Extending Time Management Support for Multi-agent Systems

Alexander Helleboogh, Tom Holvoet, Danny Weyns, and Yolande Berbers

AgentWise, DistriNet, Department of Computer Science K.U.Leuven, Belgium
{Alexander.Helleboogh, Tom.Holvoet, Danny.Weyns,
Yolande.Berbers}@cs.kuleuven.ac.be

Abstract. Time management is essential when simulating multi-agent systems (MASs) as it allows consistent and repeatable simulation runs. So far, time management lacks support to express the timing requirements of a simulation explicitly and at an abstraction level appropriate for MAS developers. Moreover, integrating time management into a MAS requires the developer to alter the design of the MAS. In this paper, we first propose *semantic duration models* to capture timing requirements that reflect the semantics of MAS activities in an explicit model. Second, we present a time management infrastructure that starts from a semantic duration model description to integrate all time management functionality into a MAS transparently, i.e. without requiring the developer to alter the design of the MAS. We use aspect-oriented programming technology as it allows *separation of concerns*, a crucial software engineering requirement. As a case, we apply our approach to the Packet-World.

1 Introduction and Problem Statement

Simulation platforms enable multi-agent systems (MASs) to be tested before they are deployed in the real world. An important requirement for such platforms is that a MAS can easily be integrated with the simulation infrastructure. The developers have to be relieved from the low-level technical issues associated with simulations [1]. This allows the developer to concentrate his or her efforts on the relevant domain application logic.

An essential technical issue which has to be provided by a simulation platform is *time management* [2]. Time management ensures that all temporal characteristics of the problem domain are correctly reproduced in the simulation. Time management is required in simulation platforms to allow controlled and repeatable simulation runs.

Currently, time management is generally supported by means of time management mechanisms [2, 3, 4] which are built into the simulation platforms. Time management mechanisms are necessary to enforce all simulation events to be processed in time-stamp order, irrespective of arbitrary and variable delays in the execution platform. Examples of time management mechanisms are time-stepped execution and conservative or optimistic event synchronization mechanisms. When time management mechanisms prevent the execution platform

P. Davidsson et al. (Eds.): MABS 2004, LNAI 3415, pp. 37–48, 2005.

from introducing causality errors, the consistency and repeatability of a simulation can be guaranteed.

Time management is also essential in the context of simulating MASs [5], because timing delays introduced by the underlying execution platform may otherwise affect the simulation results. For example, in [6, 7, 8] it is shown that alterations in the execution platform of the agents can have a severe impact on the simulation behavior of the MAS as a whole, possibly introducing unexpected and unwanted behavior.

MASs allow a system to be modeled at a high level of abstraction. Therefore, it is essential that the support for time management in simulation platforms is raised to an abstraction level appropriate for MAS developers. Currently, time management *mechanisms* are built into the simulation platforms to hide the technical issues related to maintaining logical time consistency. Nevertheless, a MAS developer is still confronted with a number of unsupported time management issues when simulating a MAS. First, there is a lack of support to express the relation between the activity within a MAS and logical time in an *explicit* way. Outside a simulation context, the concept of logical time is hardly ever employed: agents are generally not designed as entities maintaining a logical clock and generating time-stamped events. If such systems are simulated, the mapping to logical time has to be tackled by the developer without any support, since time management mechanisms require that the time stamps are already assigned to the events, and only provide support for time stamp ordering. A second problem is the lack of support to integrate all time management functionality into a MAS. Currently, this integration requires the developer (1) to reimplement each agent's actions on the environment to transform them into time stamped events and (2) to direct these events to the simulation platform [9, 10]. Besides the fact that this requires a fair understanding of the simulation platform and its interfaces, it also forces developers to alter the design of the MAS.

This paper describes a way to extend time management support for simulating MASs in order to deal with the problems mentioned above. We give a high-level overview of our approach, based on Fig.1. First, we employ Semantic Duration Models to provide support for the developer to make the timing requirements for the simulation of a MAS *explicit*. Semantic duration models enable the developer to express the mapping of the activity within a MAS to logical time at a high level of abstraction, allowing the semantic meaning of MAS activities to be taken into account. Second, we describe the Time Management Infrastructure we developed. Our prototype allows time management to be integrated in a MAS *transparently*, i.e. without requiring the developer to make design changes in the MAS or to have any knowledge from the simulation platform and its interfaces. Our approach employs aspect-oriented programming to achieve *separation of concerns*. Separation of concerns is important from a software engineering point of view, as this allows all time management functionality needed for simulation purposes to be decoupled from the MAS's functional structure. Based on the description of a semantic duration model, aspect-oriented programming allows time management functionality to be "woven" into a MAS.

