Towards Time Management Adaptability in Multi-agent Systems

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

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

Abstract. So far, the main focus of research on adaptability in multi-agent systems (MASs) has been on the agents' behavior, for example on developing new learning techniques and more flexible action selection mechanisms. In this paper, we introduce a different type of adaptability in MASs, called *time management adaptability*. Time management adaptability focuses on adaptability in MASs with respect to execution control. First, time management adaptability allows a MAS to be adaptive with respect to its execution platform, anticipating arbitrary and varying timing delays which can violate correctness. Second, time management adaptability allows the execution policy of a MAS to be customized at will to suit the needs of a particular application. We discuss the essential parts of time management adaptability: (1) we employ *time models* as a means to explicitly capture the execution policy derived from the application's execution requirements, (2) we classify and evaluate *time management mechanisms* which can be used to enforce time models, and (3) we introduce a *MAS execution control platform* which combines both previous parts to offer high-level execution control.

1 Introduction and Motivation

Traditionally, the scope of research on adaptability in multi-agent systems (MASs) has been focused on trying to improve adaptability with respect to the behavior of individual agents and agent aggregations. As a consequence, the progress made by improving learning techniques and developing more flexible action selection mechanisms and interaction strategies over time is remarkable. This, however, may not prevent us from opening up our perspective and investigating other issues requiring adaptability in MASs. This paper is a report on ongoing work and introduces *time management adaptability* as an important form of adaptability with respect to the execution control of MAS applications.

1.1 The Packet-World

We introduce the Packet-World application we have developed [1], since this is used as an example MAS throughout the text.

D. Kudenko et al. (Eds.): Adaptive Agents and MAS II, LNAI 3394, pp. 88–105, 2005.

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 their correspondingly colored destination. The grid contains one destination for each color. Fig.1 shows an example of a Packet-World of size 10x10 wherein 5 agents are situated. Colored squares symbolize packets and colored circles are delivery points. The colored rings symbolize pheromone trails discussed below.

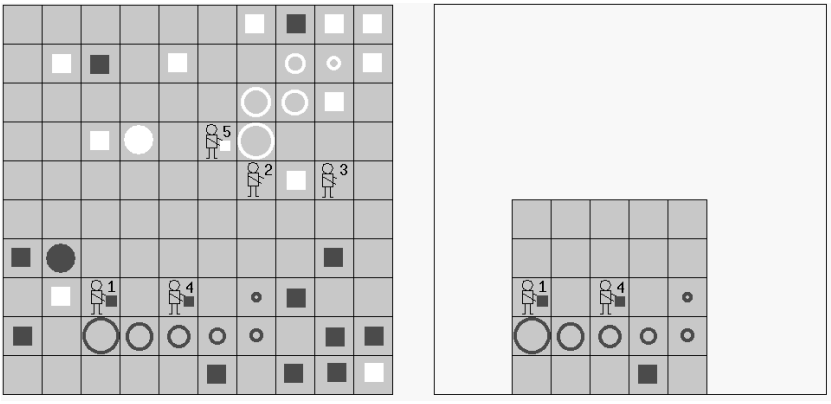

Fig. 1. The Packet-World: global screenshot (left) and view range of agent 4 (right)

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 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 agent 4 in Fig.1).

Furthermore, agents can interact with other agents too. We allow agents to communicate indirectly based on stygmergy [2, 3, 4]: agents can deposit pheromone-like objects at the field they are located on. These pheromones can be perceived by other agents. Typical characteristics of pheromones are evaporation, aggregation and diffusion. Evaporation means that the strength of pheromones diminishes over time. Aggregation on the other hand means that different pheromone deposits at the same location are combined into a single pheromone with increased strength. Finally, diffusion means that pheromones deposited at a particular location are spread to neighboring locations over time, making it more likely that agents further away can perceive them.

In the Packet-World, only pheromone evaporation and aggregation are currently supported. This allows the impact of information that is not regularly reinforced to gradually decrease over time and to disappear eventually. In Fig.1, pheromones are symbolized by colored rings. The color of the ring corresponds to the packet color. The radius of the ring is a measure for the strength of the pheromone and decreases as the pheromone evaporates. In the Packet-World, agents use stygmergy to construct pheromone trails only when they move from clusters of packets towards the corresponding destinations [5]. Other agents noticing such a pheromone trail can decide to follow it in the direction of increasing pheromone strength to get to a specific destination (e.g. when they are already carrying a packet of corresponding color). On the other hand, agents can also decide to follow the trail in the direction of decreasing pheromone strength leading to the packet cluster (e.g. when they are not carrying anything). Hence they can help transporting the clustered packets, reinforcing the evaporating pheromone trail on their way back from the packet cluster to the destination. In this way, stygmergy provides a means for coordination between the agents which goes beyond the limitations of the agents' locality in the environment.

