A second point are the number of rollbacks. The absolute value also increases because there are more straggler events. Accordingly more events have to be retracted, when increasing the percentage of remote events. However, the simulation executive maintains almost linear increases in throughput time, and therefore scales well.

5.2.3 With fixed computational effort per event

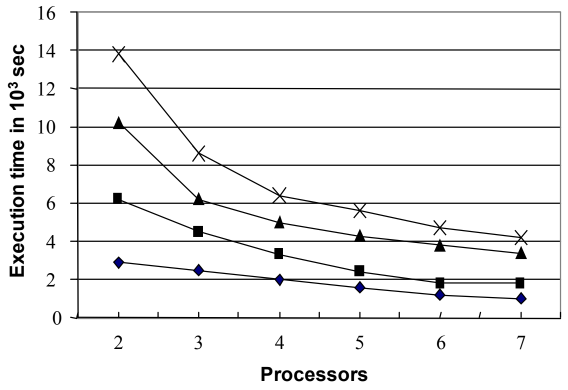
In this experiment we added a new parameter to the PHOLD model. To provide a simple way of exploring performance of agent based models over a wide spectrum of event granularity we use integer matrix multiplication. Using a workload parameter to set the size of each matrix, we can vary the effective grain size of events for experimentation. In the experiment, the matrix size varied from 20x20, to 30x30, 60x60, and 90x90. Figure 6 shows the obtained throughput times as well as the rollbacks. First of all it is worth to note that although workload and therefore execution times increase the total number of rollbacks drops. For example if we compare the workload 20x20 with 90x90 on 6 processors the number of rollbacks drops at a ratio of 3 to 1. Also remarkable, for an 90x90 integer matrix the execution time on 4 nodes is nearly half of the execution time required with 2 nodes. This is a really good relative speedup. We therefore conclude that especially deliberative agent models with complex reasoning capabilities and high computational effort per event can profit from distributed execution.



% of remote Events	Processors/Agents/Events				
	2 /20/ 500.000	3 /30/ 750.000	4 /40/ 1.000.000	5 /50/ 1.250.000	6 /60/ 1.500.000
0%	98	101	109	113	126
15%	131	126	143	156	168
25%	152	178	193	201	212
35%	183	201	218	238	251

Total number of rollbacks

Figure 5: Execution time/Rollbacks for Contract Net model with Constrained Time Warp



Workload	Processors					
	2	3	4	5	6	7
20x20 ◆	238	249	268	289	312	352
30x30 ■	206	208	234	246	267	293
60x60 ▲	166	178	198	203	214	221
90x90 ×	108	118	126	119	134	144

Total number of rollbacks

Figure 6: Execution time/Rollbacks with varying workload per event (PHOLD with 100 agents and 10.000 events)

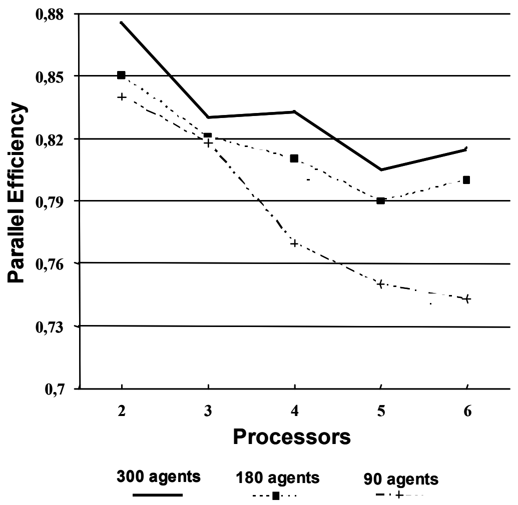
5.2.4 Parallel Efficiency of Agent Models

An often used scalability metric for parallel algorithms is efficiency. Although it is meant for data driven algorithms it can also be used to evaluate the efficiency of PDES problems. Parallel efficiency is defined as the speed-up obtained with *p* processors divided by the number of processors. In other words we can think of efficiency as how well we are utilizing *p* processors when computing an algorithm in parallel. An efficiency of 100 percent means that all of the processors are being fully used all the time.

Figure 7a shows the efficiency reached with a version of the decentralized sensor model with varying model size. In this experiment the number of agents and therefore the size of the model is fixed, while number of processors is increased. This is sometimes referred to as *strong scalability*. Again the simulation executive scales fairly well. Only marginal drops of efficiency could be found. One reason for this result is the lower event scheduling overhead when the number of logical processes per processor decreases with increasing number or processors.

In the second experiment we have measured the performance of our simulator using isoefficiency. This metric was first introduced by Grama et. al (2003). The isoefficiency function for a given efficiency indicates, for a varying number of processors, how much total work is necessary to achieve this efficiency. In other words we investigate, how fast *s* (the size of our simulation model) has to grow with an increasing number of processors p so that efficiency remains constant. If the workload *w(s)* grows as fast as $f_e(p)$ constant efficiency can be maintained.