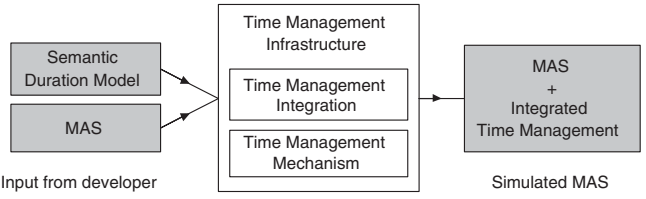

Fig. 1. Overview of the Time Management Infrastructure for simulating MASs

This paper is structured as follows. We first elaborate on semantic duration models in Sect.2 and present a basic formalism based on set theory to describe semantic duration models. Next, the time management infrastructure is described in Sect.3. Section 4 demonstrates our approach using the Packet-World as a case, after which we draw conclusions in Sect.5.

2 Semantic Duration Models

To obtain meaningful simulation results, it is essential that the timing requirements for a simulation reflect the timing characteristics of the MAS's problem domain. We describe how *semantic duration models* can support a developer to capture all timing requirements of a MAS simulation in an explicit way and at the semantic level of a MAS.

Semantic duration models capture the timing characteristics for simulating a MAS in an explicit way, using the technique of *duration modeling* at a semantic level. The idea of duration modeling is to maintain a logical clock for each agent and advance that clock for each "primitive" that is executed by the agent. The duration of a "primitive" performed by an agent is the (logical) time period it takes until the effects of that "primitive" are noticeable. The developer has to describe all timing characteristics by means of assigning logical durations to each of the "primitives". Advancing the logical clock in a way that is independent of computer loads and processor speeds, enables repeatable simulation results.

Duration modeling was first described by Anderson and Cohen in [11, 12], where it was applied in the context of the agent's deliberation activity. Anderson distinguishes between low-level and high-level duration models. In low-level models, durations are assigned to individual programming language instructions. However, this results in timing characteristics of a MAS simulation that are described in terms of low-level implementation issues. Because in a problem domain it is the semantics of what the agent is actually doing that determines the timing characteristics, Anderson emphasizes high-level duration models. For example, "evaluating a board position" for a chess playing agent, or "generating an internal plan to reach a particular destination" can be considered as primitives with semantic meaning for duration modeling in a high-level model. However, Anderson's approach is limited to modeling the agent's deliberation activity, and does not take into account other forms of activity within a MAS.

Duration modeling is also addressed in the SPADES system by Riley and Riley [13]. Their approach is not limited to modeling the duration of agent deliberation, but also incorporates the agent's sensing and acting activities. This allows the duration of perception and agent actions to be taken into account. However, in contrast to Anderson's work, the logical thinking time of the agents is now based on the measurement of CPU-time. Moreover, the approach can only be applied to agents whose architecture supports a rigid sense-think-act cycle.

Our notion of semantic duration models combines the best ideas of both approaches described above. First, analogous to the high-level models of Anderson, we consider the "primitives" of duration modeling at the level of activities with a semantic meaning in the behavior of an agent. As a consequence, the duration of each of the activities depends upon the semantic meaning within the context of the simulation only, and is irrespective of the programming language and implementation. Second, analogous to the SPADES system, we extend duration modeling from agent activities employed for deliberation purposes, to activities an agent can perform on the environment. In our semantic duration models, we make a distinction between the agent's internal and external activities. *Internal activities* are typically related to deliberation and do not cross the agent's boundaries. *External activities* on the other hand cross the boundaries of an agent and typically include perception of the environment, sending or receiving communication messages and performing actions on the environment. In contrast to the sense-think-act cycle employed in the SPADES system, we impose no order on the agent's internal and external activities.

In our current model, we assume that an agent is the unit of concurrency. As such, each agent can only perform one activity at the same time. However, activities performed by different agents can of course be concurrent.

