A Generic Time Management Service for Distributed Multi-Agent Systems

Lars Braubach, Alexander Pokahr, Winfried Lamersdorf

Distributed Systems and Information Systems

University of Hamburg, 22527 Hamburg, Germany

{braubach, pokahr, lamersd}@informatik.uni-hamburg.de

Karl-Heinz Krempels

Communication and Distributed Systems

RWTH Aachen, 52056 Aachen, Germany

krempels@informatik.rwth-aachen.de

Peer-Oliver Woelk

Institute of Production Engineering and Machine Tools

University of Hannover, 30159 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 behavior 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 time management 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 (Lees et al. 2004) .

The main problem addressed in this paper is the coupling and synchronization of independently developed agent systems for testing and simulation purposes. It is argued that existing approaches and tools for time management cannot be easily adapted to the special requirements imposed by the Agent.Hospital and Agent.Enterprise settings. As a solution, a standards-compliant middleware *time service* component is proposed, which allows controlling the execution of process-flows in the distributed MAS. The design and implementation of the time service is based on techniques from the field of simulation.

In the next section, the foundations of simulation in general and multi-agent based simulation in particular 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 requirements of multi-agent systems is described. An example of how the time service is applied in a real-world scenario is given in section 5. In section 6, some related work is presented and finally, in section 7, some concluding remarks are given.

2 Background

A central concept of simulation is time, which can have different meanings according to the context it is used in. To avoid confusion the notions of wallclock and simulation time are introduced. The so called *wallclock time* refers to the perceived execution time while the system is running, which could e.g. be measured by a physical clock. In contrast the *simulation* (also called *virtual* or *logical*) *time* describes the software representation of time that reproduces the relevant timing aspects of the problem domain (Fujimoto 1998) .

An important aspect of simulation environments is *time management*, which enforces the temporal charac-

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

teristics of the problem domain to be correctly reproduced in the simulation. This is necessary because e.g. the delays in the real world depend on quantities such as the speed of light whereas in the simulation world other characteristics independent of the problem domain, such as network speed have to be taken into account. Omitting time management, the order of messages cannot be ensured and simulation runs with the same initial settings could produce entirely different results. Time management can be accomplished by a number of different time management mechanisms which guarantee the timely correct delivery of events and therefore lead to consistent repeatable simulation runs (Fujimoto 1998) .

2.1 Multi-Agent Based Simulation

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) . 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 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 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., Theodoropoulos and Logan say “[...] conventional distributed simulation techniques cannot easily be applied to the problem of modeling 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 modeled in terms of mentalistic attributes whereas objects are purely reactive, use method invocation and are modeled 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 German priority research programme SPP 1083 is focused on research in the field of agent technology for business applications. Two domains are covered by the participating projects: Manufacturing logistics and hospital logistics. Initially the projects have developed stand-alone prototype multi-agent systems focusing 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 analyzed and modeled. 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 algo-

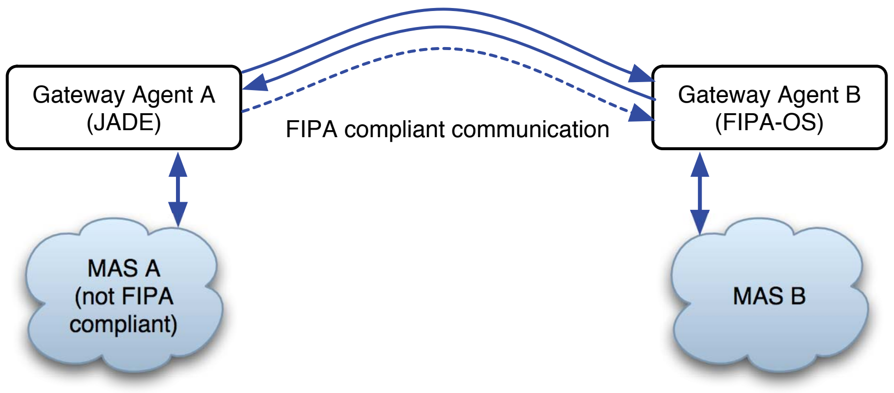


Figure 1: Coupling of MAS using gateway agents, adapted from (Krempels et al. 2003)

rithms. 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 analyzed, formalized and modeled 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 (see Fig. 1). 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 utilize the Agentcities[2] infrastructure to publish and locate the services provided by the different subprojects. Current developments are dealing with enhancements in synchronization 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.

4 Realization of a Middleware Time Service Component

In this section the requirements, design and implementation of the synchronization mechanism for the distributed architecture of Agent.Hospital and Agent.Enterprise are presented.

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

4.1 Requirements

From the Agent.Enterprise and Agent.Hospital scenarios and the agent paradigm itself, a number of general requirements of MABS regarding large-scale agent systems can be derived.

