ANALYSIS OF TEMPORAL DEPENDENCIES OF PERCEPTIONS AND INFLUENCES FOR THE DISTRIBUTED EXECUTION OF AGENT-ORIENTED SIMULATIONS

Nicolas Sébastien*
Rémy Courdier*
Didier Hoareau*
Marc-Philippe Huget**

*EA2525 LIM-IREMIA
2 rue Joesph Wetzel
97490 Sainte-Clotilde
France
nicolas.sebastien@univ-reunion.fr
remy.courdier@univ-reunion.fr
didier.hoareau@univ-reunion.fr

**LISTIC
5 chemin de Bellevue
BP 80439
74944 Annecy-Le-Vieux cedex
France
marc-philippe.huget@univ-savoie.fr

KEYWORDS
Multiagent simulation, Temporality model, distributed parallel execution

ABSTRACT

Agent-Oriented Simulations (AOS) are helpful to understand real complex systems particularly via their modeling ability and the observations they provide. To fully exploit these simulations, AOS platforms should offer shorter simulation time even if simulated systems are more and more complex.

In this paper, we propose a method to minimize the execution time of AOS through the parallel execution of simulation agents. Our solution is ensured by a network of AOS platforms that relies on the identification of agents’ temporal dependencies.

Our proposal is based on the Temporality model that allows the agents to express their temporal dynamics.

INTRODUCTION

Agent-Oriented Simulation (AOS) is an appealing approach to model complex systems. Reproducing the real system complexity, AOS provides multi-scale observation of a model (Ralambodrainy and Courdier 2007) in various configurations. This approach faces a scalability problem with ever more complex simulations implying throng of interacting agents which increase simulation time.

Grid technologies (Buyya and Venugopal 2005, Foster 2002) tackle such scalability problems. However, distributed simulation requires a specific scheduler to maintain time causality between events generated on different computing devices. Based on a temporal model (commonly constant step, event-driven or mixed approaches), the scheduler impact the overall simulation time.

We recently developed a temporal model named the Temporality model. This one supports various classical temporal models like constant step or event-driven approaches, by adding informations to agents’ activation timestamps. In this paper, we show how to exploit these informations to propose a scheduler based on the Temporality model that provides parallel execution of agent-oriented simulations.

This paper is structured as follows. We first define our scheduling algorithm based on the Temporality model before detailing how to detect causality links established in the simulated model. We show how to provide parallel execution of simulation given the causality links before we conclude this paper.

SCHEDULING AND PARALLELISM

The Temporality model

The Temporality model is defined as a temporal model for Agent-Oriented Simulations. One of its major assets is the fact that it eases the identification in time of the different dynamics agents participate in. Indeed, when using the Temporality model, agents define their own activation times. The scheduler is so driven by the needs of the agents composing the simulated model. Such needs are described by a data structure called temporality and defined by $t = id, d, f, p, v$ with:

- $id$: temporality identifier. Such identifier can be used by agents to associate a specific behavior with the temporality.
• \([d, f]\): time interval during which the temporality can be triggered

• \(p\): the time interval between two executions of this temporality \((p = 0\) if the action is executed once).

• \(v\): the variability that defines the temporal precision for the triggering of a temporality.

A temporality trigger the agent’s behavior at each date \(x\) verifying: \(x = d + p \cdot k\), with \(k\) an integer such as \(0 \leq k \leq n\) and \(n\) the biggest integer verifying \((d + p \cdot n) = f\). Agents first give the scheduler their temporalities at simulation initialization. Then, during simulation, agents are free to modify their temporalities as part of their behavior: this enables them to adapt their own dynamic to the situation.

Considering the temporal engine, we introduce two additional data structures:

• Time Slot: a point on the simulated temporal axis on which the simulator will have to trigger the execution of an agent’s behavior.

• Tempo: structure that wraps temporalities that have the same period and are “situated” on the same temporal slot. The period characterizes the tempo as all the temporalities it contains will have the same rhythm.

