Proceedings of the 29th Annual German Conference on Artificial Intelligence (KI 2006).
Lecture Notes in Computer Science, Vol. 4314. Springer, Berlin, 2006.

# A Hybrid Time Management Approach to Agent-based Simulation

Dirk Pawlaszczyk[1], Ingo J. Timm[2]

[1] Technische Universität Ilmenau, Institute for Information Systems,
Postfach 10 05 65, 98684 Ilmenau, Germany
Dirk.Pawlaszczyk@tu-ilmenau.de
[2] Center of Computing Technologies (TZI), University of Bremen
Postfach 33 04 40, 28334 Bremen, Germany
i.timm@tzi.de

**Abstract.** In this paper we describe a time management approach to distributed agent-based simulation. We propose a new time management policy by joining optimistic synchronization techniques and domain-specific knowledge based on agent communication protocols. With respect to our experimental results, we assume that our approach helps to prevent too optimistic event execution. Consequently, the probability of time consuming rollbacks is reduced in comparison to a pure time warp based solutions. The approach has been implemented as a synchronization service for the JADE agent platform *SimJade*. The paper concludes by the discussion of our experimental results and future improvements.

## 1 Introduction

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, with uncertain and a dynamically changing environment. For validation of those systems agent based simulation seems to be well-suited [1].

Simulation is the imitation of a system's behaviour and structure in an experimental model to reach findings, which are transferable to reality. In multiagent-based simulation (MABS) real world systems are modelled using multiple agents. The system emerges by interaction of the individual agents as well as their collective behaviour. Agents send messages with respect to some communication protocol and are disposed at some discrete point in time (see Definition 1). In this context, a software agent is defined as, a program that acts autonomously, communicates with other agents, is goal-oriented (pro-active) and uses explicit knowledge [2]. Beyond this, MABS is influenced and grounded on existing simulation techniques such as object oriented simulation and distributed simulation [3]. Agent-based simulation has reached a growing attention from both science and industry in recent years. Agent-based modeling and therefore agent-based simulation seems to be the right tool for domains characterized by discrete decisions and distributed local decision makers

[4],[5]. MABS is an appropriate method if we wish to understand the evolution of a distributed system, minted with non-linear dynamics. For instance, a rising number of contributions are dealing with agent-based simulation models in the context of supply chain management since this domain provides distributed entities with autonomous behaviour [6]. Furthermore agents based simulation can be applied to evaluate decentralized decision policies for example within agile or holonic manufacturing systems [4]. As a third point agent-based simulation can help to analyze the behavior of complex self-organizing and emergent systems [8]. Last but not least MABS can be used to validate multiagent systems within the simulation model, before they are deployed to real environments, as part of the software development process [1],[9].

**Definition 1. (Simulation Model):**
*A Simulation model S is defined as a tuple* $<A^*, M^*, P^*, T^*>$ *where*
$A^* = \{a_0, a_1, \ldots, a_n\}$ *is a set of agents,*
$M^* = \{ m_0, m_1, \ldots, m_n\}$ *is a set of messages,*
$P^* = \{ p_0, p_1, \ldots, p_n\}$ *is a set of agent communication protocols, and*
$T^* = \{t_0, t_1, \ldots, t_n\}$ *is a set of time stamps within simulation time* ($n \in N$).

*Let* $M(p) \subseteq M^*$ *define a set of messages that belongs to some protocol p* ($p \in P^*$).

*Let* $T(a) \subseteq T^*$ *denote a set of time stamps that some agent a* ($a \in A^*$) *processes during a simulation run.*

Obviously, MABS has strong requirements with respect to its inherent computational complexity. Nevertheless, in many domains, when distributed systems or high complex world models are in question, MABS seems to be the most adequate form of distributing the simulation model resp. to run agents on multiple computational nodes in a parallel manner. The distribution of the simulation model, however, 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. Event-based simulation is the 'gold standard' for simulation. In distributed simulation, the distributed simulation processes could compute progress in the simulation in an asynchronous way, i.e., simulation events in the distributed computational node are happening at different time points with respect to the time in simulation as well as the current time in the real world. Events, which were scheduled to happen after each other, could happen in a varying order if the computational nodes would not be synchronized. However, complete synchronization would create an almost sequential simulation behaviour, which is not desirable and would prevent the system to scale with respect to speed-up. In Consequence, synchronization is the key challenge in distributed simulation [10],[11],[12]. In this paper, we propose a hybrid event driven time management approach based on local knowledge to provide efficient distributed simulation. Therefore, we first give a short introduction on time management protocols and why they have relevance in distributed simulation. In section 3 we propose a new synchronization approach to MABS using constrained optimistic behaviour. Furthermore, we introduce *SimJade* as a prototypical implementation of the resulting synchronization service with our approach to time man-

agement policy. Finally, we present experiential results, discuss related work, and conclude with a discussion and a brief outline of future work.

