Abstract. Time management is essential when simulating multi-agent systems (MASs) as it allows consistent and repeatable simulation runs. So far, time management lacks explicit support to express the timing characteristics of a simulation at the level of activities within a MAS. Moreover, integrating a MAS into a simulation platform requires the developer to alter the design of a MAS. In this paper, we first propose semantic duration models to capture timing characteristics that reflect the semantics of MAS activities in an explicit model. Second, we describe a simulation platform prototype that supports semantic duration models and integrates time management into a MAS without requiring the developer to alter the design of a MAS. We employ aspect-oriented programming technology as it allows separation of concerns, a crucial software engineering requirement. We use the Packet World as a case to illustrate our approach.

1 Introduction and Problem Statement

Simulation platforms and test-beds enable multi-agent systems (MASs) to be simulated before they are employed 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]. As a consequence the developer can 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 in general is supported by a variety of different time management mechanisms provided to the developer [2–4]. For instance, time-stepped and event-oriented mechanisms enforce that all simulation events are processed in time stamp order, even in the presence of arbitrary delays in the execution platform. Because these mechanisms prevent the execution platform from introducing causality errors, the consistency and repeatability of a simulation can be assured.

When simulating MASs, time management is also essential because timing delays introduced by the underlying execution platform may affect the simulation results. As shown in [5–7], 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.
However, because MASs allow a system to be modeled at a high level of abstraction, it is essential that the support for time management in simulation platforms is raised to a corresponding abstraction level. Time management support generally provides particular time management mechanisms which relieve the developer from the technical issues to ensure logical time consistency. Nevertheless, a MAS developer still remains 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 a MAS and logical time in an explicit way. Outside a simulation context, the concept of logical time is hardly ever employed and agents are generally not designed as entities 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 assume 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 for integrating a MAS into a simulation platform that uses a particular time management mechanism. Currently, this integration requires reimplementing each agent’s actions on the environment to transform them into time stamped events that are directed to the simulation platform [8, 9]. 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 for simulating MASs and support the developer with respect to the problems mentioned above (see figure 1). First, we provide support for the developer to make the timing requirements for the simulation of a MAS explicit by means of a semantic duration model. Semantic duration models also enable the developer to express the mapping of a MAS to logical time at a level which exceeds programming language syntax, and allows the semantic meaning of MAS activities to be taken into account. Second, we describe a simulation platform prototype that allows time management infrastructure to be integrated into a MAS transparently, without requiring design changes in the MAS or 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 the simulation infrastructure 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. Time Management for Simulating MASs. The gray parts have to be provided by the developer](image-url)
This paper is structured as follows. We first elaborate on semantic duration models in section 2 and present a basic formalism based on set theory to describe semantic duration models. The simulation platform prototype is described in 3. Finally, section 4 demonstrates our approach using the Packet World as a case, after which we conclude in section 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 developer has to describe all timing characteristics by means of assigning logical durations to each of the “primitives”. Because advancing the logical clock is independent of the CPU time, computer speeds can not affect the simulation results.

Duration modeling was first described by Anderson and Cohen in [10, 11], 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 agent activity.

Duration modeling is also addressed in the SPADES system by Riley and Riley [12]. 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 ideas of both approaches described above. 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, communication 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.

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

\[ A = \{a_1, a_2, \ldots, a_n\} \text{, the set of all agents in the MAS:} \]
\[ \forall a_i \in A : \]
\[ D_i = \{d_{i1}, d_{i2}, \ldots, d_{in_i}\} \text{, the set of all internal activities of agent } a_i \]
\[ E_i = \{e_{i1}, e_{i2}, \ldots, e_{in_i}\} \text{, the set of all external activities of agent } a_i \]

By combining sets \( D_i \) and \( E_i \) we obtain:

\[ \forall a_i \in A : \]
\[ C_i = E_i \cup D_i = \{e_{i1}, e_{i2}, \ldots, e_{in_i}, d_{i1}, d_{i2}, \ldots, d_{in_i}\} \text{, the set of all activities of } a_i \]
\[ \text{or } C_i = \{c_{i1}, c_{i2}, \ldots, c_{in_i}\} \text{ with } u_i = m_i + n_i \]

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:

\[ Duration_i : C_i \times S_i \times W \rightarrow \mathbb{R} \]
\[ Duration_i(c_j, s_i, w) = r_j^i \]

where \( S_i \) is the set of all states of agent \( a_i \), \( W \) is the set of all states of the environment, \( \mathbb{R} \) is the set of real numbers and \( Duration_i \) is the semantic duration operator for agent \( a_i \). In general, the duration of a particular activity \( c_j^i \) for an agent is a function of the state of the agent \( s_i \) and the state of the environment \( w \).