Standardized Interactions The participants of the simulation are not necessarily agents developed to run on the same agent platform. Different subprojects will have to 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.

Efficient Execution The subsystems deal with different cut outs of the superordinated system resulting in events irregularly distributed over time. E.g., in the Agent.Hospital scenario some systems optimize intra-day schedules, while others deal with weekly work plans. The simulation should be able to handle this divergency effectively.

Easy Integration It should be possible to adapt existing systems to the simulation with little effort. In the Agent.X scenarios a lot of implementation work already had been done, before the initiatives decided to couple the MAS of the independent subprojects.

Robustness The time management mechanism should be as robust as possible with respect to a dynamic environment. Therefore, communication failures and the breakdown of single agents should not affect the system as a whole.

Scalability The number of time clients that need to be synchronized may vary and the time management mechanism should not degrade inadequately for a large number of participants.

Flexibility Participating 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.

Agent Autonomy (a) The system must not restrict the agents to only a single task. Inside each agent there may be a number of parallel executing tasks (e.g. concurrent negotiations with different agents). Each task may independently require synchronization with other agents. Therefore, an agent may require a separate time point for each of its tasks. The mechanism has to provide ways for an agent to announce its interest in more than a single event, and take care of routing wake-up calls to the corresponding agent tasks. (b) Agents may autonomously want to revise former made decisions due the newly received information or due to their own cognition processes. With respect to the time management this means that a time client must have the option to change its next wakeup time.

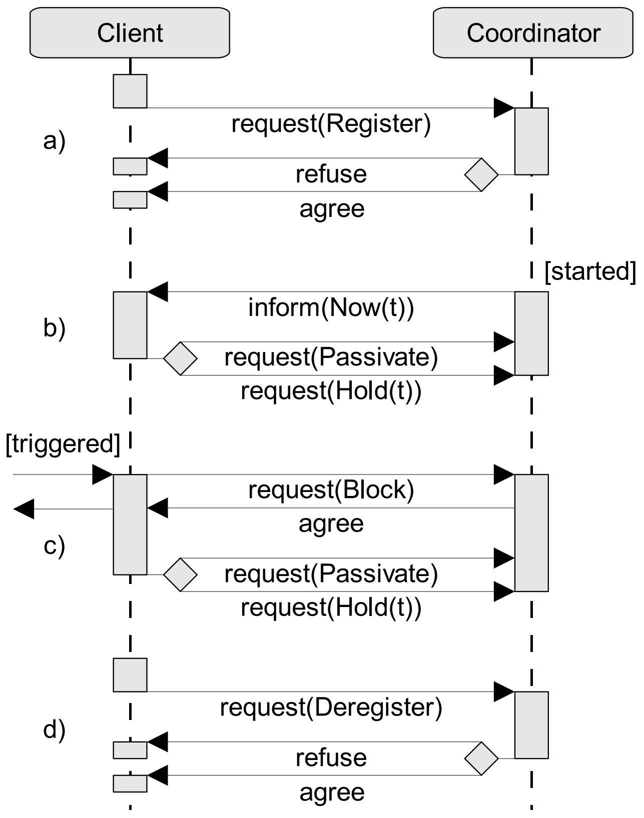


Figure 2: Time Service Protocol

4.2 Design

Some of the above mentioned requirements are 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 efficient, it "considerably simplifies the implementation of correct simulations by avoiding problems of deadlock and possibly overwhelming message traffic" (Ferscha and Chiola 1994) . To respect the autonomy of the individual agents, the traditional coordinator design has to be extended to accept messages from any agent or process at any time, because the continuation of the simulation may not only depend on the agent that received the last wakeup call.

An AUML diagram of the time service interaction protocol is depicted in Fig. 2. 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. The conversation id field provided by FIPA-ACL can be used at the client side to identify the receiving agent task for a coordinator message, when an agent has more than one task that requires synchronization.

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 current activity and subsequently sends back a message when it is finished. To avoid causality errors, the time-advancement is blocked while the activity is executed. 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. For robustness reasons the simulation continues with the next event, if the participant does not answer in a pre-defined timeout period.