## 2 Synchronization – Technical Background

The main component of simulation is the simulation model. Commonly, it is represented by a set of variables and specific behaviour for their value changes, i.e., induced by some input, each variable is modified with respect to the input and its value over time. Another key component of simulation is the time model. There are different notions of time within simulation [10]. The *physical* time denotes the time of the physical system, whereas the *simulation* time is used to represent the physical time within the simulation model. Beyond this, we have to discern the *wallclock* time, which refers to the processing or lead-time of the simulation program.

Depending on their timely fashion, simulation approaches are distinguished as *continuous* and *discrete* simulation techniques. Within continuous simulation, changes to the system state occur continuously in time, whereas in discrete models states are modified at discrete time points only. Continuous simulation is normally performed by using a system of differential equations.

Discrete simulation can further be slit into *time-stepped* models, where simulation advances in equidistant time steps, and *event-driven* models, where the system state is changed only if an event is fired [10]. In a time-stepped model, simulation time can only advance systematically. In fact, not every state variable is really changed in each time step, but must be synchronized anyway. Actions occurring at the same time step are normally considered to be simultaneous and therefore independent of each other. For many problem domains, such an assumption seems to be rather insufficient. By choosing a high granularity of the time steps such situation may be avoided. However, this again must be paid with a high synchronization overhead. Nevertheless, many agent-based simulation test beds rely on this synchronization approach, since it is easy to implement [5],[9],[13].

Within event-driven simulation, a time stamp is assigned to each event, to indicate the point in simulation time, when the event occurs. Discrete event simulation can easily be realized on a single machine using one global event list that manages the events of all *Logical Processes* (LP), i.e. the agent process. All agents in the simulation model are synchronized by a centralized data structure. Hence, it is ensured that events are processed in the correct order of time stamps (*sequential* simulation approach). But this solution is clearly contradictory to the underlying idea of MABS, i.e., the inherent parallelism is hardly used. The dominant approach in parallel event-driven simulation (PEDS) on the other hand is based on the approach, that each LP maintains its own local simulation time (*Local Virtual Time* (LVT)).

**Definition 2. (local virtual time):** *LVT(a, t) denotes the current local virtual time, i.e., how far an agent has progressed in simulation time, with* $a \in A^*$, $t \in T(a)$.

During simulation, the progress of the agents varies from agent to agent. Additionally, it is assumed, that there is no global event list but each LP contains an individual

event list, locally. As soon as simulation is processed on different processors, there is the necessity of synchronizing event execution to preserve the correct event order with respect to their time stamps. Since every agent manages its event list independently, the correct order of events with respect to their time stamps is not ensured. This is commonly known as the *causality problem* within distributed simulation [10]. *Figure1* gives an example for violation of causality. Corresponding to this figure, we can define a straggler message (see definition 3a).

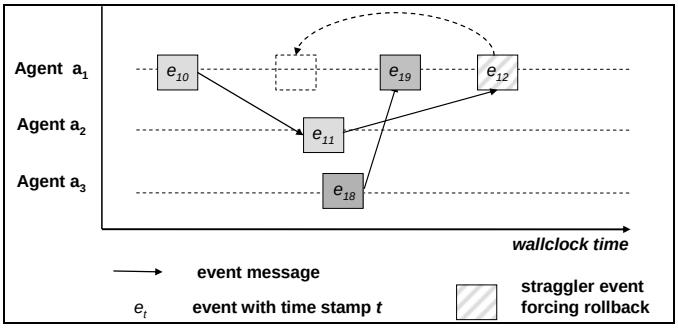


**Fig. 1.** Scenario in distributed simulation causing violation of event ordering (causality error). When event $e_{12}$ arrives, $e_{19}$ has already been processed by agent $a_1$ *(straggler event).*

**Definition 3a. (straggler message):** *If a message m is received by agent a with time stamp $t_2$ which is less than current local time of the agent LVT(a,$t_1$), this "late" message is referred to as straggler message:*

$$\forall a, t_1, t_2, m: (LVT(a, t_1) \wedge received(a, m, t_2) \wedge (t_2 < t_1) \rightarrow isStraggler(m))$$
$$a \in A^*, m \in M^*, t_1, t_2 \in T(a)$$

*The ternary relation received(a, m, t) denotes that agent a has received message m on time stamp t.*

see definition (1),(2)

There are two main approaches in place to ensure correct time stamp order: *conservative* and *optimistic* synchronization. Conservative synchronization algorithms introduce constraints on events and therefore avoid violation and ensure local causality Conservative algorithms prevent the event ordering from being violated. Accordingly, a situation as illustrated in Figure 1 should never occur when using a conservative synchronization scheme. Therefore conservative protocols cannot fully take advantage of parallelism within the application resp. the parallel infrastructure of multiple computational nodes, as in the conservative approach, the guidance is addressed to the worst-case scenario only, i.e. the incoming of a straggler message, which may rarely actually occur in practice. *Optimistic* algorithms like Jefferson's *Time Warp* mechanism on the other hand explicitly allow causality errors, and provide suitable techniques to recover from an incorrect system state. In the example introduced above, agent $a_1$ has to go back in time and rollback its execution state from *LVT 19* to *LVT 12.* Occasionally, an incorrect message that has already been processed by an

agent can result in the generation of additional incorrect messages that have been processed by other agents, which results in still other incorrect messages. Undoing all effects of incorrect computations, the agent has to unsent all previously sent (possibly) incorrect messages using so called *anti-messages*[10]. Each agent that receives such an anti-message has to rollback its state as well. Therefore a rollback within optimistic simulation can be defined as:

**Definition 3b. (rollback):** *Let m be an anti-message resp. a straggler message received in* $t_2$, *the rollback is defined as the behavior initiated by the receiving agent a. Doing so, the agent reactivates its state to a former point in time* $t_3$ *(new LVT) that is less then the current local virtual time* $t_1$, *and less or equal* $t_2$ *(cf.* definition (1),(2),(3a))*:*

$$\forall a, t_1, t_2, t_3, m: (LVT(a, t_1) \wedge (isAntiMessage(m) \vee isStraggler(m)) \wedge received(a, m, t_2) \rightarrow rollback(a, t_3) \wedge newLVT(a, t_3))$$

$$a \in A^*, m \in M^*, t_1, t_2, t_3 \in T(a), \text{with } t_3 \leq t_2 < t_1$$

Conservative as well as optimistic time management protocols have gained remarkable speedups in the recent years. Various approaches have been proposed to control the optimism of the simulation model. Conservative synchronization scales well, if the lookahead, i.e. the ability of an LP to make predictions about its own future, is high. But, in dynamic environments with frequently changing relationships between model entities optimistic synchronization insures a better scalability, since it does not rely on the lookahead of a certain simulation model. Additionally, the model developer has not to be familiar with details of synchronization, as within conservative simulation. Since we are looking for a suitable time management approach to agent-based simulation that provides enough performance for a wide range of models optimistic synchronization seems to be a good choice.

## 3 The Hybrid Approach

In conservative synchronization, algorithms prevent the event ordering from being violated. Within pure Time Warp implementations, no constraints exist on the distance in time an agent process resp. the advance ahead of others into future. Consequently, the probability of incorrect computations increases. As shown above, each straggler message causes one or even more rollbacks. If the time to perform a rollback is high, i.e. many states have to be rolled back, the performance of the simulation decreases significantly. This commonly known performance hazard within optimistic synchronization is caused by too optimistic event execution [10]. A good time management protocol should avoid such situations. At the same time, it has to be ensured, that parallelism is not fully lost within the simulation model, as it is caused by a too restrictive time management policy for example. The question that arises is how to prevent these potential performance hazards? Communication is one of the

key features in agent technology. Messages are sent out from a sender to a receiver resp. a list of receivers. Messages are encoded in an *Agent Communication Language (ACL),* an external language that defines 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). Commonly, communication of agents is based on such protocols. The FIPA specification, as an internationally agreed agent standard, defines an agent communication language as well as a set of interaction protocols, most commonly used [14]. The FIPA *Request Interaction Protocol* for example allows an agent to request another agent to perform some action. The participant needs to decide whether to accept or refuse the request. In any case, the message receiver has to respond with a reply to a request message. Even if the receiver does not have any clue how to deal with a message, the specification prescribes to send a least a *not-understood* message. With this in mind we define policies for time management (see def. 4.) Furthermore we have to consider some special cases, where the agent exceptionally is allowed to do something, even if it is waiting for a reply-message (see def.5a,5b).

**Definition 4. (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 reply $m_2$ for $m_1$. The rule "wait for" is defined as follows: If the expected reply message was not received yet, the agent should wait for this particular message, before going on with the next message:*

$$\forall m_1, m_2, a_1, a_2, p: (sent(a_1, m_1, a_2) \wedge validReply(m_2, m_1) \notin \varnothing \wedge \neg received(a_1, m_2, t) \rightarrow wait\text{-}for(a_1, m_2))$$
$$a_1, a_2 \in A^*,\ a_1 \neq a_2,\ m_1, m_2 \in M(p),\ m_1 \neq m_2,\ p \in P^*,\ t \in T(a)$$

*Whereby the ternary relation sent($a_1$, $m_1$, $a_2$) denotes that agent $a_1$ has sent message $m_1$ to the receiver $a_2$.*

see definition (1)

**Definition 5a. (execution condition):** *Given some message $m_2$ which has been received while agent a is waiting for message $m_1$ of the same sender, and message $m_2$ matches to $m_1$ (message performative and conversation ID of $m_2$ are the same as in $m_1$) the execution condition is defined as the following behavior of the agent: agent a does not need to rollback, process the message and remove wait-for condition:*

$$\forall a, m_1, m_2, p, t: (wait\text{-}for(a, m_1) \wedge received(a, m_2, t) \wedge matched(m_1, m_2) \rightarrow \neg rollback(a) \wedge process(m_2) \wedge \neg wait\text{-}for(a, m_1))$$

$$a \in A^*,\ m_1, m_2 \in M(p),\ p \in P^*,\ t \in T^*$$

see definition (1),(3b),(4)