We describe semantic duration models using a basic form of set theory:

$A = \{a_1, a_2, \ldots, a_n\}$, the set of all agents in the MAS:
$\forall a_i \in A$:
$D_i = \{d_1^i, d_2^i, \ldots, d_{n_i}^i\}$, the set of all internal activities of agent a_i
$E_i = \{e_1^i, e_2^i, \ldots, e_{m_i}^i\}$, the set of external activities that agent a_i
$D_i \cap E_i = \phi$

By combining sets D_i and E_i we obtain:

$\forall a_i \in A$:
$C_i = E_i \cup D_i = \{e_1^i, e_2^i, \ldots, e_{m_i}^i, d_1^i, d_2^i, \ldots, d_{n_i}^i\}$, the set of all activities of a_i
or $C_i = \{c_1^i, c_2^i, \ldots, c_{u_i}^i\}$ with $|C_i| = u_i = m_i + n_i$, the cardinality of C_i

To obtain a semantic duration model for an agent, the duration of all its activities is expressed in terms of logical time. Formally this is equivalent to a function assigning a logical duration to each activity:

$$Duration_i : C_i \times S_i \times W \rightarrow \Re$$
$$Duration_i(c_j^i, s_i, w) = r_j^i$$

where S_i is the set of all states of agent a_i, W is the set of all states of the environment, $\Re$ is the set of real numbers and $Duration_i$ is the semantic duration function for agent a_i. $Duration_i$ defines the logical time period it takes until the effects of activity c_j^i performed by agent a_i are noticeable, given that the state of agent a_i is s_i and the state of the world is w. In general, the duration of a particular activity for an agent not only depends on the kind of activity, but also on the state of the agent as well as on the state of the environment.

3 Time Management Transparency

In order to integrate time management into a MAS transparently, the following requirements have to be fulfilled. First, explicit and developer-friendly support for describing semantic duration models must be provided to the developer. The developer should only describe the internal and external activities and their semantic durations (see Sect.2). Based on this, the platform should be able to enforce the time mapping without further intervention from the developer. Second, it must be possible to simulate a MAS without requiring the developer to perform changes in the design of the MAS. However, because time management requires monitoring and controlling the activities of all agents according to user-defined timing characteristics, it requires introducing code in many places across the system. We could refactor all the code and perform the appropriate insertions, but in a large MAS, this would be a time-consuming and error-prone job, which we would like to avoid.

3.1 Aspect-Oriented Programming

Time management is a *crosscutting concern*, i.e. the time management functionality cross-cuts the MAS's basic functionality. The problem of crosscutting concerns is that they can not be modularized with traditional OO-techniques. This forces the implementation of time management to be scattered throughout the code of the MAS, resulting in "tangled code" that is excessively difficult to develop and maintain. Aspect-oriented programming [14, 15] handles crosscutting concerns by providing *aspects* for expressing these concerns in a modularized way. An aspect is a modular unit of crosscutting implementation. Aspect-oriented programming does not replace existing programming paradigms and languages, but instead, it can be seen as a co-existing, complementary technique that can improve the utility and expressiveness of existing languages. It enhances the ability to express the separation of concerns which is necessary for well-designed, maintainable software systems.

A language extension to Java which supports aspect-oriented programming, is AspectJ. In AspectJ, defining an aspect is based on two main concepts: pointcuts and advice. A *pointcut* is a language construct in AspectJ that selects particular

join points, based on well-defined criteria. Each *join point* represents a particular point in the execution flow of a program where the aspect can interfere, e.g. a point in the flow when a particular method is called. As such, pointcuts are a means to express the crosscutting nature of an aspect. *Advice* on the other hand is a language construct in AspectJ that defines additional code that runs at join points specified by an associated pointcut. An aspect encapsulates a particular crosscutting concern and can contain several pointcut and advice definitions. The process of inserting all crosscutting code of an aspect at the appropriate join points within the original program code, is called *aspect weaving*. Aspect weaving is performed at compile-time in AspectJ.

3.2 The Prototype

According to the requirements above, we developed a prototype in Java which uses AspectJ to integrate time management as a separate concern. We illustrate its working using Fig.2.