3 Time Management Transparency

In order to hide time management infrastructure from the developer, the following requirements have to be fulfilled. First, explicit and developer-friendly support for describing a semantic duration model must be provided to the developer. The developer only describes the internal and external activities and their semantic durations (see section 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 appro-
priate 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 crosscuts the functional structure of a MAS. The problem with cross-cutting concerns is that they cannot be modularized with traditional OO-techniques, as their code is scattered across many places in the system. Aspect-oriented programming [13, 14] handles cross-cutting concerns by providing aspects for expressing these concerns in a modularized way. The aspect describes the concern’s code together with the places in the system where it has to be inserted. The process of inserting the aspect’s code into the system is called aspect weaving. 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.

3.2 The Prototype

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

To be able to use time management support, the developer uses the Semantic Duration Model Configuration to describe a semantic duration model for each agent within the MAS. Currently, in our prototype abstraction is made from the state dependency in semantic duration models. As a consequence, $\text{Duration}_i$ is simplified to:

$$
\text{Duration}_i : C_i \rightarrow \mathbb{R} \\
\text{Duration}_i(e^j_i) = r^j_i
$$

This allows $\text{Duration}_i$ to be described in terms of a list of $(e^j_i, r^j_i)$-pairs for each agent $a_i$, with $e^j_i$ mapping to a Java method name which is executed by the agent when performing a particular semantic activity, and $r^j_i$ is a constant denoting the logical duration of the activity.

After a semantic duration model has been defined for each agent in the MAS, the platform 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 semantic time mapping $\text{Duration}_i$ of that agent (which maps $e^j_i$ 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. As a consequence, when the Time Monitor is notified of the execution of activity $e^j_i$, 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 semantic activities $e^j_i$ of the agent. The goal of the inserted code is to intercept the execution of the agent upon each activity and to notify the Time Monitor, such that agent’s logical clock can be advanced appropriately. This is represented by the arrowed lines in figure 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 when agents perform external activities on the environment out of logical clock order. As soon as an agent tries to perform an external action, the Time Monitor is notified and blocks the 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.

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 [15]. We describe a semantic duration model and demonstrate how time management infrastructure is integrated transparently.

4.1 The Packet World

The Packet-World consists of a number of different 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 their corresponding 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 neighbor fields around him. 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 figure 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 message handling is performed by the environment.

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 on the environment. There is no notion of time-stamps and events at the agents’ level of abstraction. However, for our simulation, we would like the agents to behave according to specific timing characteristics. First, picking up or putting down a packet only takes half the time of performing a step. On the other hand, obtaining perception of the environment can be done instantaneously. The time it takes for an agent to analyze its perception cannot be neglected. Detecting a destination based on its 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 for an agent to select its next action is equal to that of performing a move. To limit the number of communication messages, 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 look to perceive its surroundings, move, pick up a packet, put down a packet and talk. Formally (see section 2):

\[ \forall a_i \in A : \\
E_i = \{ \text{look, move, pick, put, talk} \} \]

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

\[ \forall a_i \in A : \\
D_i = \{ \text{detectdest, findpacket, selectaction} \} \\
and C_i = \{ \text{look, move, pick, put, talk, detectdest, findpacket, selectaction} \} \]

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:

\[ \forall a_i \in A : \\
\text{Duration}_i(\text{move}) = \text{Duration}_i(\text{selectaction}) = 1 \\
\text{Duration}_i(\text{pick}) = \text{Duration}_i(\text{put}) = \text{Duration}_i(\text{detectdest}) = 0.5 \\
\text{Duration}_i(\text{look}) = 0 \\
\text{Duration}_i(\text{findpacket}) = 0.25 \\
\text{Duration}_i(\text{talk}) = 2 \]

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 integrated. As an example, we consider the \text{findpacket} internal activity of an agent (see figure 4). Based on the semantic duration model described above, an aspect is generated for the \text{findpacket} activity. The pointcut of the aspect refers to the location of the \text{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 (see figure 2).

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 simulation platform prototype which allows MASs to be simulated while hiding all time management infrastructure necessary for the simulation. The developer describes all timing requirements by means of semantic duration models, and does not need to alter the design of the MAS. To achieve separation of concerns, which is important for well-designed and maintainable software systems, aspect-oriented programming is used. It allows time management infrastructure necessary for the simulation to be incorporated transparently, i.e. 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 require further research and will be considered in detail in future work.

- With respect to the semantic duration models, we exclusively elaborated upon agent activities, both internal and external. However, other activities can be present in today's MASs. For example indirect communication based on digital pheromones which propagate and evaporate over time is a source of activity in MASs which gains importance and hence also requires further investigation.
- In the current model, there is no support to allow overlap of activities, as described in [12]. All activities of an individual agent happen sequentially. An important issue for future work is to extend 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 $c_{ij}$ with a duration of $r_{ij}$ and activity $c_{ik}$ with a duration of $r_{ik}$, and suppose $c_{ij}$ calls $c_{ik}$. If agent $a_i$ then executes activity $c_{ij}$, it is unclear whether agent $a_i$ has to be assigned a logical delay of $r_{ij}$ as defined earlier, or $r_{ij} + r_{ik}$ (which is currently the case in our prototype).

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

References