Temporality model scheduling

At simulation start, we consider that each is agent is executed on a node and will remain on that node during simulation. We also consider that the scheduler is centralized; this latter makes the simulated time progressing by jumping from one time slot to an other. At each time slot, each tempo is executed by triggering all the temporalities it is composed of. When all simulation nodes have finished to execute a tempo, the scheduler determines the new time slot to attach the considered tempo to by considering its period. Thus, it only requires one operation to determine the next triggering date of each temporality composing the tempo.

To avoid inconsistency, environment remains unchanged during a time slot execution. Execution nodes must then diffuse environment updates on the network before processing the next time slot. After this synchronization, the scheduler trigger execution of the next time slot and so on until simulation end.

As agents declare their temporalities at the beginning of the simulation, it is possible to obtain a first draft of the simulated temporal axis. The Figure 1 shows a simulated temporal axis followed by the scheduler for the distributed execution: agents situated on different simulation nodes have different temporalities colors.

Figure 1: Simulated temporal axis of the Temporality model

parallel scheduling

We aim at proposing a scheduler that supports parallel and distributed execution of AOS based on the Temporality model. Our main concern is to minimize the global simulation time. We use the temporal informations to define a scheduler which does not depend on the application’s semantic.

Such a scheduler enables the triggering of tempos at the most early real time \((i.e.\) the time we live in) while ensuring that this early execution will not introduce inconsistency in the simulated model. For example, considering a resources producer \(P\) and a consumer \(C\) in a model which states the production before the consumption, an inconsistency would be having \(C\) executed before \(P\) produces the resources.

Defining a scheduler which ensures both performances in simulation and coherence of the distributed model is an issue referring to the causality principle in distributed systems. The causality principle states that if a phenomenon (called cause) produces an other phenomenon (called effect) then the effect can not occur before the cause. Given this definition, the execution of an agent at the most early real time can be expressed by:

“An agent can be activated by the scheduler as soon as this one can ensure that there is no causality relation between this agent and other agents that have not already been activated and that precede it on the simulated temporal axis.”

In the field of distributed simulation, there are two general approaches that maintain the causality between events (Ferscha and Tripathi 1994). The conservative approach is based on the identification of causality links to avoid a violation of this principle. The optimistic approach allows a causality violation but uses different mechanisms to recover such errors: it restores a previous state to take into account the event that caused the violation.

In this paper, we propose a scheduler based on the conservative approach. This choice leads us to identify the causality links that exist between agents.
IDENTIFICATION OF CAUSALITY LINKS USING THE TEMPORALITY MODEL

Definition of a causality link based on our simulation model

In our model, agents interact through the environment. They perceive their environment and influence it in order to produce an alteration in it (Ferber 1999). An agent \( a \) is influenced by an agent \( b \) if the perception made by \( a \) differs whether \( b \) has been activated before or not. A difference in a perceptions means that \( b \) dropped an influence that modified the perceived world of \( a \). The perception/influence mechanisms establish causality links between agents. As perceptions and influences are made during agents activation time (triggered by temporalities), identifying agents causality links consists in identifying causality links between temporalities.

Causality links research

Causality links between temporalities come from two facts:

i Both two considered temporalities are defined by the same agent. Behaviors associated to these temporalities uses the same agent’s data.

ii The agent activated by the temporality which is most advanced on the simulated temporal axis (we will call it later temporality) perceives a part of the environment that have been influenced by an agent activated by a previous temporality (the earlier one).

The fact (i) denotes a causality link that exists between two temporalities of an agent. The behavior associated with the earlier temporality can update the agent internal data : behavior associated with the later temporality is so based on the potentially updated data. As we cannot easily determine the impact of the execution of a temporality on an agent internal data, we extend the fact (i) by considering that a causality link exists between two temporalities if these two temporalities belong to the same agent.