By experiment, we tried to determine the isoefficiency function for an agent based model (Figure 7b). Again we were using an instance of the PHOLD model. Our test comprises 15% remote events. Workload was simulated by computing a 30x30 integer matrix when processing an event. We applied our prototype to simulate the agent model on different scales. The first set of experiments was done with a fixed ratio of 1:1 between the number of agents and the number of processors. Accordingly we started with 100 agents on 1 processor. Next time we doubled the number of agents and the processors, again with a fixed ratio. The test was repeated for agent-to-processor growth functions of 0.5:1 and 1.25:1. Figure 7b shows the parallel execution efficiency for the three ratios. For a ratio of 0.5:1 the efficiency quickly drops. For a ratio of 1.25:1 where model size grows a bit faster than the number of processors almost the same efficiency value was reached. In other words, we were able to demonstrate a linear isoefficiency of our proposed constrained time warp algorithm for agent based simulation.



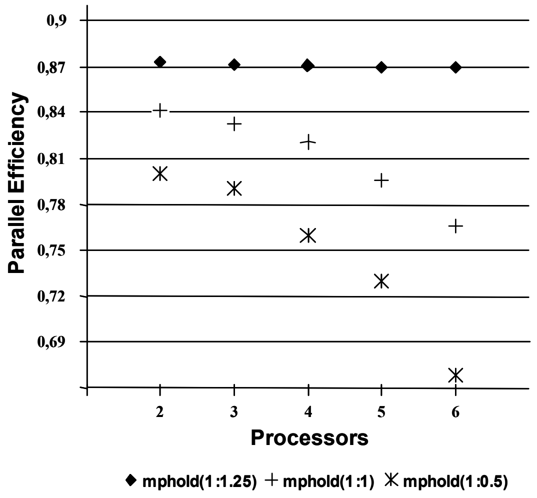


Figure 7: (a) Parallel Efficiency for the Sensor Net model with varying size (b) Isoefficiency for the PHOLD model

6 SUMMARY AND CONCLUSIONS

While agent-based simulation as a methodology has received much attention from the research community in the past, scalability and thus industrial applicability of this technology has been somewhat neglected. In this paper we have discussed prerequisites and measures for scalability within agent-based simulations. In specific we have argued that only the efficient distributed simulation of agent-based models can provide the architectural basis for a scalable solution. We have further on argued that only the deployment of an optimistic synchronization protocol can yield the required performance. Further on, we have demonstrated a constrained optimistic synchronization protocol, which eliminates a significant amount of the drawbacks and risks that exist within the original optimistic time warp protocol. This is achieved by taking advantage of specific characteristics that are inherent in many agent based interaction protocols making it therefore an almost ideal solution for scalable distributed simulation of agent based models. Empirical performance results for a variety of tests support this statement. Future work would need to validate these results with a real industrial application.

REFERENCES

Bordini, R. H., J. F. Hübner. 2007. Jason - A Java-based interpreter for an extended version of AgentSpeak. `<http://jason.sourceforge.net/Jason.pdf>` [accessed June 23, 2009].

Chaturvedi, A. R., J. Chi, S. R. Mehta, and D. R. Dolk. 2004. SAMAS: Scalable Architecture for Multi-resolution Agent-Based Simulation. In: *International Conference on Computational Science*, ed. M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, 779-788. Springer LNCS series, Vol. 3038, 2004.

Davidsson, P. 2000. Multi Agent Based Simulation: Beyond social simulation. In: *Multi Agent Based Simulation*, ed. S. Moss and P. Davidsson, 97-107. Springer LNCS series, Vol. 1979, 2000.

Drougoul, A., D. Vanbergue, T. Meurisse. 2002. Multi Agent Based Simulation: Where are the agents? In: *Multi Agent Based Simulation II*, ed. J. S. Sichman, F. Bousquet, and P. Davidsson, 2-15. Springer LNCS series, 2002.

FIPA. 2002. FIPA ACL Message Structure Specification. FIPA Spec 00061. Available via <http://www.fipa.org/specs/fipa00061/SC00061G.pdf> [accessed April 1, 2009].

FIPA. 2002a. FIPA Interaction Protocols. FIPA Specs 0026-0036. Available via <http://www.fipa.org/repository/standardspecs.html> [accessed April 1, 2009].

Fujimoto, R. 2000. *Parallel and Distributed Simulation Systems*. Wiley Interscience, 2000.

Gasser, L., K. Kakugawa, B. Chee, M. Esteva. 2005. Smooth Scaling Ahead: Progressive MAS Simulation from Single PCs to Grids. In: *Multi-Agent and Multi-Agent-Based Simulation*, 1-10. Springer LNCS series, Vol. 3415, 2005.