While executing a timed activity, only the sending of new wake-up calls is blocked. Waiting processes may still react to messages received from other agents or the waiting agent may change its mind due to other internal deliberation processes (*c*). New information that affects the waiting process may lead to new activities in this process and a new activation time. Therefore, the triggered process can decide to activate itself with respect to 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. The coordinator has to wait until all active processes 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 initially activated process when the coordinator may advance the time. A subsequently triggered process would have to notify the calling process when it has finished its activities. This would have to be done in 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 interaction 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 then 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.

```
<goals>

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

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

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

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

</goals>
```

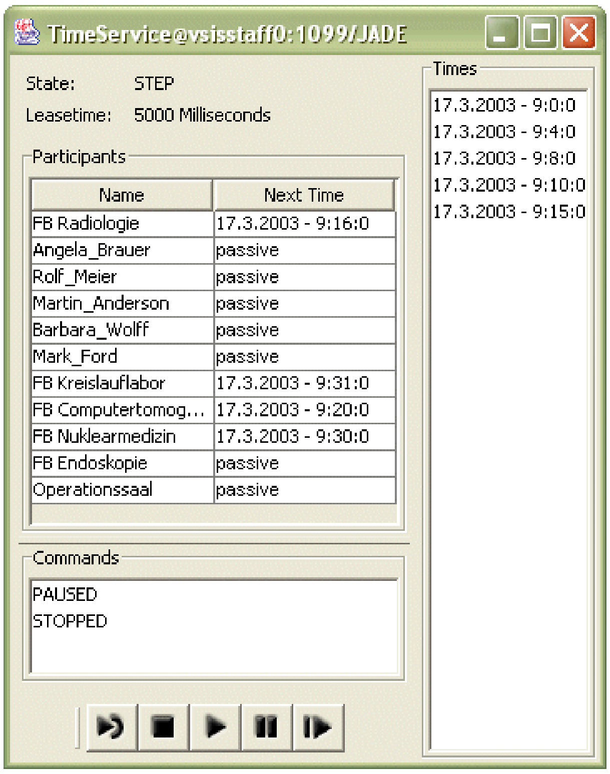
Figure 3: Client Goals (digest)

4.3 Implementation

For the communication between the time service and the participating 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 modeled domain concepts describe things like dates and points in time. The ontology was defined in the ontology modeling tool Protégé (Grosso et al. 1999) .

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. 3) that can be used to easily initiate actions with the time service. These actions correspond 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) . To simplify the usage of the time service a graphical user interface has been developed (see Fig. 4). This interface allows the service to be monitored and controlled manually. The list of participants with current activation points is displayed at the left hand side. Below, the executed service commands are shown. The icons allow issuing commands to the time service (from left to right: slow, stop, start, pause, step). At the right hand side is a visual representation of the global time history (with the



State: STEP
Leasetime: 5000 Milliseconds

Participants

Name	Next Time
FB Radiologie	17.3.2003 - 9:16:0
Angela_Brauer	passive
Rolf_Meier	passive
Martin_Anderson	passive
Barbara_Wolff	passive
Mark_Ford	passive
FB Kreislauflabor	17.3.2003 - 9:31:0
FB Computertomog...	17.3.2003 - 9:20:0
FB Nuklearmedizin	17.3.2003 - 9:30:0
FB Endoskopie	passive
Operationssaal	passive

Commands

```
PAUSED
STOPPED
```

Times

```
17.3.2003 - 9:0:0
17.3.2003 - 9:4:0
17.3.2003 - 9:8:0
17.3.2003 - 9:10:0
17.3.2003 - 9:15:0
```

Figure 4: Time Service GUI

most recent time point at the bottom). The user interface proved very useful for testing and debugging purposes.

5 Example

To facilitate the coordination of the over-all process flows of the Agent.Enterprise and Agent.Hospital scenarios, the time service based synchronization has to be integrated into the interaction between gateway agents, as well as into the different subprojects themselves. The usage of the time service will be explained exemplarily in the context of the MedPAge (Paulussen et al. 2003; Paulussen et al. 2004) project. It is a subproject of the Agent.Hospital scenario and addresses primarily the creation of treatment and examination schedules for patients. These schedules are based on clinical pathways (Dept of Veteran's Affairs, Australia 2000) and are derived through autonomous negotiations between patient and resource agents. The project's objective is to evaluate different coordination mechanisms like e.g., MedPaCo (Paulussen et al. 2003) and compare them with one another and especially with the scheduling technique that is currently applied in most hospitals and subsequently referred to as status-quo. For evaluation purposes, in the Agent.Hospital scenario a hospital model was realized using the SeSAm simulation toolkit (Klügl and Puppe 1998) . This model is used as an input source to generate the external events of the simulation such as admissions of new patients and requests for treatments.