To be able to use time management support, the developer composes a particular *Semantic Duration Model Configuration* which describes a semantic duration model for each agent within the MAS (see Fig.2). Currently, in our prototype abstraction is made from the state dependency in semantic duration models. As a consequence, $Duration_i$ is simplified to:

$$Duration_i : C_i \rightarrow \Re$$
$$Duration_i(c_j^i) = r_j^i$$

This allows $Duration_i$ to be described in terms of a list of (c_j^i, r_j^i)-tuples for each agent a_i, with c_j^i mapping to a Java method that the agent executes to perform a particular activity with semantic meaning, and r_j^i a constant denoting the logical duration of that activity.

After a semantic duration model has been defined for each agent in the MAS, the prototype generates an *Aspect* and a *Time Monitor* for each agent. The *Time Monitor* of agent a_i contains a logical clock for the agent, together with the time mapping as described by $Duration_i$ of that agent (which maps c_j^i to r_j^i). The goal of a *Time Monitor* is to keep the agent's logical clock up-to-date by advancing it according to the activities the agent decides to perform. When the *Time Monitor* is notified of the execution of activity c_j^i, it advances its clock by r_j^i. The goal of the *Aspect* on the other hand is to notify the *Time Monitor* of all activities the agent executes. Therefore, the *Aspect* weaves code into all methods that are defined as activities c_j^i of the agent. The goal of the inserted code is to intercept the execution of the agent as soon as it decides to perform an activity and to notify the *Time Monitor*, such that the agent's logical clock is advanced appropriately. The notification of the *Time Monitor* by the inserted code is represented graphically by the arrowed lines in Fig.2.

The combination of *Aspects* and *Time Monitors* allows the logical clock of all agents to advance according to all executed activities. A *MAS Time Synchronizer* prevents the occurrence of causality errors. The developer can specify a subset

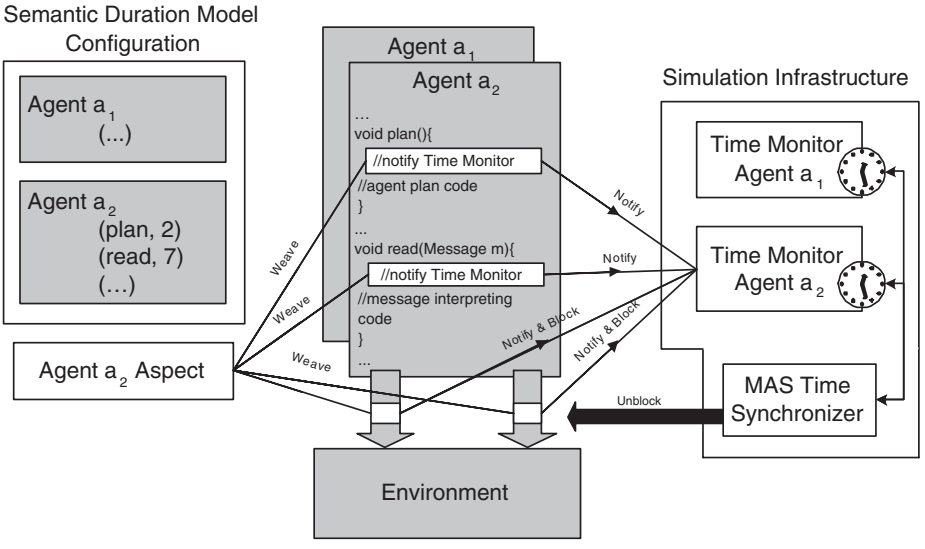

Fig. 2. Time Management Infrastructure for MASs: the gray shaded parts have to be provided by the developer. All white parts are hidden from the developer

of activities that can introduce causality errors and hence have to be controlled by the *MAS Time Synchronizer* to ensure that these activities are not executed out of logical clock order. By default, the set of activities for which causality has to be preserved contains all external activities, because these activities cross the agent's boundaries (see Sect.2). In Fig.2, the gray arrows between the agent and the environment represent external activities the agent can perform on the environment.