**Definition 5b. (delayed execution condition):** *A delayed execution condition is defined by some message $m_2$ which is received while agent a is waiting for message $m_1$ from a sender different to the sender of $m_2$, whereby $m_2$ is not part of the current interaction protocol p. In this case the message is buffered.*

$$\forall a,m_1,m_2 , p, t:(\text{wait-for}(a, m_1) \wedge \text{received}(a, m_2, t) \wedge \text{sender}(m_1) \neq \text{sender}(m_2) \wedge m_1 \in M(p) \wedge m_2 \notin M(p) \rightarrow \text{bufferMessage}(a , m_2))$$

$$a \in A^*, m_1,m_2 \in M^*, p \in P^*, t \in T^*$$

see definition (1),(3b),(4)

We have to avoid *deadlock* situations, where agent $a_1$ waits for agent $a_2$, and to the same time agent $a_2$ waits for agent $a_1$, since both independently have sent a message to each other. Both agents are blocked; each is waiting for a message event which will never occur. Hence, the agent must also process a message from its opponent even if it is not belonging to the protocol it currently processes:

**Definition 6. (deadlock avoidance condition):** *If some message $m_2$ is received while agent a is waiting for message $m_1$ of the same sender, but $m_2$ does not belong to the current interaction protocol ($m_2 \notin M(p)$), and agent a doesn't need to rollback, then the deadlock avoidance condition is defined as the behavior of processing the new $m_2$ despite of any wait-for condition:*

$$\forall a, m_1 ,m_2 , p, t: (\text{wait-for}(a,m_1) \wedge \text{received}(a,m_2,t) \wedge (\text{sender}(m_1) = \text{sender}(m_2)) \wedge m_1 \in M(p) \wedge m_2 \notin M(p) \wedge \neg \text{rollback}(a) \rightarrow \text{process}(a , m_2))$$

$$a \in A^*, m_1,m_2 \in M^*, p \in P^*, t \in T(a)$$

See definition (1),(3b),(4)

**Definition 7. (consideration of cyclic dependencies):** *If some message $m_2$ is received while agent a is waiting for message $m_1$ with a sender different to the sender of $m_2$, and the new message $m_2$ does belong to the current interaction protocol ($m_2 \in M(p)$), and agent a doesn't need to rollback, then the condition consideration of cyclic dependencies is defined as the behavior of processing $m_2$ in despite of any wait-for condition, since the protocol could not proceed otherwise:*

$$\forall a, m_1, m_2, p, t:(\text{wait-for}(a, m_1) \wedge \text{received}(a, m_2 , t) \wedge \text{sender}(m_1) \neq \text{sender}(m_2) \wedge (m_1,m_2 \in M(p)) \wedge \neg \text{rollback}(a) \rightarrow \text{process}(a , m_2))$$

$$a \in A^*, m_1,m_2 \in M^*, p \in P^*, t \in T(a)$$

see definition (1),(3b),(4)

Finally, cyclic dependencies have to be handled adequately. Although situations as described in the following are not very probable, there may occur situations when for example in multi-staged-protocols an agent receives a request within the same conversation, from a new communication partner different from its original opponent. In such a case, this message of course must be processed before going to wait state again (see def. 7).

Accordingly, on each message send agent $a_i$ executes:

```
send(message msg) to a_n{...
    if (msg.requiresReply()≠∅) then{
       waitForReply:=true;      //[def.4]
       MsgTemplate = msg.createReplyTemplate();
    }...
}
```

When a message is received by agent $a_i$ it exectues:

```
receive(message msg){...
     if (LVT(a_i)>msg.Time)∨ msg.isANTIEVENT = true)
         then rollback();          //[def.3]
     else
     if(waitForReply = true) then
        if (MessageTemplate.match(msg)=true) then
        {
            waitForReply:=false;
            process(msg);          //[def.5a]
        }
        else
        if(msg.Sender = MsgTemplate.Sender)
            then process(msg);     //[def.6]
        else
        if(msg.ProtocolID = MsgTemplate.ProtocolID)
            then process(msg);     //[def.7]
        else
            bufferMessage(msg);    //[def.5b]
     }
     else process(msg); //no wait condition was set
}
```

Now remembering the example from *section 2*, where a straggler message caused the agent to rollback and recover to an earlier point in time. With the new policy in place such situations could easily be avoided. Now the agent waits until it gets a reply message, and not simply processes with the next message (see *figure 2*). As shown above, delayed execution helps to preserve event order and therefore avoids wrong computation. Of course, such a policy cannot fully prevent rollback situations. In fact, there are conceivable cases, where this policy may fail. But it seems to be at least fairly better than a pure TW solution, without fully loss of parallelism, and is considered within the evaluation of this approach. Furthermore 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 stage only. Depending on the pro-

tocol length, the policy is applied more frequently. Particular long interaction protocols, like the *fipa-contract-net* are most eligible (see *Table 1*). Since this approach is joining optimistic techniques with constrained optimism, we actually pursue a *hybrid* time management approach.