1.2 Problem Statement

So far, time in MASs is generally dealt with in an *implicit* and *ad hoc* way: once the agents have been developed, they are typically hooked together using a particular activation regime or scheduling algorithm, without an appropriate time management (see Fig.2). MASs with an implicit notion of time are generally not adapted at all to run-time variations of timing delays introduced by the underlying execution platform, e.g. network delays, delays due to scheduling policies, etc. Moreover, variations with respect to the execution of the agents can have a severe impact on the behavior of the MAS as a whole [6, 7, 8]. The main reason for this is that the temporal relations existing in the problem domain differ significantly from the arbitrary and variable time relations in an execution platform [9]. In other words, delays in an execution platform are based on quantities which have nothing to do with the problem domain. This emphasizes the need for an *explicit* time management. To illustrate this, consider the following examples from the Packet-World application:

1. In the Packet-World, agents with simpler internal logic can react faster than agents with more complex internal logic. Consider a packet lying in between a cognitive agent and a significantly faster reactive agent, for instance the

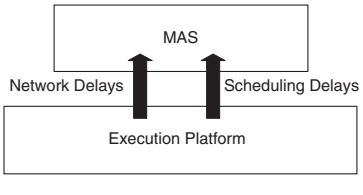

Fig. 2. MAS directly built upon an execution platform

white packet between agents 2 and 3 in Fig.1. In case both agents start reasoning at the same time, it is required that the reactive agent can pick up the packet before the cognitive one can. However, in practice, the response order of both agents is arbitrary, because the underlying execution platform could cause the cognitive agent's process to be scheduled first, allowing it to start thinking earlier and pick up the packet before the faster reactive agent even got a chance.

2. The agents can deposit pheromones in the environment to coordinate their activity. These pheromones evaporate over time. Because the effectiveness of pheromone-based communication is strongly dependent upon this evaporation rate, the latter is tuned to suit the needs of a particular application. However, fluctuations in the load of the underlying execution platform can cause agents to speed up or slow down accordingly, leading to a significant loss of pheromone effectiveness which affects the overall behavior of the MAS.
3. Problems can also arise with respect to the actions agents can perform. In the Packet-World application, the time period it takes to perform a particular action is required to be the same for all agents. However, fluctuations in processor load can introduce variations with respect to the execution time of actions. As a consequence a particular action of an agent can take longer than the same action performed by other agents. This leads to agents arbitrarily obtaining privileges compared to other agents due to the execution platform delays, a property which is undesirable in our problem domain.

The examples above show that a MAS without time management support is not adapted to varying delays introduced by the execution platform, which can be the cause of unforeseen or undesired effects. The execution of all entities within a MAS has to be controlled according to the execution requirements of the MAS, irrespective of execution platform delays. Currently the only option for the developer is to hardcode these requirements from scratch into the MAS application. However, this is a complex and error-prone task which has to be tackled by the developer without any support. In this paper, we introduce *time management adaptability* as a generic solution to this problem and a structured way to control the execution of a MAS.

1.3 Time Management Adaptability

Time management adaptability allows the execution of a MAS to be controlled according to an execution policy that can be adapted to suit the needs of a particular MAS application. An execution policy specifies the desired timing behavior for a MAS application in a platform-independent way. By enforcing a particular execution policy, even in the presence of variable execution platform delays, time management adaptability allows all temporal relations that are essential for the application to be correctly reproduced in the software system. In this way, a MAS application is adapted to varying delays introduced by the execution platform. Moreover, time management adaptability allows easy

adaptation of a MAS's timing behavior, because it deals with time in an explicit manner and introduces execution control into a MAS as a separate concern. In order to achieve this, time management adaptability consists of three main parts (see Fig.3):

1. **Time models** are necessary to explicitly model the MAS's required timing behavior, irrespective of the underlying execution platform. As such, time models capture an execution policy according to which the MAS's execution has to be controlled. Time models are explicit which allows them to be adapted to reflect the customized needs of a particular MAS application.
2. **Time management mechanisms** are a means to enforce a MAS's customized time model, even in the presence of arbitrary delays introduced by the execution platform.
3. A **MAS execution control platform** combines both time models and time management mechanisms to control the execution of a MAS. In a MAS execution control platform, time models capture the execution requirements, and time management mechanisms are employed to prevent time models from being violated during execution. In this way, time as experienced from the point of view of the MAS application can be decoupled from the timing delays of the platform on which the MAS executes.

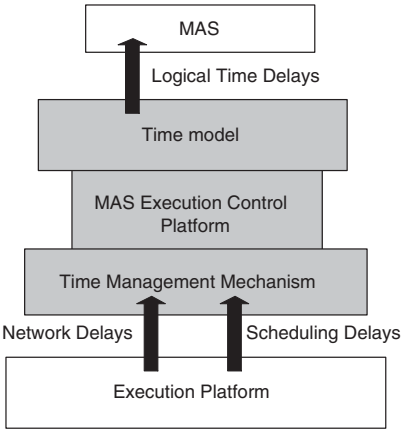

Fig. 3. Time management adaptability in MASs

Outline of the paper. We first clarify the concept of time in MASs in Sect.2. Then we discuss the various parts of time management adaptability: in Sect.3 we elaborate on *time models*. In Sect.4, we give an overview of the main *time management mechanisms* existing today, and Sect.5 discusses *MAS execution control platforms* in more detail. Finally, we look forward to future work in Sect.6 and draw conclusions in Sect.7.