The status-quo technique will be explained in detail here. Patients who need examinations or treatments request appointments from the corresponding functional hospital units. The units decide in their own responsibility

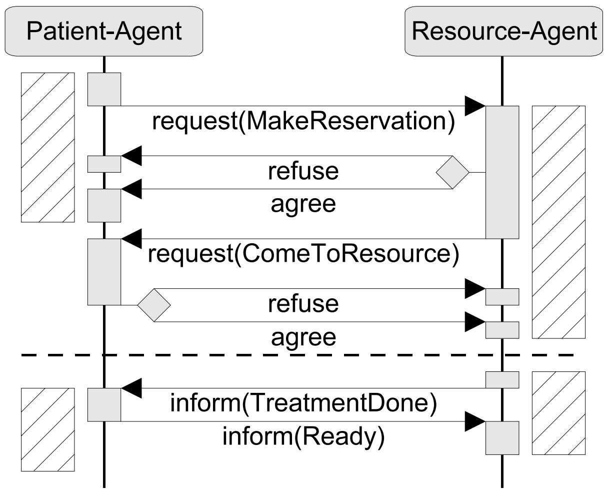


Figure 5: Status-quo protocol

what patient will get the next free slot and will be served. After the decision is made the corresponding patient will be called to the functional unit. It may occur that the patient is unable to keep the appointment due to e.g., another ongoing treatment. In this case the functional unit keeps the patients reservation and tries to schedule another patient first.

Fig. 5 shows the AUML-interaction protocol (Odell et al. 2000) between patient and resource agents. The authors extended the description slightly to support protocol timing aspects. Hatched bars are introduced to indicate that the participant is performing some action and has forbidden the time service to advance the simulation time during the covered interaction, while a horizontal dashed line denotes that the simulation time has advanced. The interaction starts with a patients *MakeReservation* request that can be accepted or refused by the resource. Having received the resources answer, when no other reservation can be made the patient passivates itself, thereby unblocking the time service, and awaits a *ComeToResource* request. The resource blocks the time service after having received the *MakeReservation* request and checks its internal state to decide whether it is not busy and can send for the patient. This will be done by sending the *ComeToResource* request whereupon the patient can agree or refuse. In the case the patient refuses, the call will be repeated later. In the other case the resource waits while the treatment goes on, by sending a *Hold(t)* message to the time service. The time service is now neither blocked from the patients nor from the resources side and gets its clock ahead. It wakes up the resource with a time message whereupon the resource notifies the patient about the finished treatment and awaits an acknowledgement before it may unblock the time service. The patient blocks the service before confirming to the resource and has now the possibility to make further reservations at the current simulation time, by interacting with other resources.

This technique is further clarified by an example interaction between two patients called A and B and one resource. It is assumed that these three participants (A, B and resource 1) are registered by the time service, and that the time service has just started the simulation at 9 o'clock (Fig. 6). Patients A and B request a reservation at resource 1, which agrees to both requests and sends a *Hold* message with 10 o'clock to the time service as this is the desired start time of both treatments. The patients have no more activities to perform, and send a *Passivate* message to the service. The time service adds this new information to its participants list as displayed in the component. At 10 o'clock the time service wakes up the resource with a *Now* message (see Fig. 7). The resource sends for patient A and the patient accepts due to no other obligations. Knowing that the treatments duration is estimated by one hour, the resource sends 11 o'clock as next wakeup time to the service. At 11 o'clock (see Fig. 8), the resource informs patient A that the treatment is finished. The patient blocks the time service (cf. Fig. 2*c*) in order to negotiate for some other appointments at the current time, and answers the resource with a mandatory *Ready* message. The resource sends for patient B, that refuses to come to the resource, because of an ongoing treatment at another resource. The refusal includes the time 12:00 to let the resource know when the patient is available again. After having finished its negotiations (not shown here) patient A unblocks the time service with a *Passivate* message.

The example presented above has been applied to a hospital model derived from field studies, which is comprised of six functional units. During simulation runs, resource agents always synchronize their activities. Patient agents only need to synchronize when they are actively negotiating. It has to be noted, that in the example usually only a few concurrent negotiations will happen at each single point in time, because for any event only one agent will receive a wakeup call from the coordinator. Most of the agents will just wait until their next time point is reached. Nevertheless, a first scalability analysis on a using a profiler tool shows that even when creating a number 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. The tests where performed on a single computer system, therefore it still has to be tested how much communication from different nodes the service can handle, before it becomes a bottleneck.

6 Related Work

The related work concerning multi-agent based simulations can coarsely be categorized into two distinct research areas. *Agent-based simulation systems* use agents as components within a simulation and provide the necessary framework for executing in-the-box experiments. On the contrary, *simulation middleware* provides the utilities to control the execution of MAS with respect to temporal constraints from the problem domain. Therefore, simulation middleware employs time management mechanisms for assuring time consistency in MAS.

6.1 Agent-Based Simulation Systems

Simulation systems are generally used to conduct experiments with a world model that reproduces a relevant cutout of the real world. Experiments are meant for either prediction or explanation purposes and are based on the (statistically evaluated) simulation results. In the following some typical representatives of simulation systems will be presented with respect to their primary application area (microscopic vs. macroscopic perspective).