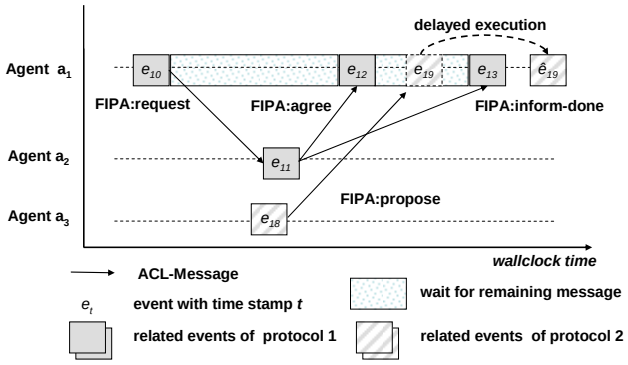


**Fig. 2.** Delayed event execution based on protocol information. Agent $a_1$ receives a proposal from Agent $a_3$, while he is waiting for an *inform-done* message of Agent $a_2$. Instead of immediately processing the incoming message, the execution is delayed. Thus, event order is preserved and still valid.

**Table 1.** Communication complexity for some standard FIPA Agent Interaction Protocols (m – number of participants) [14]

| Protocol name | Min. message number | Max. message number |
|---|---|---|
| Propose | 2 | 2 |
| Request | 2 | 3 |
| Query | 2 | 3 |
| Contract Net | 2m | 5m |

## 4 The *SimJade* Synchronization Service

A prototype implementation of the proposed synchronization service was realized by using the *Java Agent Development Environment* (JADE) [15]. This framework offers an appropriate middleware to simplify the implementation of multiagent systems. Beyond this, it is widely used in academia. As one part of this agent toolkit, there are ready-to-use behaviour objects for standard interaction protocols such as *fipa-request* and *fipa-query-ref*. By supporting generic interaction protocols, application developers just need to implement domain specific actions, while the framework will carry out all application independent protocol logic. Since JADE is FIPA compliant, a high degree of interoperability is guaranteed. To test our approach, we have implemented an extension named *SimJade* to support a local synchronization scheme. This service implements optimistic Time Warp based synchronization algorithm, first introduced by *Jefferson* and discussed in [10]. Each agent is equipped with a local control mechanism for event scheduling. Furthermore, a dedicated synchronization service,

which is integrated into the agent platform, is provided. This service offers the functionality of computing state copies of an agent as well as recovering a former agent state. A specialized simulation manager agent implements global control mechanisms, like memory reclaiming, starting and stopping simulation of experiments, as well as detecting termination of simulation runs. For distributed computation of the global virtual time, a procedure based on a snapshot algorithm first proposed by *Mattern* has been implemented [16]. Unused memory is reclaimed by using fossil collection [10]. Most of the described functionality is transparent to the agent developer. This is realized by encapsulating all time management functionality within a single agent superclass. Using the new service only requires that the domain agents are inherited from this new agent class instead of the default agent class.

Since *SimJade* synchronization service is based on a widespread agent toolkit it enables the testing of multiagent systems developed within JADE before they are deployed in the real world. Moreover using an optimistic synchronization scheme relieves the agent developer from most technical issues associated with time management. Thus, the programmers can concentrate their efforts on implementing the domain specific application logic.

## 5 Evaluation

To evaluate the proposed time management policy, we performed a number of experiments using the *SimJade* service together with a self-defined agent model. Our test environment consists of a mini-cluster with five P4 2.8 GHz workstations (256 MB RAM, *Suse Linux 8*) which are connected by a 100 Mbit switched Ethernet.

The test model comprises of 30 agents. In a single run those agents together exchange over 2000 event messages. To emulate a realistic workload, each agent in our evaluation scenario implements standard reasoning capabilities using *JESS*-engine and the respective behaviour for integration in JADE [17]. The *JESS* contribution can easily be combined with the generic interaction protocols provided by JADE [15]. If an agent process receives a message, the JESS inference engine, to create a suitable reply-message, first interprets this message. A combination of different standard FIPA protocols (see *Table 1*) with mixed communication lengths where used within each run. We tracked the wall clock time required to finish as simulation run, as well as the total number of rollbacks to measure the efficiency of our approach. *Figure 3* shows the obtained lead times. For orientation, the dotted line shows the cycle time using sequential synchronization scheme. As can be seen, our time management policy clearly outperforms the pure time warp implementation in this scenario. Even more, a remarkable speedup could be gained, compared to sequential synchronization approach. The last fact is not self-evident, since results are crucially determined by the parallelism of the application and even more by the synchronization overhead caused by distribution. The main reason for these good performance results probably grounds on the small number of rollbacks compared to none-constrained optimistic synchronization policy (*Figure 4*). Within the pure Time Warp solution agents tend to be too far ahead of each other in simulation time, and therefore more frequently causing rollbacks. By deploying message delays we could minimize the probability of

time consuming rollbacks, at least for the test scenario. To summarize, there is a clear connection of rollback frequency and obtained speed up, whereby our time management policy seems to have an advantage over the pure time warp based solution.