2 Time in MAS

2.1 Time and Execution Requirements

MASs are characterized by a high degree of parallelism. A MAS consists of a (large) number of active, autonomous agents which coexist in a shared, active environment. A MAS does not have a global control flow, but instead each agent has its own, local flow of control [10] which allows the agent to have control over its own behavior. Because a MAS consists of several agents, the execution of all agents relative to each other has to be organized appropriately. More precisely, the execution of all agent activities within a MAS needs to be managed in a way that reflects the requirements of that particular MAS application. Because all agents run in parallel, each activity an agent can perform is related in time to the activities of other agents within the MAS. Moreover, also the environment of the MAS can contain activity, such as evaporating pheromones used for indirect communication [11]. The temporal relations between all activities (originating from the agents as well as from the environment) determine the relative timing and order in which these activities are executed. In MASs, these temporal relations between activities generally depend upon the semantic properties of the activities within the MAS application, and are independent of the time it takes to execute the corresponding code instructions. As such, the temporal relations determine execution requirements for the MAS application. These execution requirements of a MAS need to be enforced at run-time, irrespective of arbitrary delays in the execution platform.

2.2 Different Concepts of Time

One of the most common points of confusion with respect to time management is what is actually meant by time in software systems. In order to use time to express execution requirements of a MAS application, which are dependent upon semantic properties of the MAS, we need to make a clear distinction between two sorts of time [9] which are of relevance for the rest of the paper:

- **Wallclock time** is the (execution) time as measured on a physical clock while running the software system. For example, in the Packet-World the execution of a particular agent performing a move action might take 78 milliseconds on a specific processor, while a pick up packet action takes 140 milliseconds on the same processor.
- **Logical time** (also called virtual time) is a software representation of time. Logical time could for instance be represented by an integer number. From the viewpoint of the MAS application, only logical time is experienced. As such, temporal relations between activities in the MAS application can be expressed by means of logical time. For example, in the Packet-World a move action for an agent could take 2 units of logical time, while a pick up packet action semantically only takes 1 unit of logical time, and after executing the program for 3 minutes of wallclock time, 634 units of logical time may have passed in the Packet-World.

According to the way logical time advances in a system, a number of execution modes can be distinguished [9]. In a *real-time execution*, logical time advances in synchrony with wallclock time. In *as-fast-as-possible executions*, logical time is advanced as quickly as possible, without direct relationship to wallclock time. An example: suppose that in a software system 100 units of logical time are processed after 5 minutes of wallclock time. As a consequence, the total number of logical time units processed after another 5 minutes of wallclock time must be 200 in case of a real-time execution, and can be any number greater than 100 in case of an as-fast-as-possible execution.

3 Time Models

Time models are inspired by research in the distributed simulation community, where they are used implicitly to assign logical time stamps to all events occurring in the simulation [12][13]. In software simulations, the logical time stamp of an event corresponds to the physical time the event was observed in the real world which is being simulated.

However, we extend the use of time models from pure simulation contexts to execution control for MAS applications in general. Here, logical time is not used to obtain correspondence to physical time, which has no meaning outside the scope of simulation, but as a means to express the semantic time relations that reflect the execution requirements of a MAS (see Sect.2.1). Also in contrast to software simulations, time models are now explicitly represented, which has the advantages that all execution requirements are made explicit on the one hand and can be adapted on the other hand.

A time model captures the execution requirements in terms of logical time, according to the semantic properties of the MAS application. More precisely, a time model defines how the *duration* of various activities in a MAS is related to logical time [14], and these logical durations of activities are used as a means to determine the relative execution order of all activities within a MAS. In this way a time model allows the developer to describe the required execution of the MAS in a platform-independent way. The execution of a MAS application on a particular execution platform must be controlled according to the defined time model.

Because time models capture time relations that reflect the semantic properties of activities within the MAS application, we first investigate the structure of a MAS in order to identify the relevant activities that need to be time modeled.

3.1 Agent Activities

Agents within a MAS are generally able to perform several agent activities[1], and each agent can autonomously decide which activity to perform. As a conse-

[1] With the general term *agent activities*, we refer to all internal deliberations, as well as all actions in the environment and all perceptions of the environment, insofar they (1) can be performed by an agent and (2) are considered semantically relevant.

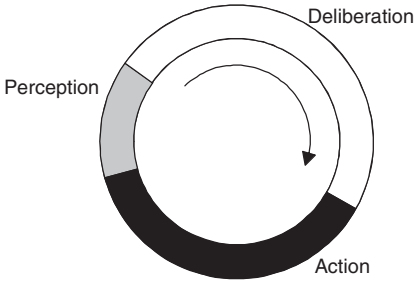

Fig. 4. A typical agent control flow cycle

quence, time modeling requires assigning logical durations to all of the agent's activities. Each time an agent decides to perform a particular activity, the agent's logical clock is advanced with a logical time period equal to the duration of that activity. For our discussion, we assume that an agent has a control flow cycle as the one depicted in Fig.4.