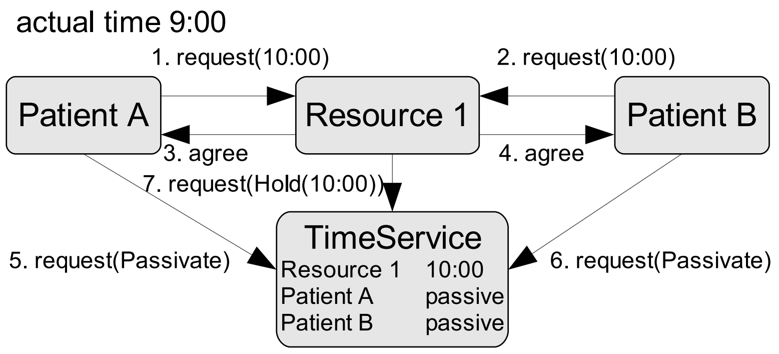


Figure 6: Two patients request a reservation

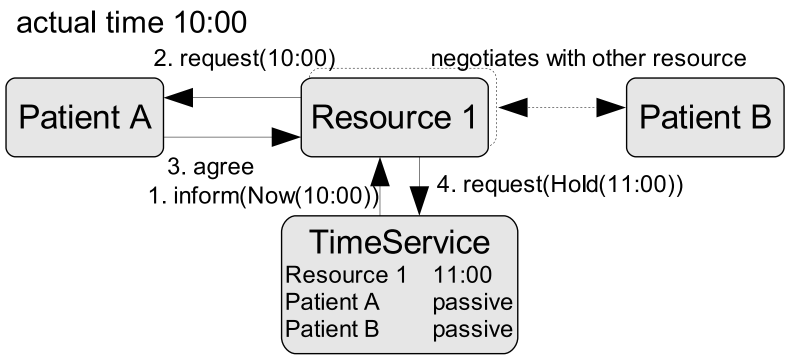


Figure 7: The resource calls patients A

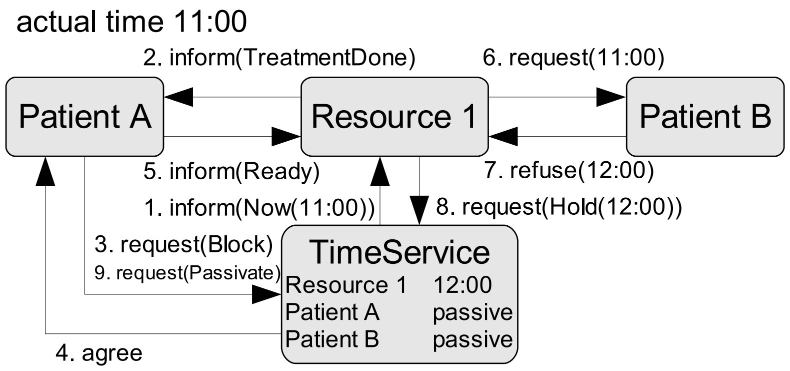


Figure 8: The resource informs patient A and requests patient B

At the microscopic level simulation systems are utilized to evaluate artificial intelligence aspects of the agent internal cognition processes. E.g., James (Uhrmacher and Schattenberg 1998) is a 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 on the premise that the agents planning time is tracked by the system (Schattenberg and Uhrmacher 2001) . Another simulation framework 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 is similar to the more basic MESS system (Anderson 1997) , which tracks the computation of agents at the level of LISP instructions.

At the macroscopic level simulation systems are used to investigate facets of collective and emergent behavior of agent groups. They allow the simulation of complex societies, e.g., for biological, economical or sociological experiments. For example SeSAm (Klügl 2001) is a Java-based simulation system that is intended to be useable by domain experts and was primarily used for biological experiments. It offers a graphical programming language which allows the definition of agent behaviors in a graphical UML-statechart (Object Modeling Group 2001) like fashion. Another well-known system is RePast (Collier 2001) which is originally based on Swarm (Swarm Development Group 2000) and aims to support the modeling of organizations and institutions as recursive social constructions. In contrast to SeSAm, RePast is a framework approach and addresses mainly software engineers as intended users.