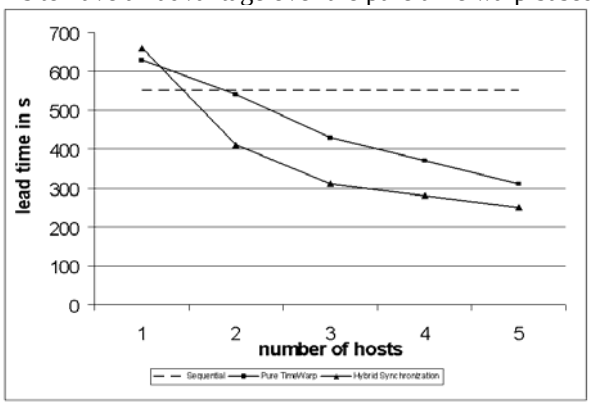


**Fig. 3.** The figure shows the arising average execution times with and without an advanced time management policy in place. The dotted line indicates the lead time reached by using a sequential (none distributed) synchronization policy.

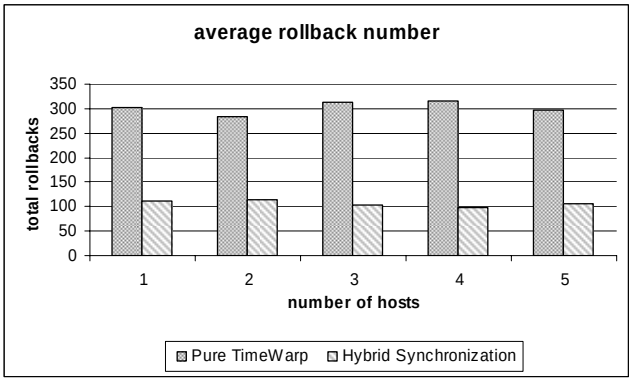


**Fig. 4.** The figure depicts the total number of rollbacks within a single simulation run comparing pure time warp solution with our advanced synchronization approach.

## 6 Related Work

Many different agent-based simulation toolkits have been proposed in recent years. So far, the majority neither offer a support to distributed simulation nor provide any advanced time management schemes.

One of the first general purpose frameworks for agent-based simulation is the GENSIM system by Anderson and Evans [18], where agents are given perceptions at fixed intervals, and with a fixed amount of time to react to each perception. Within the distributed version, DGENSIM a time-stepped synchronization mode is used [19]. A support for swarm-like simulation is realized within SWAGES simulation envi-

ronment [20]. This framework gives support to parallel execution of agent-based models and dynamic load balancing. Therefore, simulated agents are referred to a spatial model. Agents are allowed to act asynchronously within their event horizon, but have to consider whenever a non-local entity could potentially influence them, and potentially affects the sphere of influence of an agent.

The SYNCER framework is another remarkable contribution, that enables running a distributed simulation with the well known SWARM agent toolkit based on a time stepped time management approach, whereas remote communication is realized via proxy objects located on each computational node [21]. The simulators used in the RoboCup Simulation Leagues like rescue and 2D soccer simulator use a fixed step discrete time model [22]. In [5], a sequential synchronization service for linking different agent simulation test beds is presented. Popov et al [223] again describe a parallel sequential simulation approach to simulation of $10^6$ agents to capture the behaviour of web users. This vast number of simulated agents is reached by keeping the agent implementation unchanged. Furthermore they are using a relative weak notion of agency, and do not consider deliberative agent structures.

A conservative synchronization approach is discussed as part of the MPADES framework, a middleware for building (distributed) simulation environments [24]. Furthermore there are some contributions in place concerning the distributed simulation through federation of agent-based simulation environments using the *High Level Architecture* (HLA), a generic, language-independent specification, that allows the integration of different sequential simulators, and was originally initiated by the *Department of Defence* and meanwhile committed as an ISO standard [25],[26]. With HLA simulators, referred to as federates, can be integrated into a global simulation context known as federation. A strict hierarchical tree-oriented model is used to structure respective federations. Communication between certain simulators is enabled by predefined gateways. Thus HLA can be considered as a centralized coordination approach to distributed simulation resources. HLA_AGENT for example introduces support for distributed simulation to the SIMAGENT toolkit [26]. HLA_REPAST is another distributed simulation environment that uses HLA to parallelize simulation of the artificial life-toolkit REPAST [25]. HLA clearly focuses on the interoperability between different sequential simulation toolkits and is not designed to gain speedups.

To enable the simulation of large-scale agent system the MACE3J system by Gasser and Kakugawa provides services, for registration, scheduling and messaging of so-called *ActivationGroups* [27]. These groups again consist of *Active Objects*, i.e. the agents within the system. Scaling up is on main design criteria of the system. The time management relies on a conservative synchronization regime. MACE3J has been run with up to 5000 agents on a shared memory system too prove scalability; admittedly, agents have not changed any messages in this test. In [28] scalable multi agent simulation using the grid approach is discussed. This contribution clearly aims on providing an infrastructure for distributed simulation, without treating synchronization issues in detail.