Agent Deliberation. A first important activity an agent can perform, is internal deliberation. The purpose of this deliberation is determining the next action the agent is going to perform. As such, agent deliberation is an activity that does not cross the agent's boundaries. Depending on the context, agent's deliberation can be very simple (e.g. stimulus-response behavior in reactive agents) or immensely complex (e.g. sophisticated learning and planning algorithms used in cognitive agents).

In the context of agent-based simulation, modeling the duration of an agent's deliberation activity has received a lot of interest. We describe various models which have been proposed to model how much logical time the deliberation of the agents takes, and discuss their relevance for specifying execution requirements.

- A constant time model for the agents' deliberation [15] implies that the deliberation of all agents is performed in a constant logical time, irrespective of the actual wallclock time that is needed to execute the deliberation. As a consequence, constant time models are independent of the underlying execution platform, which makes them suitable for specifying execution requirements. The advantage of a constant deliberation time model is the fact that it is simple and easy to understand and use. By assigning constant logical time durations to the deliberation activity of each agent, one can for instance specify the relative reaction speed of all agents within a MAS, irrespective of their implementation or execution efficiency. However, constant time models suffer from a lack of expressiveness since the agent's deliberation is considered as a single, course-grained, and black box activity.
- The deliberation activity of an agent can also be time modeled in a more fine-grained way. The time model has to take into account what the agent is actually deliberating about. In this case, a number of *deliberation primitives* [16] that are semantically relevant are distinguished within the deliberation activity of an agent, e.g. evaluating a board position for a chess playing

agent, or calculating the shortest path to reach a particular destination, etc. A logical duration is assigned to each of the deliberation primitives, and only the deliberation primitives the agent actually uses are taken into account to determine the logical duration of its deliberation activity. This kind of time model is also suitable for specifying execution requirements since it is independent of the actual implementation or underlying execution platform.
- A popular approach counts the actual computer instructions as a basis for determining the logical deliberation time [16]. Modeling the logical deliberation time of an agent as a function of the number of code instructions executed during deliberation is then considered as an extreme example of the previous approach in which each computer instruction is treated as a deliberation primitive. However counting computer instructions is dependent on the programming language of the underlying execution platform. As a consequence, this way of time modeling cannot be used for specifying execution requirements, since it is not platform-independent. Moreover, changes in the implementation and the presence of GUI or debug code can have a significant influence on the logical deliberation time, although these issues have no semantic value.
- The logical deliberation time can be modeled as a function of the wallclock time used for executing the deliberation [17]. This approach is also not feasible for specifying execution requirements, since the logical duration is now susceptible to the load and performance of the underlying computer system.

Agent Action. Another important activity of an agent is performing actions (see Fig.4). In contrast to agent deliberation, agent actions are activities that cross the agent's boundaries: actions typically change the state of the environment. Compared to agent deliberation, time modeling agents' actions in the environment has received little interest. However, in the context of defining execution requirements, imposing time models on the actions agents perform is indispensable. Depending on the semantic context, actions can be assigned a period of logical time. This period can be considered as the time it takes until the effects of the action are noticeable in the environment. In our initial time model, we assume that the agent is not allowed to perform anything else until the action has completed. However, we are currently investigating an extension which allows overlap of the activities of one agent. This could for example allow us to model an agent that can think while it is moving, something that is not possible in the current time model.

Agent Perception. Another important agent activity is perception, which allows an agent to observe its environment. Agent perception is also an activity that crosses the agent's boundaries, but it differs from agent actions as perception does not alter the state of the environment. Time modeling agent perception is often neglected, but its duration could be significant. Assigning a logical duration to agent's perceptions is analogous to time modeling the agent's actions, and allows a developer to specify the time it takes for an agent to perceive its neighborhood.

3.2 Ongoing Activities

Besides activities originating from the agents, there can be other activities within a MAS which require time modeling. In the MAS research community, there is an increased environmental awareness. The environment itself is often dynamic and evolves over time [11]. As a consequence the environment itself can contain a number of *ongoing activities* [18] which are essential for the correct working of the MAS as a whole. An example of ongoing activities in the Packet-World application are the pheromone trails which evaporate continuously. Ongoing activities are characterized by a state which evolves over time, even without agents affecting it. Agents can often initiate ongoing activities and influence their evolution. As a consequence the environment is not passive, but active, and responsible for managing all ongoing activities.

In many MAS applications however, the dynamics of ongoing activities are dealt with in an ad hoc way. Typically the evaporation rate of pheromones is modeled in wallclock time and the correlation between agent activity on the one hand and pheromone activity on the other hand is not specified. As a consequence optimal coordination efficiency can hardly be maintained: varying loads on the execution platform cause agent activity to slow down or speed up accordingly. Because pheromone evaporation is determined upon wallclock time, it does not adapt itself to the changes in the execution speed of the agents, and coordination efficiency drops. Therefore, to guarantee the correct execution behavior, it is essential to specify the logical duration of ongoing activities in the environment, because this allows us to relate their execution to other activities within a MAS application.