The fact (ii) establishes causality links between temporalities through the environment. An agent, during the execution of a temporality \( t \), can produce an influence that would lead to a modification of the environment. All the temporalities during which an agent behavior would be affected by this influence present a causality link with \( t \).

To identify such causality links, it would require to know about the scope of each influence and perception of each agent. Such information is a priori hard to determine if the users are not asked for it during the conception phase. This is why we propose to enlarge the conditions of causality links detection induced by the fact (ii) in considering the multi-environment approach (Payet et al. 2006).

The multi-environment approach consists in describing the world agents evolve in by a set of environments, each environment having its own evolution laws. An agent can perceive and influence several environments during a temporality. With this approach, we will make considerations on environments on which perceptions and influences are made: we consider that each request can impact the whole environment it is made on. By using the multi-environment approach, we can replace the fact (ii) with:

“There exists a causality link between two temporalities if the agent activated by the later temporality makes a perception on an environment that have been influenced by the agent activated by the earlier temporality.”

Formalization

Using the Temporality model and the multi-environment approach, we consider two facts to identify causality links:

i Both the two temporalities have been defined by the same agent.

ii The agent activated by the later temporality does a perception on an environment that have been influenced by the agent activated by the earlier temporality.”

Causality links induced by the fact (i) are trivial to identify.

The fact (ii) requires information about agents’ behavior: we have to know the environments that will potentially be perceived and the environments that will potentially be influenced during the temporality’s activation.

We define \( E = \{E_1, E_2, ..., E_n\} \) the set of environments described in the simulated model, we call:

- \( E_R(a,t) \) the set of environments that can be perceived by agent \( a \) at the temporality \( t \) (\( R \) stands for read).
- \( E_W(a,t) \) the set of environments that can be influenced by agent \( a \) at the temporality \( t \) (\( W \) stands for written).

By using the previous notations, we propose an extension of the description of temporalities \( t = \{id, d, f, p, v, E_R(a,t), E_W(a,t)\} \).

We call \( t_{sl} \) the simulated timestamp value of the current activation of temporality \( ti \). We define \( t_{sg} \) the global simulated timestamp of the system. At a given real time, \( t_{sg} \) is the maximal simulated timestamp value for which all temporalities \( ti \), with \( t_{sl} \leq t_{sg} \), have been executed.
Let $A$ the set of simulated model’s agents, a temporality can be executed according to the fact (ii) when $t_{sg}$ is such as the following relation is verified:

$$E_R(a, t_k) \cap \bigcup_{(k \in A, t_j \in \{t_k, t_{sg} < t_{ak} < t_{s1}\})} E_W(k, t_j) = \emptyset$$

We underline that the set $\{t_k, t_{sg} < t_{ak} < t_{s1}\}$ is the set of temporalities that have not been yet executed and having a triggering time lower that the one of the considered temporality. As both the Temporality model and the multi-environment approach aim at uncoupling real system’s dynamics, an agent will interact with few environments during a temporality. This tend to minimize the dynamics, an agent will interact with few environments.

Large-scale simulation implies throng of interacting agents and so a great number of temporalities. As considering temporalities would then require too much resources, we transpose our work on tempos. We so define, for each tempo $T_k$:

- $E_R(T) = \bigcup_{t_k \in T} E_R(a, t_k)$ the set of environments that can be perceived during the execution of $T$.
- $E_W(T) = \bigcup_{t_k \in T} E_W(a, t_k)$ the set of environments that can be influenced during the execution of $T$.

With $TE(a)$ the set of temporalities that have been defined by the agent $a$.

**SCHEDULER’S OPERATION**

Large-scale simulation implies throng of interacting agents and so a great number of temporalities. As considering temporalities would then require too much resources, we transpose our work on tempos. We so define, for each tempo $T_k$:

- $E_R(T) = \bigcup_{t_k \in T} E_R(a, t_k)$ the set of environments that can be perceived during the execution of $T$.
- $E_W(T) = \bigcup_{t_k \in T} E_W(a, t_k)$ the set of environments that can be influenced during the execution of $T$.