The first simulation framework that gives support to optimistic time management is the well-known JAMES system [29]. Actually, there is no report about using any adaptive optimism schemes. Furthermore, JAMES does not support a general simulation model, since it comes with predefined agent architecture. Another contribution

from Logan et al proposes a metric to compute the degree of optimism based on the shared state of the agent system. This metric is used to define a *moving time window* for constraining optimism. To test their approach they are using an external library together with the SIM_AGENT toolkit [11]. This approach is comparable to our simulation service, since it relies on the idea of constraining the optimism of the simulation model.

To enable testing to a wide range of real world agent applications Helleboogh et al propose a semantic duration model to capture timing requirements that reflect the semantic of agent activities in an explicit way [30]. This time management approach is primary meant to ensure causality within a simulation run. Beyond this, it is not supporting efficient simulation or distributed simulation by default. At last several time management policies have been introduced within the context of conventional parallel discrete event simulation [10]. However, these approaches of course do not consider the deviations of agent-based simulation approach effectively, but could be easily combined with our policy.

Although MABS has received a lot of attention in recent years there are only a few contributions that deal with the problem of efficient time management. Moreover, only a small set of agent-based simulation toolkits does support distribution of simulation over multiple hosts. In fact, most currently existing simulation environments support a simple time-stepped model, which is inappropriate to simulate multitude real world system in reliable manner (see section 3). Namely, the JAMES simulation toolkit [29] and contributions by Lees and Logan [31] explicitly make use of an optimistic synchronization scheme within the context of multi-agent-based simulation. Beside this, nearly all existing general-purpose simulation frameworks are lack to be compatible with the FIPA standard. Mostly, they refer to a particular type of agent model respectively application field, like artificial life or social science. Since they oblige to some particular agent architecture, there is a low degree of freedom to the application developer left.

## 7 Conclusion

Simulation is one of the key features for testing and evaluating distributed systems [1]. However, the simulation of multiagent systems or the simulation using agent-based models is still under research. It is commonly assumed, that the inherent distribution of multiagent systems could also be used for scale-up resp. speed-up simulation. However, in practice this does not has to be true. The objective of our approach outlined in this paper was to introduce a new time management approach to agent-based simulation. The approach integrates time warp synchronization with constraints. Therefore a hybrid framework for synchronization in simulation has been introduced consisting of policies to avoid excessive rollbacks as well as too high deviations in simulation time of the various agent processes. As we've pointed out, using interaction protocols within agent-based simulation can offer an appropriate way to constrain optimism of the underlying simulation model. First experimental results show a significant benefit of this hybrid approach compared to pure time warp based solution.

The evaluation of the proposed algorithm is still going on; concurrently with the integration of new simulation models from manufacturing and logistics domain. Additional performance improvements, as well as the introduction of advanced load balancing schemes are planed, to support scalable simulation for a wide range of agent based models.

## References


1. Timm, I.J.; Scholz, T.; Fürstenau, H.: From Testing to Theorem Proving. Chapter IV.8 in Kirn, S. et al. (Hrsg.): Multiagent Engineering - Theory and Application in Enterprises. Springer-Verlag (Handbuch): Berlin, (2006), pp. 531-554.
2. Weiss, G.: Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence. The MIT Press: Cambridge, Massachusetts, (1999)
3. Davidsson, P.: Multiagent Based Simulation: Beyond Social Simulation. In: Moss, S., Davidsson, P. (eds.): 3. Workshop on Multiagent Based Simulation (MABS) 2000, LNAI Vol.1997. Springer-Verlag Berlin Heidelberg New York, (2000) 97-107
4. Gentile, M., Paolucci, M., Sacile, R.: Agent-Based Simulation. In: Paolucci, M., Sacile, R.: Agent-Based Manufacturing and Control Systems: new agile manufacturing solution for achieving peak performance. CRC Press LCC, (2005)
5. Braubach, L., Pokahr, A. et al: A Generic Simulation Service for Distributed Multi-Agent Systems. In: Trappl, R. (eds.): Cybernetics and Systems 2004 (Vol. 2), Vienna, Austria, (2004) 576-581
6. Dangelmaier, W., Franke, H., et al: Agent-based Simulation of Transportation Nets. In: Coelho, H., Espinasse, B.(eds.): 5 th Workshop on Agent-based Simulation. Lisboa, Portugal, (2004) 174-179
7. Kádár, B., Pfeifer, A., Monostori, L.: Building Agent-Based Systems in a Discrete-Event Simulation Environment. In: Pechoucek, M., Petta, P., Varga L.Z. (eds.): CEEMAS 2005, LNAI Vol. 3690, Springer-Verlag Berlin Heidelberg New York, (2005) 595-599
8. Serugendo, G.D.: Engineering Emergent Behaviour: A Vision. In: D. Hales et al (eds.): Multiagent Based Simulation (MABS) 2003, LNAI 2927, (2003) 1-7
9. Klügl, F.; Herrler, R.; Oechslein, C.: From Simulated to Real Environments: How to use SeSAm for software development In: M. Schillo et al. (eds) Multiagent System Technologies - 1st German Conferences MATES, (LNAI 2831), (2003) 13-24
10. Fujimoto, R.M.: Parallel and Distributed Simulation Systems. John Wiley & Sons Inc. (2000)
11. Lees, M., Logan, B. et al: Time Windows in Multi-Agent Distributed Simulation. (2004)
12. Wang, F., Turner, S. J., Wang L.: Agent Communication in Distributed Simulations. In: Davidsson, P. et al (eds.): Workshop on Multiagent Based Simulation (MABS) 2004, LNAI Vol.3415. Springer-Verlag Berlin Heidelberg New York, (2005) 11-24
13. Timm, I.J.: Dynamisches Konfliktmanagement als Verhaltenssteuerung Intelligenter Agenten. DISKI 283 – Dissertationen Künstliche Intelligenz, infix-AKA Verlagsgruppe, (2004)
14. Foundation For Intelligent Physical Agents (FIPA): Interaction Protocol Specification Document no. SC00026H-SC00036H, http://www.fipa.org/specs/, (2002)
15. JADE Framework, http://sharon.cselt.it/projects/jade/
16. Mattern, F.: Efficient algorithms for distributed snapshots and global virtual time approximation. In: Journal of Parallel and Distributed Computing 18(4), (1993) 423-434
17. Friedman-Hill, E. J.: Jess, The Rule Engine for the Java Platform. Distributed Computing Systems, Sandia National Laboratories, Livermore, CA21 http://herzberg.ca.sandia.gov/jess