3.3 Execution Requirements for the Packet-World

We now return to the Packet-World application, and illustrate the use of time models to capture the execution requirements that are necessary for the correct working of the MAS.

In the Packet-World, the following activities can be distinguished for each agent: the deliberation activity of deciding upon its next action, the perception activity of perceiving its neighborhood, and a number of activities corresponding to the actions an agent can perform in the environment: move to a neighboring field, pick up a packet, put down a packet and drop pheromone. Stated formally:

$E = \{think, look, move, pick, put, drop\}$
with *think*: deliberate upon next action
look: perceive neighborhood
move: move to neighboring field
pick: pick up packet
put: put down packet
drop: drop pheromone
$S = \{move, pick, put, drop\} \subset E$
$P = \{look\} \subset E$
$D = \{think\} \subset E$

E is the set of all possible activities of an agent, S the set of agent actions, P the set of agent perception activities, D the set of agent deliberation activities, and S, P and D are subsets of E.

We further distinguish between two types of agents in the Packet-World: reactive agents and cognitive agents. Each agent is either reactive or cognitive. Stated formally:

$$A^R = \{a_1^r, a_2^r, ..., a_n^r\}$$

$$A^C = \{a_1^c, a_2^c, ..., a_m^c\}$$

$$A = A^R \cup A^C = \{a_1, a_2, ..., a_{m+n}\}$$

A^R is the set of all reactive agents in the Packet-World application, A^C the set of all cognitive agents, and A the set of all agents, reactive and cognitive.

The problem statement (see Sect.1.2) mentions a number of typical problems which arise in the Packet-World application. We now elaborate on these problems to derive execution requirements in terms of logical time models.

Action Requirements. We take a closer look at the third problem mentioned in Sect.1.2. In the Packet-World application it was observed that the underlying execution platform can have an arbitrary influence on the time it takes to perform an action. However, in the problem domain it is required the same amount of time is needed for all agents to perform a particular action. Stated formally:

$$\begin{aligned}
&\forall a_i \in A; move, pick, put, drop \in S : \\
&\quad \Delta T_{move}(a_i) = cst_{move} \\
&\quad \Delta T_{pick}(a_i) = cst_{pick} \\
&\quad \Delta T_{put}(a_i) = cst_{put} \\
&\quad \Delta T_{drop}(a_i) = cst_{drop} \\
&\quad cst_{move}, cst_{pick}, cst_{put}, cst_{drop} \in I\!N
\end{aligned}$$

A is the set of all agents, S the set of all agent actions, $\Delta T_s(a_i)$ the logical duration of action $s \in S$ performed by agent $a_i \in A$, and $I\!N$ the set of natural numbers.

Since perception is considered as a kind of action in our application, we obtain the following expression:

$$\begin{aligned}
&\forall a_i \in A; look \in P : \\
&\quad \Delta T_{look}(a_i) = cst_{look} \\
&\quad cst_{look} \in I\!N
\end{aligned}$$

A is the set of all agents, P the set of all agent perception activities, $\Delta T_p(a_i)$ the logical duration of perception activity $p \in P$ performed by agent $a_i \in A$, and $I\!N$ the set of natural numbers.

Both expressions above can be combined in a more compact notation:

$$\forall a_i \in A; \forall e \in S \cup P :$$
$$\Delta T_e(a_i) = cst_e$$
$$cst_e \in I\!N$$

Deliberation Requirements. We now return to the first example of the Packet-World application (see Sect.1.2). The problem was the fact that the underlying execution platform can influence the reaction speed of the agents, leading to a response order which is arbitrary. However, this is not desired in the Packet-World, where it is required that a reactive agent always reacts faster than a cognitive agent, in case both start deliberating at the same moment in time. Based on the agent's control flow cycle as depicted in Fig.4, the duration in logical time it takes an agent to complete a control flow cycle can be stated formally as:

$$\forall a_i \in A; \forall s \in S; \forall p \in P; \forall d \in D :$$
$$\Delta T^{cycle}(p, d, s, a_i) = \Delta T_p(a_i) + \Delta T_d(a_i) + \Delta T_s(a_i)$$

$\Delta T^{cycle}(p, d, s, a_i)$ is the duration in logical time it takes agent $a_i \in A$ to complete a control flow cycle consisting of perception activity $p \in P$, followed by deliberation activity $d \in D$ and by action $s \in S$. $\Delta T_p(a_i)$ is the logical duration of perception activity $p \in P$ performed by agent $a_i \in A$, $\Delta T_d(a_i)$ the logical duration of deliberation activity $d \in D$ performed by agent $a_i \in A$ and $\Delta T_s(a_i)$ the logical duration of action $s \in S$ performed by agent $a_i \in A$.