We explain the approach employed for synchronization by using an example. Suppose that a particular agent decides to perceive its neighboring environment and triggers an external perception activity. The code inserted by the *Aspect* intercepts the execution of the agent right before the chosen activity is actually executed, notifies that agent's *Time Monitor*, which advances that agent's logical clock with the appropriate duration and then blocks that agent's execution. Unblocking can only be done by the *MAS Time Synchronizer*, which monitors the logical clocks of all agents and employs a conservative time management mechanism [3] to prevent causality errors. The specific way of interception ensures that the logical clocks of the agents are already updated before the corresponding activities are actually executed. This enables the *MAS Time Synchronizer* to have prior knowledge of the time stamp of the next activity a particular agent will perform. As a consequence, the synchronization approach applied here does not rely on a lookahead to prevent starvation. In our example, the perception activity of the agent will be unblocked as soon as the *MAS Time Synchronizer* can guarantee that all external activities the other agents will perform, have a higher logical time stamp than the perception activity of the former agent.

As such, the former agent perceives the environment in correspondence to the causal order that arises from the semantic duration models.

4 Time Management Applied in the Packet-World

In this section, we illustrate our approach by means of the Packet-World application we have developed [16]. We describe a semantic duration model and demonstrate how time management functionality is integrated transparently.

4.1 The Packet-World

The Packet-World consists of a number of differently colored packets that are scattered over a rectangular grid. Agents that live in this virtual world have to collect those packets and bring them to the correspondingly colored destination. The grid contains one destination for each color. Figure 3 shows an example of a Packet-World with size 10 wherein 5 agents are situated. Squares symbolize packets and circles are delivery points.

In the Packet-World, agents can interact with the environment in a number of ways. We allow agents to perform a number of basic actions. First, an agent can make a step to one of the free neighboring fields around it. Second, if an agent is not carrying any packet, it can pick one up from one of its neighboring fields. Third, an agent can put down the packet it carries on one of the free neighboring fields around it, which could of course be the destination field of that particular packet. It is important to notice that each agent of the Packet-World has only a limited view on the world. This view only covers a small part of the environment around the agent (see Fig.3). Furthermore, agents can interact with other agents too. We allow agents to communicate with other agents by sending messages. In this way, agents can inform each other about the position of packets and destinations. All action and message handling is performed by the environment.

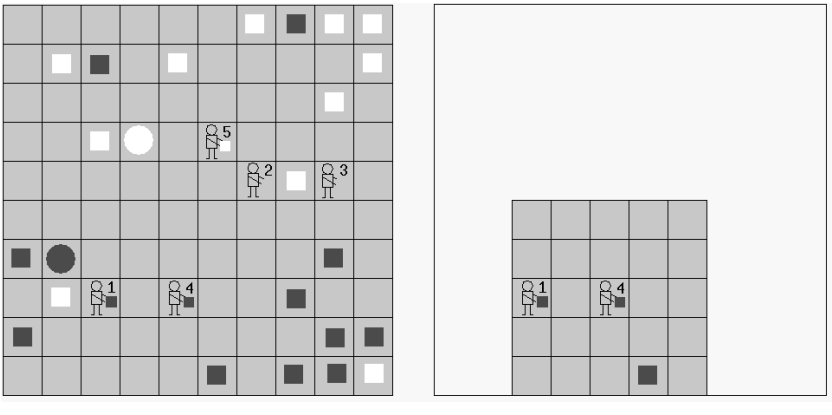

Fig. 3. The Packet-World: global screenshot *(left)* and view range of agent nr.4 *(right)*

4.2 Timing Requirements for the Simulation

In the Packet-World, each agent is an autonomous and pro-active entity which continuously deliberates and invokes actions in the environment. Neither timestamps, nor events are employed in the agents' design. However, for our simulation, we would like the agents to behave according to specific timing characteristics. Suppose we impose the following timing requirements on the agents: first, picking up or putting down a packet only takes half the time for an agent than performing a step. On the other hand, obtaining perception of the environment or retrieving messages which have arrived, can be done instantaneously. The time it takes for an agent to analyze its perception cannot be neglected. Searching for a destination field based on the input obtained from perception takes as long for an agent as performing a pick up packet action, while finding the nearest packet based on its perception only takes half as long. The time it takes for an agent to select its next action is equal to that of performing a move. Finally, sending a message is twice as costly as performing a step.