Considering the multitude of agent-based simulation systems one can state that these frameworks offer various mechanisms for simulating in an event-based and time-driven manner. So why can't we take one of the above mentioned systems to easily synchronize agent actions in a given multi-agent system? One reason for this is that agent simulation systems are closed systems and are not intended to be used as part of an application. Therefore they do not offer standardized interfaces for using specific functionalities like their time management. Another important point concerns the kind of agents that are represented within a simulation system. These agents do not employ the agent concept at the implementation level, which means that agents in simulation systems have some characteristics making them different from software agents inhabiting agent platforms. Drogoul et al. say in (Drogoul et al. 2002) "... [simulated] agents nowadays constitute a convenient model for representing autonomous entities, but they are not themselves autonomous in the resulting implementation of these models". This reduced degree of autonomy helps in building agent simulation frameworks in that it reduces the complexity of the system behavior and allows a superordinated control of the agents through the environment. It also facilitates the handling of the time management mechanisms. In addition simulated agents are different from platform agents in that restrictions are imposed on the intra-agent architecture allowing only relatively simple and non agent-oriented behavior models for describing basic agents to be used (Drogoul et al. 2002) .

6.2 Simulation Middleware

Simulation middleware can be categorized into proprietary and standards-compliant middleware. A well-established standard for simulation middleware is the High Level Architecture (HLA) standard for interoperability of distributed simulations (IEEE 1516, see e.g. Kuhl et al. 1999), which offers, among other services, interfaces for time management (Fujimoto 1998) . Recently, several approaches have been proposed to use HLA middleware for synchronization of distributed multi-agent based simulations. In (Kratkiewicz et al. 2004) a technical solution is presented that combines the Cougaar agent platform (Helsinger et al. 2004) with HLA federates for military logistics simulations. Another approach presented in (Lees et al. 2004) extends the SIM_AGENT toolkit to distribute simulations using HLA. Both approaches reuse and extend the existing time management facilities as provided by Cougaar societies respective the SIM_AGENT scheduler, and introduce HLA federates only to synchronize instances of those high-level constructs. Therefore, the way how the single agents are connected to the time management is specific to the used system (Cougaar resp. SIM_AGENT).

In contrast, (Turner and Wang 2004) proposes an architecture where agents are attached directly to so called gateway federates, which provide access to the HLA runtime infrastructure (RTI) and uses the time management capabilities to control all the message exchanges between agents. This approach is in principle independent of the underlying system, as long as communication between those systems is possible. Unfortunately, although the architecture is implemented on the FIPA-compliant agent platform JADE (Bellifemine et al. 1999) , the system uses a proprietary communication mechanism, and therefore cannot be used to integrate other FIPA-compliant platforms.

Even when operating in conservative mode, the HLA allows the local virtual time of federates to advance independently of each other. To avoid causality errors all actions have to be managed by the federates and all messages have to include a timestamp such that before a message is delivered the local time of the sender can to be compared to the local time of the receiver. Having a runtime infrastructure responsible for coordination resembles to some extent the idea of performing agent interactions indirectly through coordination artifacts (Viroli et al. 2005) . Coordination and synchronization in both approaches happens at the level of individual (inter-)actions. The time service on the other hand is a lightweight approach, because only time dependent (inter-)actions have to be synchronized. Other actions and communication may happen at any time without the need for synchronization, because the time service assures that in any case there is always a single global time valid for all agents.

In (Holvoet and Weyns 2004a; Holvoet and Weyns 2004b) a proprietary, but very innovative approach concerning time management in MAS is described, which treats the handling of time as a separate application concern. Hence it is possible to detach the simulation specific components and use the system without simulation overhead. The approach is based on two fundamental building blocks. Time models are used to explicitly capture the execution policy derived from the relevant timing constraints in the application domain, whereby specific time management mechanisms are employed to enforce the given time models. A MAS execution control component serves as new infrastructure layer between the agent application and the MAS platform. It combines the time models and time management mechanisms to control the execution of a MAS.

The time models are semantic duration models, which reflect the timing characteristics of the MAS's problem domain by ascribing durations to the relevant internal and environmental actions. Each agent possesses a local clock, which is advanced according to the durations of actions the agent is performing. Whenever an external action is executed it is validated against the other local clocks and eventually delayed until all agents have proceeded to this point in time. In the meantime the agent is frozen and therefore not allowed to engage in further actions. This is a serious problem with respect to environments in which the negotiation of agents can occur at any point in time, as required by the example presented in section 5. Additionally, the approach currently does not support more than one local clock per agent which hinders an agent to engage in two or more actions independent of each other.

7 Concluding Remarks

The objective of this paper is to describe a lightweight middleware service component for time management 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.

Looking back at the requirements presented in section 4, one can see how and to what degree they are addressed by design and implementation choices:

Standardized Interactions The communication is FIPA-compliant and therefore the clients and the service can be easily decoupled and implemented on different platforms.

Efficient Execution The use of discrete event-based simulation techniques allows the events to be arbitrarily distributed over time without loss of performance. The choice of a conservative approach ensures that only correctly ordered wake-up calls are generated and additionally respects the decentralized multi-agent system structure which normally does not allow actions to be rolled back as would be needed in an optimistic approach.