In the Packet-World, $P = \{look\}$ and $D = \{think\}$ are singletons. As a consequence, the moment in logical time an agent completes an action can be defined as:

$$\forall a_i \in A; \forall s \in S :$$
$$T_{end}(s, a_i) = T_0 + \Delta T^{cycle}(look, think, s, a_i)$$

$T_{end}(s, a_i)$ is the moment in logical time agent a_i completes action s. T_0 is the moment in logical time that a new cycle in the control flow of agent a_i starts (which corresponds to the moment in logical time the previous action of agent a_i was completed).

The requirement that a reactive agent can always pick up the packet before a cognitive agent in case both start deliberating at the same moment time, is hence formalized as follows:

$$\forall a_i^r \in A^R; \forall a_j^c \in A^C; pick \in S :$$
$$T_{end}(pick, a_i^r) < T_{end}(pick, a_j^c)$$

A^R is the set of reactive agents, A^C the set of cognitive agents, and S the set of all agent actions in the environment.

By substitution we obtain:

$$T_0 + \Delta T^{cycle}(look, think, pick, a_i^r) < T_0 + \Delta T^{cycle}(look, think, pick, a_i^c)$$

or

$$T_0 + cst_{look} + \Delta T_{think}(a_i^r) + cst_{pick} < T_0 + cst_{look} + \Delta T_{think}(a_j^c) + cst_{pick}$$

Simplifying both sides of the equation gives us:

$$\Delta T_{think}(a_i^r) < \Delta T_{think}(a_j^c)$$

With $a_i^r \in A^R$ a reactive agent, and $a_j^c \in A^C$ a cognitive agent.

Hence to assure that the reactive agent always acts faster than the cognitive one, the logical duration of the agents' deliberation needs to be modeled such that the reactive agent's deliberation duration is smaller than the cognitive agent's deliberation time. This also corresponds to our intuition.

Pheromone Requirements. Finally, we take a closer look at the second problem of Sect.1.2. The load on the underlying execution platform causes the agents' execution speed to change accordingly. However, if we want to maintain pheromone effectiveness, we need a continuous adaptation of the pheromone evaporation rate to the agents' execution speed. Hence we want the duration of pheromone activity to be semantically related to the duration of agent activities. As a first step, we use a simple model for pheromone activity. For the pheromone evaporation rate in the Packet-World application we can state more formally:

$$\Delta T_{evap} = cst_{evap}$$
$$cst_{evap} \in I\!N$$

with ΔT_{evap} the logical duration it takes for a pheromone to evaporate until only half of its initial strength is remaining. Agent activity is related to logical time. Relating pheromone activity to the same logical clock, instead of the wallclock, allows the dynamics of both agents and pheromones to be coupled.

3.4 A Time Model for the Packet World

In Sect.3.3 we formulated a number of execution requirements which have to be met to allow the execution of our MAS to evolve according to the semantic properties of all activities within the Packet-World application. To formulate a simple time model, specific values to the various activities have to be assigned, expressing the logical durations. These values are expressed in *logical time units* (LTU). The execution requirements derived above give rise to an array of constraints which all have to be satisfied in a time model for the application.

An example time model which satisfies all requirements of the Packet-World application is given:

$$\forall a_i \in A :$$
$$\Delta T_{move}(a_i) = 3\ LTU$$
$$\Delta T_{pick}(a_i) = 2\ LTU$$
$$\Delta T_{put}(a_i) = 2\ LTU$$
$$\Delta T_{drop}(a_i) = 1\ LTU$$
$$\Delta T_{look}(a_i) = 1\ LTU$$

$\forall a_i^r \in A^R$:
$\Delta T_{think}(a_i^r) = 4\ LTU$

$\forall a_j^c \in A^C$:
$\Delta T_{think}(a_j^c) = 12\ LTU$

$\Delta T_{evap} = 100\ LTU$

Note that this time model is only one example which satisfies all requirements of the Packet-World application. Alternative time models that also satisfy the requirements can be modeled. As such, there exists a degree of freedom for the developer, enabling her/him to further tune and refine the working of the application within the boundaries defined by the execution requirements.

4 Time Management Mechanisms

By specifying time models, the developer can define execution requirements for a MAS. By relating MAS activities to logical time, time models impose an order on all MAS activities, dictated by logical time. However, as illustrated in the introduction, the temporal characteristics of the execution platform are not necessarily the same as those described in the logical time model. As a consequence, we additionally need *time management mechanisms* to ensure that all activities are executed according to the time model specification. These mechanisms avoid that any event with a logical time in the future can have influence on things with a logical time in the past, even in the presence of arbitrary network delays or computer loads. In other words, time management mechanisms preserve causality dictated by logical time.