With such sets, the fact (ii) consists in waiting for the $t_{sg}$ that verifies:

$$E_R(T_i) \cap \bigcup_{(T_j \in \{T_k, t_{sg} < t_{ak} < t_{s1}\})} E_W(T_j) = \emptyset$$

Providing agents do not modify their temporalities, it is possible to determine a global simulation period, called $P$: it is the least common multiple of each temporality’s period. We can then establish a dependency table between tempos on a global simulation period. During simulation, a modification of an agent’s temporalities will trigger a revision of the dependency table.

**Dependency table**

The dependency table is established considering the tuples $(E_R(T)|E_W(T))$ of each tempo. We propose the following algorithm:

for each tempo $T_k$ do
  for each perceived environment $E_l$ during $T_k$ execution do
    identify tempos $T_i$ such as $\text{ts}i \in [\text{tsk}-P, \text{tsk}]$ and during which $E_l$ is influenced.
  end for
end for

This algorithm is used to determine causality links between tempos through the environment. The dependency table is then completed by integrating the causality links between tempos containing temporalities belonging to the same agents.

We consider the temporal axis of figure 2 on which we did not represent the tempos but only indicated the tuples $(E_R(T)|E_W(T))$. We consider, in this example, that each agent has declared only one temporality (there are no causality links induced by the fact (i)).

![Figure 2: Example of asynchronous execution](image)

The nature of the two environments here considered does not matter. Indeed, the mechanisms we previously presented does not take in consideration any knowledge on the type of environment. We deduce the dependency table from the example’s temporal axis (dependency is noted $\rightarrow$):

- $T_0 \rightarrow T_2(P-1) \land T_3(P-1) \land T_4(P-1)$
- $T_1 \rightarrow T_2(P-1) \land T_3(P-1) \land T_4(P-1)$
- $T_2 \rightarrow T_1$
- $T_3 \rightarrow T_0 \land T_1 \land T_2 \land T_4(P-1)$
- $T_4 \rightarrow T_1$

The notation (P-1) means that the tempo depends on a tempo belonging to the previous global period: in figure 2, $T_0$’s execution depends on the execution of $T_2$, $T_3$ and $T_4$ at the previous period.

The dependency table is valid as long as no agent modify one of its temporalities. Manipulation of temporalities by agents is critical for the scheduler. Let us consider an example: the scheduler triggers a temporality $T_2$ during which the associated agent may perceive the environment $E_1$. Another temporality, $T_1$, have to be executed with $t_{a1} < t_{a2}$ but during which its associated agent will not influence $E_1$. The trouble is that during the execution of the behavior associated with $T_1$, the activated agent may declare a new temporality $T_3$ with $t_{a3} < t_{a2}$ and during which it would influence $E_1$. Such a situation would produce a causality violation as $t_3$ should be activated before $t_2$. The detection issue induced by temporalities modification can be resolved by determining the temporalities which activation may lead to a modification of the associated agent’s temporalities. In our approach, the Temporality model is represented as a temporal environment. Thus, agents declare their temporalities by...
influencing this environment. A temporality activation is seen as a perception in the temporal environment. With such an approach, the issue induced by temporalities modification is transposed into the detection of causality links through environments. The causality links that can be created by modification of temporalities are identified in exactly the same way as for other environments. So, the dependency table naturally takes into account perceptions/influences on the temporal environment.

Scheduling in simulation

We consider here a scheduler that supports distributed and parallel execution. With their temporalities distributed on the execution infrastructure, tempos are executed on several nodes. Each node broadcasts a message when it completes a tempo execution. When all platforms complete a tempo execution, this tempo is declared “accomplished” in the dependencies of the dependency table. Every tempo for which all dependencies are “accomplished” can be executed. We detail how the scheduler works in reusing the previous example of Figure 2. A light gray cell color signifies that the tempo is accomplished. The dark grey cell in the first column denotes the tempos that are in execution. When the simulation begins, all tempos that should be executed at “the previous period”, that are annotated \((P - 1)\), are considered accomplished.