4.3 Defining a Semantic Duration Model

We identify all agents' activities in the Packet-World simulation. Using the description above, we can distinguish the following *external activities* on the environment: an agent can (1) look to perceive its surroundings, (2) move, (3) pick up a packet, (4) put down a packet, (5) send a message, and (6) receive messages that have arrived. Formally (see Sect.2):

$$\forall a_i \in A :$$
$$E_i = \{look, move, pick, put, send, receive\}$$

With respect to the internal activities of the agents, in our simulation a distinction is made between (1) detecting a destination, (2) finding the nearest packet and (3) selecting the next action. Formally:

$$\forall a_i \in A :$$
$$D_i = \{detectdest, findpacket, selectaction\}$$
$$\text{and } C_i = \{look, move, pick, put, send, receive, detectdest, findpacket, selectaction\}$$

To define a semantic duration model, we have to assign a duration to each of the activities of an agent, according to the timing requirements of the simulation. We get:

$$\forall a_i \in A :$$
$$Duration_i(move) = Duration_i(selectaction) = 1$$
$$Duration_i(pick) = Duration_i(put) = Duration_i(detectdest) = 0.5$$
$$Duration_i(look) = Duration_i(receive) = 0$$
$$Duration_i(findpacket) = 0.25$$
$$Duration_i(send) = 2$$

Note that the absolute values of the durations are of no importance, only the relative values are significant.

4.4 Integrating Timing Management Code

For each activity described in the semantic duration model of the Packet-World agents, time management code has to be inserted. As an example, we consider the *findpacket* internal activity of an agent (see Fig.4). Based on the semantic duration model described above, an aspect is generated for the *findpacket* activity. The pointcut of the aspect refers to the location of the *findpacket* activity in the agent's code. At this location, the aspect's advice is woven which notifies the agent's time monitor each time the activity is performed.

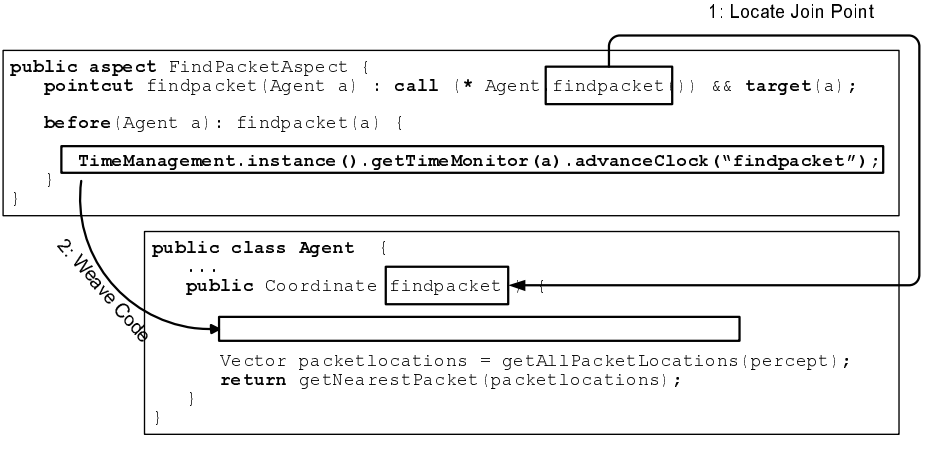

Fig. 4. Aspect weaving for the internal activity *findpacket*

5 Conclusions and Future Work

In this paper, we described a way to extend time management support for simulating MASs. Our contribution consists of two parts.

First, semantic duration models allow the timing requirements of a simulation to be described in an explicit way by means of a user-friendly formalism based on set theory. Semantic duration models employ the technique of duration modeling at a semantic rather than syntactic level and allow timing requirements to be expressed for the internal as well as the external activities of an agent.

Second, we described a time management infrastructure that allows all time management functionality to be integrated transparently in a MAS. The developer describes all timing requirements by means of semantic duration models. To achieve separation of concerns, which is important for well-designed and maintainable software systems, aspect-oriented programming is used. Our prototype

allows all time management code necessary for the simulation to be incorporated in the MAS without requiring the developer to change the design of the MAS.

In the paper, we demonstrated our approach in the Packet-World. It was shown that it is possible to control the execution of the simulation according to specific timing requirements and to integrate time management functionality in a transparent way.

Although the approach presented here is promising, a number of issues requires further research and will be addressed in detail in future work.