Distributed simulation communities have been investigating the consistency of logical time in simulations for a long time. All events happening are ordered and hence causally related by means of the global notion of logical time. Therefore various time management mechanisms have been developed to prevent causality errors:

- **Execution directed by clock**. In this approach the logical time of the system is discretized in a number of intervals of equal size. The interval size is called *time-step*. Global synchronization schemes force all entities to advance together in a lock-step mode, and hence the execution of the system proceeds synchronously. In the case of MASs, a drawback is that synchronous execution forces all agents to act at the pace of the slowest one, which severely limits execution speed [19][20]. Moreover, since a central authority must control and keep track of the execution of all agents in the system, the cost of synchronous approaches increases rapidly as the number of agents grows.
- **Execution directed by events**. In this case, events are generated by all entities [12], and each event has a precise logical time stamp which allows sorting them. During execution, the next event to be processed is the one

with the smallest logical timestamp, ensuring causality and thereby skipping periods of inactivity. However in a distributed context (distributed discrete event simulation [13]), a system is modeled as a group of communicating entities, referred to as logical processes (or LPs). Each LP contains its own logical clock (indicating its local logical time) and all LPs process events asynchronously and advance at different rates, which allow a significant speedup, but may cause causality errors. Hence, for asynchronous execution additional synchronization is needed to ensure that each LP processes messages in increasing logical time order:

- **Conservative synchronization**. In conservative synchronization [21] each LP only processes events when it can guarantee that no causality errors (out of (logical) time order messages) will occur. This causes some LPs to block, possibly leading to deadlock. The performance of conservative synchronization techniques relies heavily on the concept of lookahead, but the autonomous, proactive behavior of agents could severely restrict the ability to predict events [22]. Moreover, to determine whether it is safe for an agent to process an event, information about all other agents must be taken into account, limiting the scalability of this approach.
- **Optimistic synchronization**. In optimistic approaches, causality errors are allowed, but some roll-back mechanism to recover from causality violations is defined (e.g. time warp [23]). While this approach is feasible for simulations, providing roll-back for MAS applications in general (outside the scope of simulation) is not feasible at all. Moreover, the cost imposed by the roll-back mechanisms can easily outweigh the benefits [22], and increases rapidly as the number of agents grows.

5 MAS Execution Control Platform

To control the execution of a MAS in an appropriate way, a *MAS execution control platform* must provide support for both explicit time models on the one hand and time management mechanisms on the other hand.

First, logical time models are needed as a means for the developer to explicitly express the execution policy for all MAS activities. However, the execution policy expressed in the time model has to be enforced in the MAS application, irrespective of delays in the underlying execution platform. For this reason, time management mechanisms are needed. They prevent time models from being violated, and ensure the execution of the MAS behaves according to the execution policy which is described.

We refer to our previous work [24] for a more technical description of the structure of a MAS execution control platform, and limit our discussion to its most important characteristics:

– **Higher level of abstraction**. The execution policy is described by means of an explicit model. This model focuses on *what* the execution policy of a

particular MAS application is, while hiding the developer from *how* this execution policy is enforced. The model provides a higher level of abstraction to MAS developers: from a developer's point of view, only the logical durations described in the time model apply and determine the execution. As a consequence, abstraction can be made of the unreliable execution platform delays. By means of a time model as execution policy description, all activities within the MAS can be identified and can be assigned logical durations.
- **Separation of concerns**. A MAS's functionality can be developed without taking into account the execution policy. Controlling the execution of a MAS is considered as a separate concern. This relieves the developer from tackling execution control from scratch and from hard-coding it into the agents' functional behavior. Based on an explicit execution policy description provided by the developer, all time management mechanisms are integrated automatically into the MAS's functionality using aspect-oriented programming, without requiring the developer to change the design of the MAS. In this way, the complexity of time management mechanisms that enforce the execution policy can be hidden from the developer.
- **Independence with respect to the execution platform**. By combining a logical time model and a time management mechanism to enforce it, a MAS can be developed without taking into account the specific timing characteristics of the execution platform. A MAS application is subjected to logical delays defined in the time model, and execution platform delays can no longer introduce unforeseen effects. This results in MASs being unconstrained with respect to the timing characteristics of their execution platforms.
- **Adaptability of the execution policy**. The explicit representation of the execution requirements of a MAS in a time model has the advantage that execution requirements can be adapted. This enables fine-tuning of the existing execution policy and allows the integration of new execution requirements.

6 Future Work

This paper reports on ongoing work investigating a generic and structured way to deal with execution control in the context of MASs. The approach was described in general, and a lot of work still needs to be done on various issues described in this paper:

- We are currently working to improve the formalism for describing logical time models, to come to an approach which is generally applicable and more theoretically founded. It should for instance be possible to specify potential overlap of activities (see Sect.1). Also the dynamics of ongoing activities in the environment, such as pheromone evaporation, still needs to be investigated more in depth.
- As shown in Sect.4, mechanisms enforcing time models and ensuring global causality are limited in scalability, making these approaches inefficient for use in a large-scale distributed MASs. A possible alternative presumes we

abandon the notion of a global (logical) clock to determine causal relationships. Hence not all parts of the MAS are related in time, and there is only a locally shared notion of time. This follows from the observation that agents within a MAS typically perceive and act locally. Based on this, it makes sense only to ensure temporal relationships between agents residing in each other's neighborhood, without modeling causality between agents far away from each other, at the benefit of increased scalability. Regional synchronization [20] provides a flexible mechanism to dynamically detect clusters of agents, based on the overlap of so called *spheres of influence*. Within such clusters of agents, the mechanisms discussed in Sect.4 could be applied locally, hence avoiding scalability limitations at the cost of a loss of a global notion of logical time.