At \(t_{sg} = 0\):

<table>
<thead>
<tr>
<th>(T_0)</th>
<th>(T_2(P-1))</th>
<th>(T_3(P-1))</th>
<th>(T_4(P-1))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T_1)</td>
<td>(T_0)</td>
<td>(T_2(P-1))</td>
<td>(T_3(P-1))</td>
</tr>
<tr>
<td>(T_2)</td>
<td>(T_1)</td>
<td>(T_1)</td>
<td>(T_2)</td>
</tr>
<tr>
<td>(T_3)</td>
<td>(T_6)</td>
<td>(T_1)</td>
<td>(T_2)</td>
</tr>
<tr>
<td>(T_4)</td>
<td>(T_1)</td>
<td>(T_1)</td>
<td>(T_2)</td>
</tr>
</tbody>
</table>

At the end of \(T_0\) execution on each node, \(T_0\) is declared “accomplished” in the dependency table. As all dependencies of \(T_1\) are satisfied, \(T_1\) is executed. After its execution, both \(T_2\) and \(T_3\) can be executed. We so have the following situation:

At \(t_{sg} = t_{s1}\):

<table>
<thead>
<tr>
<th>(T_0)</th>
<th>(T_2(P-1))</th>
<th>(T_3(P-1))</th>
<th>(T_4(P-1))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T_1)</td>
<td>(T_0)</td>
<td>(T_2(P-1))</td>
<td>(T_3(P-1))</td>
</tr>
<tr>
<td>(T_2)</td>
<td>(T_1)</td>
<td>(T_1)</td>
<td>(T_2)</td>
</tr>
<tr>
<td>(T_3)</td>
<td>(T_6)</td>
<td>(T_1)</td>
<td>(T_2)</td>
</tr>
<tr>
<td>(T_4)</td>
<td>(T_1)</td>
<td>(T_1)</td>
<td>(T_2)</td>
</tr>
</tbody>
</table>

At the end of \(T_2\) and \(T_4\) execution, they become accomplished in the dependency table. We underline that \(T_2(P-1)\) and \(T_4(P-1)\) are also accomplished. \(T_3\) can then be executed. When \(T_3(P-1)\) becomes accomplished, the dependency table will be the same as for \(t_{sg} = 0\) as a simulation period has been executed; another period can be executed. In this example, we have seen that the tempo \(T_4\) can be executed in parallel with the tempo \(T_2\).

CONCLUSION AND FUTURE WORKS

In this paper, we proposed a scheduler for the parallel execution of distributed agent-oriented simulations. Our proposal is based on the Temporality model to support different classical temporal models. The parallelization strategy is conservative: observing the perception and influence mechanisms, the agents’ causality links are identified to avoid a violation of the causality principle. We exploit the multi-environment approach and the information associated with temporalities to establish a dependency table between the agents’ activations. This table is used during simulation to enable the earlier-in-the-real-time execution of agents.

Our proposition takes advantages of the simulated model parallelism to aim at better support large-scale systems. Toward this goal, our formalism does not depend on the type of parallel architecture used: we can so exploit all multi-processors architectures (supercomputers or network distribution).

Like most of works on parallel execution of simulations, the performances of the scheduler we presented here strongly depends on the simulated model (e.g. on interactions between agents, temporalities that have been defined, etc.). To take better advantages of simulations characteristics, we are working on an extension of our formalism to support the definition of strategies integrating the simulation’s dynamic aspect. A first research track would be to exploit the temporal information to anticipate agents repartition on the execution infrastructure.

ACKNOWLEDGEMENTS

This publication was made possible by a grant from the regional council of Reunion Island.

REFERENCES