Easy Integration The basis for an easy integration process was provided on the design level by choosing a conservative centralized simulation approach and on the implementation level by providing a pluggable agent module to handle timing aspects. The various administrative commands like slow and step mode that are available through a user interface support debugging as well.

Robustness The breakdown of single client agents is tolerated by the approach through the rigid use of a timeout mechanism. Nevertheless, the centralized coordinator remains a bottleneck and single point of failure in the application.

Scalability The approach does not cause much message overhead due to the fact that only timely relevant actions need to be synchronized with respect to the time service. Preliminary tests have shown that the mechanism is capable to handle a medium size application scenario (~50 agents) with acceptable synchronization overhead. Anyhow, the centralized approach is conceptually a potential scalability drawback.

Flexibility The chosen approach offers the possibility to add and remove time clients dynamically by utilizing the "register" and "deregister" commands.

Agent Autonomy (a) For each agent an arbitrary number of time clients (tasks) can participate at the time management allowing for the participation in different (potentially parallel) negotiations. (b) The time clients have the possibility to reconsider their already submitted next points in time by utilizing the "block" command and subsequently updating their choice accordingly.

Recapitulating, it can be stated that the time service approach addresses most of the aforementioned requirements (at least partly) and has the main advantage of being a simple, interoperable and agent-friendly mechanism for time management. The main drawbacks are its centralized structure and the fact that domain protocols have to include timing aspects when the corresponding domain actions need to be synchronized.

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, S. D. (1997). Simulation of Multiple Time-Pressured Agents. In S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson (Eds.), *Proceedings of the 29th Winter Simulation Conference*, pp. 397–404. ACM Press.

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

Collier, N. (2001). RePast: An Extensible Framework for Agent Simulation. Working Paper, Social Science Research Computing, University of Chicago.

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

Davidsson, P. (2000). Multi Agent Based Simulation: Beyond social simulation. In *Multi Agent Based Simulation*. Springer Verlag LNCS series, Vol. 1979.

Dept of Veteran's Affairs, Australia (2000). *Clinical Pathway Manual for Geriatric Community Nursing*. Dept of Veteran's Affairs, Australia. `http://www.dva.gov.au/health/provider/provider.htm`.

Drogoul, A., D. Vanbergue, and T. Meurisse (2002). Multi-Agent Based Simulation: Where are the Agents? In *Proceedings of MABS'02 (Multi-Agent Based Simulation)*. Springer-Verlag.

Ferscha, A. and G. Chiola (1994). Accelerating the Evaluation of Parallel Program Performance Models Using Distributed Simulation. In *Proceedings of the 7th international conference on Computer performance evaluation : modelling techniques and tools*, pp. 231–252. Springer-Verlag New York, Inc.

Foundation for Intelligent Physical Agents (2002). FIPA Request Interaction Protocol Specification. Document no. FIPA00026.

Frey, D., T. Stockheim, P.-O. Woelk, and R. Zimmermann (2003). Integrated Multi-agent-based Supply Chain Management. In *12th IEEE International Workshops on Enabling Technologies (WETICE 2003), Workshop on Agent-based Computing for Enterprise Collaboration (ACEC)*.

Fujimoto, R. M. (1998). Time management in the high level architecture. *Simulation Special Issue on High Level Architecture 71*(6), 388–400.

Fujimoto, R. M. (1999). Parallel and Distributed Simulation. In *Proceedings of the Winter Simulation Conference*, pp. 122–131.

Grosso, E., H. Eriksson, R. W. Fergerson, J. H. Gennari, S. W. Tu, and M. A. Musen (1999). Knowledge Modeling at the Millennium (The Design and Evolution of Protege-2000). Technical Report SMI-1999-0801, Stanford Medical Informatics Group (SMI), Stanford University School of Medicine.

Helsinger, A., M. Thome, and T. Wright (2004). Cougaar: A Scalable, Distributed Multi-Agent Architecture. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*. IEEE Computer Society Press.

Holvoet, A. H. T. and D. Weyns (2004a). Time Management Adaptability in Multi-agent Systems. In *Fourth Symposium on Adaptive Agents and Multi-agent Systems at the AISB '04 Convention*. (to appear).

Holvoet, A. H. T. and D. Weyns (2004b). Time Management Support for Simulating Multi-Agent Systems. In *Joint Workshop on Multi-Agent and Multi-Agent-Based Simulation (MAMABS-04) at AAMAS 2004*. (to appear).

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

Kirn, S., C. Heine, R. Herrler, and K.-H. Krempels (2003). Agent.Hospital - agent-based open framework for clinical applications. In G. Kotsis and S. Reddy (Eds.), *Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2003 Post-Proceedings)*, Linz, pp. 36–41. CS Press, Los Alamitos, CA.