18. Anderson, J., Evans, M.: A Generic Simulation System for Intelligent Agent Designs. In: Applied Artificial Intelligence 9:5, (1995) 527-562
19. Anderson, J.: A Generic Distributed Simulation System For Intelligent Agent Design And Evaluation. http://www.citeseer.ist.psu.edu/399301.html, (2000)
20. Scheutz, M., Schermerhorn, P.: Adaptive Algorithms for the Dynamic Distribution and Parallel Execution of Agent-Based Models, (2005) http://www.nd.edu/%7Eairolab/publications/scheutzschermerhorn06pardist.pdf
21. Goic, J., Sauter, J. A., Toth-Fejel, T.: Syncer: Distributed simulations using swarm. In: SwarmFest 2001, Santa Fe, NM, (2001)
22. Homepage ot the RoboCup-Rescue Simulation Project: http://www.rescuesystem.org/robocuprescue/simulation.html
23. Popov, K. et al: Parallel Agent-Based Simulation on a Cluster of Workstation. In: Kosch, H., Böszörményi, H. Hellwagner, H. (eds.): Euro-Par 2003, Springer-Verlag Berlin Heidelberg New York, (2003) 470-480
24. Riley, P.: MPADES: Middleware for Parallel Agent Discrete Event Simulation. In: Kaminka, G.A., Lima, P.U. and Rojas, R. (eds.): RoboCup 2002, LNAI Vol. 2752, Springer-Verlag Berlin Heidelberg New York, (2003) 162-178
25. Minson, R., Theordoropoulos, G.: Distributing repast agent based simulations with HLA. In: Proceedings of the 2004 European Simulation Interoperability Workshop, Edinburgh, UK, (2004)
26. Lees, M., Logan, B.: Simulating Agent-Based Systems with HLA: The Case of SIM_AGENT - Part II (03E-SIW-076), (2003)
27. Gasser, L., Kakugawa, K. et al: Smooth Scaling Ahead: Progressive MAS Simulation from Single PCs to Grids. In: Proceedings of the Joint Workshop on Multi-Agent & Multi-Agent-Based Simulation, Autonomous Agents & Multiagent Systems (AAMAS) New York, USA, (2004) 1-10
28. Ingo J. Timm, I.J., Pawlaszczyk, D.: Large scale multiagent simulation on the grid. In: IEEE International Symposium on Cluster Computing and the Grid <5, 2005, Cardiff>, NJ: IEEE Operations Center, (2005) 334-341
29. Uhrmacher, A.M., Schattenberg, B.: Agents in Discrete Event Simulation. In: Bargiela, A., Kerckhoffs, E. (eds.): Proceedings of the 10TH ESS'98, SCS Publications Ghent, (1998) 129-136
30. Helleboogh, Holvoet, T. Weyns, D.: Extending Time Mangement Support for Multiagent Systems. In: Davidsson et al. (eds): Multi Agent Based Simulation (MABS) 2004, LNAI 3415, (2005) 37-48
31. Lees, M., Logan, B., Theodoropoulos, G.: Adaptive optimistic synchronisation for multi-agent distributed simulation. In: Proceedings of the 5th EUROSIM Congress on Modelling and Simulation (EuroSim'04), (2004)