7 Conclusion

In this paper, we emphasized time management adaptability as a type of adaptability which has significant importance, although this type of adaptability is not often considered in the context of adaptive MASs. Time management adaptability allows the execution requirements for a MAS application to be described explicitly and enforced transparently, irrespective of the unpredictable execution platform delays. A *time model* is employed to explicitly capture the execution policy and relate all MAS activities to logical time. *Time management mechanisms* form a second important part of time management adaptability: they are needed to enforce time models. Time models and time management mechanisms are combined in *MAS execution control platforms*.

The advantage of time management adaptability it twofold. First, time management adaptability allows a MAS to be independent of the underlying execution platform: the combination of time models and time management mechanisms prevents the MAS's behavior from being affected by timing issues introduced by the execution platform. More precisely, the relative execution order of all MAS activities remains invariant under various execution conditions. Second, time management adaptability allows the execution policy to be adapted to suit the needs of the MAS application, providing a higher level of abstraction to organize the execution of a MAS, and a means to introduce execution control as a separate concern.

References

1. 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. (2002)
2. Sauter, J.A., Matthews, R., Dyke, H.V.: Evolving adaptive pheromone path planning mechanisms. Autonomous Agents and Multiagent Systems, AAMAS 2002 (2002)
3. Parunak, H.V.D., Brueckner, S.: Ant-like missionaries and cannibals: Synthetic pheromones for distributed motion control. In Sierra, C., Gini, M., Rosenschein, J.S., eds.: Proceedings of the Fourth International Conference on Autonomous Agents, Barcelona, Catalonia, Spain, ACM Press (2000) 467–474

4. Brueckner, S.A.: Return From The Ant – Synthetic Ecosystems For Manufacturing Control. PhD thesis, Humboldt University Berlin, Department of Computer Science (2000)
5. Steels, L.: Cooperation between distributed agents through self-organization. Decentralized A.I. (1990)
6. Axtell, R.: Effects of interaction topology and activation regime in several multi-agent systems. Lecture Notes in Computer Science **1979** (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. Fujimoto, R.: Time management in the high level architecture. Simulation, Special Issue on High Level Architecture **71** (1998) 388–400
10. Wooldridge, M.J.: Multi-agent systems : an introduction. Wiley, Chichester (2001)
11. Parunak, H.V.D., Brueckner, S., Sauter, J., Matthews, R.S.: Distinguishing environmental and agent dynamics: A case study in abstraction and alternate modeling technologies. Lecture Notes in Computer Science **1972** (2001)
12. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM **21** (1978) 558–565
13. Misra, J.: Distributed discrete-event simulation. Computing Surveys **18** (1986) 39–65
14. Anderson, S.D., Cohen, P.R.: Timed common lisp: the duration of deliberation. SIGART Bull. **7** (1996) 11–15
15. Uhrmacher, A., Kullick, B.: Plug and test software agents in virtual environments. In: Winter Simulation Conference, WSC'2000. (2000)
16. Anderson, S.D.: Simulation of multiple time-pressured agents. In: Winter Simulation Conference, WSC'97. (1997) 397–404
17. Anderson, S.: A Simulation Substrate for Real-Time Planning. PhD thesis, University of Massachusetts at Amherst (1995)
18. Weyns, D., Holvoet, T.: Formal model for situated multi-agent systems. Formal Approaches for Multi-agent Systems, Special Issue of Fundamenta Informaticae (2004)
19. Ferber, J., Muller, J.: Influences and reaction: a model for situated multiagent systems. In: Proceedings of the 2th International Conference on Multi-Agent Systems, AAAI Press (1996)
20. Weyns, D., Holvoet, T.: Regional synchronization for simultaneous actions in situated multiagent systems. Multi-Agent Systems and Applications III, Lecture Notes in Computer Science **LNAI 2691** (2003) 497–511
21. Chandy, K.M., Misra, J.: Asynchronous distributed simulation via a sequence of parallel computations. Communications of the ACM **24** (1981) 198–205
22. Uhrmacher, A., Gugler, K.: Distributed, parallel simulation of multiple, deliberative agents. In: Proceedings of the 14th Workshop on Parallel and Distributed Simulation, PADS'2000, Bologna, Italy, IEEE (2000) 101–110
23. Jefferson, D., Sowizral, H.: Fast concurrent simulation using the time warp mechanism. In: Proceedings of the SCS Multiconference on Distributed Simulation. (1985) 63–69
24. Helleboogh, A., Holvoet, T., Weyns, D.: Time management support for simulating multi-agent systems. In: Joint Workshop on Multi-Agent and Multi-Agent Based Simulation, MAMABS. (2004)