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

Klügl, F. and F. Puppe (1998). The Multi-Agent Simulation Environment SeSAm. In H. K. Büning (Ed.), *Proceedings of SiWiS'98: Simulation in Wissensbasierten Systemen*. Technical Report tr-ri-98-194, Universität Paderborn.

Kratkiewicz, G., A. Fedyk, and D. Cerys (2004). Integrating a Distributed Agent-Based Simulation into an HLA Federation. In *Simulation Interoperabilty Workshop SIW '04*. (to appear).

Krempels, K.-H., J. Nimis, L. Braubach, A. Pokahr, R. Herrler, and T. Scholz (2003). Entwicklung intelligenter Multi-Multiagentensysteme - Werkzeugunterstützung, Lösungen und offene Fragen. In K. Dittrich, W. König, A. Oberweis, K. Rannenberg, and W. Wahlster (Eds.), *Informatik 2003 - 33. Jahrestagung der GI*, Volume P-34 of *Lecture Notes in Informatics (LNI)*, pp. 31–46. Gesellschaft für Informatik e.V.: Köllen Druck+Verlag GmbH.

Kuhl, F., R. Weatherly, and J. Dahmann (1999). *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Prentice Hall PTR.

Lees, M., B. Logan, R. Minson, T. Oguara, and G. Theodoropoulos (2004). Distributed Simulation of MAS. In *Joint Workshop on Multi-Agent and Multi-Agent-Based Simulation (MAMABS-04) at AAMAS 2004*. (to appear).

Lees, M., B. Logan, T. Oguara, and G. Theodoropoulos (2004). HLA_AGENT: Distributed Simulation of Agent-Based Systems with HLA. In *Proceedings of the International Conference on Computational Science (ICCS'04)*, Krakow, Poland, pp. 907–915. Springer.

Object Modeling Group (2001). *Unified Modelling Language Specification, version 1.4*. Object Modeling Group.

Odell, J., H. V. D. Parunak, and B. Bauer (2000). Extending UML for Agents. In G. Wagner, Y. Lesperance, and E. Yu (Eds.), *Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, pp. 3–17.

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

Paulussen, T. O., N. R. Jennings, K. S. Decker, and A. Heinzl (2003). Distributed Patient Scheduling in Hospitals. In G. Gottlob and T. Walsh (Eds.), *Proceedings of the Eighteenth International Joint Conference on*

Artificial Intelligence (IJCAI-03). Morgan Kaufmann.

Paulussen, T. O., A. Zöller, A. Heinzl, A. Pokahr, L. Braubach, and W. Lamersdorf (2004). Dynamic Patient Scheduling in Hospitals. In M. Bichler, C. Holtmann, S. Kirn, J. Müller, and C. Weinhardt (Eds.), *Coordination and Agent Technology in Value Networks*. GITO, Berlin.

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

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

Schattenberg, B. and A. Uhrmacher (2001). Planning Agents in James. In A. Uhrmacher, P. Fishwick, and B. Zeigler (Eds.), *Agents in Modeling and Simulation: Exploiting the Metaphor, Special Issue of the Proceedings of the IEEE, Vol.89, No.2*, pp. 158–173.

Swarm Development Group (2000). *A tutorial introduction to Swarm*. Swarm Development Group. `http://www.swarm.org/csss-tutorial/frames.html`.

Theodoropoulos, G. and B. Logan (1999). A Framework for the Distributed Simulation of Agent-Based Systems. In H. Szczerbicka (Ed.), *Modelling and Simulation: a tool for the next millennium, Proceedings of the 13th European Simulation Multiconference (ESM'99)*, pp. 58–65. Society for Computer Simulation International: SCS Publications.

Turner, F. W. S. J. and L. Wang (2004). Agent Communication in Distributed Simulations. In *Joint Workshop on Multi-Agent and Multi-Agent-Based Simulation (MAMABS-04) at AAMAS 2004*. (to appear).

Uhrmacher, A. and B. Schattenberg (1998). Agents in Discrete Event Simulation. In A. Bargiela and E. Kerckhoffs (Eds.), *10th European Simulation Symposium "Simulation in Industry – Simulation Technology: Science and Art" (ESS'98)*, pp. 129–136. The Society for Computer Simulation International: SCS Publications, Ghent.

van Aart, C., R. Pels, G. Caire, and F. Bergenti (2002). Creating and Using Ontologies in Agent Communication. In *Second International Workshop on Ontologies in Agent Systems*.

Viroli, M., A. Ricci, and A. Omicini (2005). A Semantics for the Interaction of Agents with Coordination Artifacts. *Applied Artificial Intelligence Best of "From Agent Theory to Agent Implementation (AT2AI)-4, (this volume)"*.