- With respect to the semantic duration models, we exclusively elaborated upon agent activities, both internal and external. However, activities can also originate from the environment of the MAS, independent of the agents. An example are digital pheromones [17] that propagate and evaporate over time. Pheromones are used for indirect communication in MASs. Our approach requires further investigation with respect to such environmental activities in general.
- In the current model, there is no support to allow overlap of activities, as described in [13]. All activities of an individual agent happen sequentially. An important issue we are currently working on is extending the semantic duration model of an agent such that activities can be specified to be potentially overlapping.
- In our prototype, the current support for semantic duration models is useful but still rather limited, since only *constant* logical durations can be assigned to activities. Extensions to more complex dependencies are planned in the future.
- Finally, there is no clean duration semantics for hierarchical activities. Suppose agent a_i has two activities: activity c_j^i with a duration of r_j^i and activity c_k^i with a duration of r_k^i, and suppose c_j^i calls c_k^i. If agent a_i then executes activity c_j^i, it is unclear whether agent a_i has to be assigned a logical delay of r_j^i as defined earlier, or $r_j^i + r_k^i$ (which is currently the case in our prototype).

Acknowledgements

This research is partially funded by the KULeuven research project AgCo2 (Agents for Coordination and Control).

References

1. Maria Bruno Marietto, Nuno David, J.S.S.H.C.: Requirements analysis of agent-based simulation platforms: State of the art and new prospects. In: Multi-Agent-Based Simulation, Third International Workshop, MABS 2002. Lecture Notes in Computer Science, Springer-Verlag (2002)
2. Fujimoto, R.: Time management in the high level architecture. Simulation, Special Issue on High Level Architecture **71** (1998) 388–400
3. Chandy, K.M., Misra, J.: Asynchronous distributed simulation via a sequence of parallel computations. Communications of the ACM **24** (1981) 198–205

4. Jefferson, D., Sowizral, H.: Fast concurrent simulation using the time warp mechanism. In: Proceedings of the SCS Multiconference on Distributed simulation. (1985) 63–69
5. Helleboogh, A., Holvoet, T., Weyns, D.: Towards time management adaptability in multi-agent systems. In Kudenko, D., Alonso, E., Kazakov, D., eds.: Proceedings of the AISB 2004 Fourth Symposium on Adaptive Agents and Multi-Agent Systems. (2004) 20–30
6. Axtell, R.: Effects of interaction topology and activation regime in several multi-agent systems. In: MABS. (2000) 33–48
7. Page, S.: On incentives and updating in agent based models. Journal of Computational Economics **10** (1997) 67–87
8. Cornforth, D., Green, D.G., Newth, D., Kirley, M.: Do artificial ants march in step? Ordered asynchronous processes and modularity in biological systems. In: Proceedings of the eighth international conference on Artificial life, MIT Press (2003) 28–32
9. Uhrmacher, A., Kullick, B.: Plug and test software agents in virtual environments. In: Winter Simulation Conference - WSC'2000. (2000)
10. Himmelspach, J., Rhl, M., Uhrmacher, A.: Simulation for testing software agents - an exploration based on JAMES. In: Proc. of the 2003 Winter Simulation Conference, New Orleans, USA. (2003)
11. Anderson, S.D., Cohen, P.R.: Timed Common Lisp: the duration of deliberation. SIGART Bull. **7** (1996) 11–15
12. Anderson, S.D.: Simulation of multiple time-pressured agents. In: Winter Simulation Conference. (1997) 397–404
13. Riley, P., Riley, G.: SPADES — a distributed agent simulation environment with software-in-the-loop execution. In Chick, S., Sánchez, P.J., Ferrin, D., Morrice, D.J., eds.: Winter Simulation Conference Proceedings. Volume 1. (2003) 817–825
14. Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In Akşit, M., Matsuoka, S., eds.: Proceedings European Conference on Object-Oriented Programming. Volume 1241. Springer-Verlag, Berlin, Heidelberg, and New York (1997) 220–242
15. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.: Getting started with AspectJ. Commun. ACM **44** (2001) 59–65
16. Weyns, D., Holvoet, T.: The Packet-World as a case to study sociality in multi-agent systems. In: Autonomous Agents and Multi-Agent Systems, AAMAS 2002, Bologna, Italy. (2002)
17. Sauter, J.A., Matthews, R., Parunak, H.V.D.: Evolving adaptive pheromone path planning mechanisms. The First International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2002 (2002)