JADE 2009. Homepage ot the JADE project: <http://jade.tilab.com> [accessed April 1, 2009].

Grama, A., G. Karypis, V. Kumar, A. Gupta. 2003. *Introduction to Parallel Computing (2nd edition)*. Pearson Education.

Law, D. R. 1998. Scalable Means More Than More: A Unifying Definition Of Simulation Scalability. In: *Proceedings of the 1998 Winter Simulation Conference*, ed. D. J. Medeiros, E. F. Watson, J. S. Carson, and M.S. Manivannan, 781-788. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Lees, M., B. Logan, T. Oguara, G. Theodoropoulos. 2003. Simulating Agent-Based Systems with HLA: The Case of SIM_AGENT - Part II (03E-SIW-076). In: Proceedings of the 2003 European Simulation Interoperability Workshop. Stockholm, Sweden, June 16-19, 2003.

Logan, B. 2005. The Simulation of Agent Systems. In: *AgentLink News. European Coordination Action for Agent-based Computing*, 17: 5–8.

Mehl, H. 1992. A Deterministic Tie-Breaking Scheme for Sequential and Distributed Simulation. In: *Proceedings of 6th the Workshop on Parallel and Distributed Simulation*, ed. M. Abrams and P. F. Reynolds, 199-200.

Mustafee, N., S. J. E. Taylor, K. Katsaliaki, S. Brailsford, S. 2006. Distributed Simulation with COTS Simulation Packages: A Case Study in Health Care Supply Chain Simulation. In: *Proceedings of the 2006 Winter Simulation Conference*, ed. L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 1136-1142. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Nicol, D. M. 1998. Scalability, Locality, Partitioning and Synchronization in PDES. In: Proceedings of the 12th Workshop on Parallel and Distributed Simulation, 5-11: IEEE Computer Society.

Oechslein, C. 2004. Vorgehensmodell mit integrierter Spezifikations- und Implementierungssprache für Multiagentensimulationen. Shaker Verlag, 2004.

Popov, K., V. Vlassov, M. Rafea, F. Holmgren, P. Brand, S. Haridi. 2003. Parallel Agent-Based Simulation on a Cluster of Workstations. In: Proceedings of the Euro-Par 2003 Conference, ed. H. Kosch, L. Böszörményi, H. Hellwagner, 470-480. Springer LNCS series, Vol. 2790, 2003.

Riley, P. 2003. MPADES: Middleware for Parallel Agent Discrete Event Simulation. In: *RoboCup 2002: Robot Soccer World Cup VI*, ed. G. A. Kaminka, P.U. Lima, and R. Rojas, 162-178. Springer LNCS series, Vol. 2752, 2003.

Russel, S., P. Norvig. 2003. *Artificial Intelligence: a modern approach*. Prentice-Hall, New Jersey, 2003

Theodoropoulos, G. K., B. S. Logan. 1999. Distributed Simulation of Agent-based Systems. In: *Proceedings of 3rd Euro-conference on Parallel and Distributed Computing for Computational Mechanics*, ed. B. H. V. Topping, 147-153. CIVIL-COMP-PRESS, Edinburgh.

Uhrmacher, A. M., K. Gugler. 2000. Distributed, parallel simulation of multiple, deliberative agents. In: *Proceedings of the 14th Workshop on Parallel and distributed simulation*, 101-108. Washington, DC, USA : IEEE Computer Society, 2000.

Vitaglione, G., F. Quarta, E. Cortese. 2002. Scalability and Performance of JADE Message Transport System. In: *Proceedings of the AAMAS Workshop on AgentCities*, Bologna, 2002.

Wooldridge, M., N. R. Jennings. 1995. Agent Theories, Architectures, and Languages: a Survey. In: *Intelligent Agents*, ed. M. Wooldridge and N. R. Jennings, 1-22. Springer, 1995.

AUTHOR BIOGRAPHIES

DIRK PAWLASZCZYK is a system engineer at the Dresden Elbe Flugzeugwerke, a subsidiary of EADS. He holds a diploma degree in business and computer sciences from the Ilmenau University of Technology, Germany, where he also worked as a senior researcher from 2002 to 2007. His research interests include agent-based systems as well as distributed simulation. His email is <pawlaszczyk@hotmail.com> .

STEFFEN STRASSBURGER is a professor at the Ilmenau University of Technology. Previously he was head of the "Virtual Development" department at the Fraunhofer Institute in Magdeburg, Germany and a researcher at DaimlerChrysler Research in Ulm, Germany. He holds a doctoral and a Diploma degree in Computer Science from the University of Magdeburg, Germany. He is member of the editorial board of the *Journal of Simulation*. His research interests include distributed simulation and general interoperability topics. His email is <steffen.strassburger@tu-ilmenau.de>.