Generic Simulation Tools Based on MAS Organization

Fabien Michel, Jacques Ferber and Olivier Gutknecht

LIRMM Laboratoire d'Informatique, Robotique et Micro-électronique de Montpellier.
C.N.R.S. - Université Montpellier II, 161 rue Ada 34392 Montpellier Cedex 5 - France
{fmichel, ferber, gutkneco}@lirmm.fr
WWW home page: http://www.lirmm.fr/~{fmichel, ferber, gutkneco}

Abstract. This paper presents generic simulation tools which rely on an original methodological approach of designing multi-agent simulators. We will see that these tools are generic specially because they are not related to a particular scheduling method. On the contrary they aim at providing facilities that allow to design complex activation structures that remain comprehensible, analyzable and moreover modifiable, thanks to a problem division. To achieve this, the main idea of this methodology is to express the multi-agent system (MAS) simulator as a particular MAS itself and use explicitly its subjacent organizational structure. We will show how the AALAADIN organizational model enables us to finely apply such a methodology. Precisely, we will present a particular agent of MADKIT (the platform that relies on AALAADIN): the *Scheduler* agent and its tool called *Activator*.

1 Introduction

As Axtell has pointed out [1], agent interaction and **activation structures** can play important roles in multi-agent systems (MAS). Especially the output results given by a single MAS model can be very different considering the way it has been computed [2] [3]. This observation finds an explanation in the fact that we do not have a suitable formalism to compute agent interactions. Particularly, modeling the simultaneity of those interactions is very difficult by now as theory is almost missing on this particular point[4]. However very few works concern the analysis of the simulators themselves. In other words, we mean that habitually the main preoccupation of simulation designers is to compute agents' behaviours rather than explore different ways of computing the MAS simulation dynamic. So this matter that we will call the **scheduling problem** does not have almost any methodological support whereas paradoxically this is a mandatory stage when computing a MAS simulation.

Indeed, MAS paradigm is founded implicitly on the composition of concurrent individual behaviours [5]. Moreover multiprocessors architectures do not provide an obvious solution: without synchronization, agents evolve at the rate of their internal architecture complexity. Then it is not possible to control the coherence of the entire system. Thus all the solutions planned to simulate entities

concurrency generate more or less skews in the simulation course. In other words the computation of this particular point has depth influence on the evolution of MAS models whereas its neutrality should be wished.

On the other hand, this observation can be an explanation of the multitude of simulation platforms. Indeed, when one wants to compute a MAS model one has two choices: find a platform that fulfill the needs or, at least, develop one from scratch. Considering that almost every simulation platform is domain-specific, and that there is also a great probability for the considered model to be field identifiable too (robotic, ecology, sociology, ethnology, multi-agent coordination, etc), we could expect that everyone can find its wishes. And truly speaking, for many the second choice is a reality. Why this? Because almost all platforms are more or less esoteric in their engineering. By this we mean that most of the time it is difficult, and at least not possible as an external project user, to modify the simulator's basic operations. Especially the scheduling method employed. So if the considered model is complex, it is surely more reassuring to develop from scratch in order to be aware of all the simulation development stages and then be able to understand all the meanings of the output results.

So considering that many have to develop from scratch, we really think that a proposal for generic simulation tools should be very interesting. Rather than give a customized platform, the idea is to provide building tools that are based on a methodology that can be helpful for everybody who wants to compute a MAS simulation platform independently of the model considered. So our first goal is that these tools allow to compute MAS simulators without having to write everything from scratch.

Moreover, as we think that the scheduling problem must take a dominating place when developing a MAS simulation, such a methodology have to be established with the worry of allowing a simple analysis of this problem. Especially, this methodology must make this problem explicit and clarify it.

Without solving the simultaneity matter, the main idea here is not to check an hypothetical match with reality but to better understand our experimentation tools and their implication in the final result, especially in order to be able to evaluate approximations and errors that depend directly on this problem.

Within this framework, our working hypothesis is that any MAS simulator can itself be expressed as a MAS and then, by nature, this system defines a subjacent organizational structure. An explicit use of this structure must provide a solution to the problems that we raised, as well on a description level as on computation of scheduling mechanisms.

We will see how this approach can easily be exhibited using the AALAADIN organizational meta-model[6]. Based on three core concepts -agent/group/role- this model allows simple and powerful description of organizational structures and interactions independently of the agents' internal architecture. Thus its use will enable us to simply define the MAS dynamic by dividing up the global synchronization problem into several sub problems defined as terms of various groups and roles emerging from the whole simulation's model structure. We will

show how this method enables us to compute very different activation structures that can coexist inside only one single simulation.

In the first part of this paper, we specify the stakes of the scheduling problem and we make a synthesis of the current scheduling techniques and the difficulties that they raise. Section 3 describes related works and defines our approach. section 4 exposes the proposed methodology. We present briefly the AALAADIN model and we describe our approach. Then we finish by some prospects.

2 The Problem of Time in MAS

2.1 The Fundamental Scheduling Mechanisms

In a classical way, a simulation consists in trying out models given in the form of mathematical relations between variables representing real physical objects. On the contrary MAS simulation proposes to directly represent the individuals, their behaviours and their interactions [7].

The enumeration of all the works based on MAS simulations could be very tiresome. However we can note that the majority have as an aim to simulate a specific MAS model: a ant colony[8], soccer robots[9], or to offer a more or less generic platform related to a particular applicability: ecology (CORMAS[10], ECHO[11]), robotic (MISSIONLAB[12]), ethology (LIVEWORLD[13]), multi-agent coordination(MASS[14]).

The development of simulation platforms is usually made around the following concepts:

- The agent type considered: reactive, cognitive, situated, etc.
- The environment in which the agents evolve : discrete, continuous, 2D, 3D, etc.
- The interactions nature: coordination, negotiation, perceptions and actions on the environment, speech acts, etc.

being given the variety of MAS applications, this approach, which one can describe as " theoretical model directed ", is justified and models can require thorough analysis.

Modeling Time But such an approach does not reveal the painful problem of modeling time. And it is clear that each model will not require the same level of temporal granularity: from real time in robotics to several years for ecological models. On this particular point, very interesting approaches can be found in [14] and [15].

Simultaneity of Actions In addition, a MAS simulation supposes that one has computed a mechanism allowing to synchronize the agents' actions. Assume Σ defines the whole possible system states, every MAS simulation is based on the assumption that the environment evolution from one moment t to the next

$t+dt$ results from the composition of the actions $A_1(t), A_2(t)...A_n(t)$ produced by the agents at t. In a simplified way, the problem is to build a time function, *Dynamic* $D : \Sigma \mapsto \Sigma$, such as

$$\sigma(t + dt) = D(\Pi A_i(t), \sigma(t)) \quad (1)$$

The symbol Π is used here to appoint the action composition operator. It defines how the actions produced at t must be summoned in order to calculate their consequences on the initial world state $\sigma(t)$.

Without detail this calculus, it is easy to measure the difficulty of conceptualizing such an operation knowing the multitude and the nature of concepts hidden behind the word action (movement, decision-making, environment modification). Moreover, as we said in introduction, the implicit simultaneity of actions is extremely difficult to model. Thus, all MAS simulation designers have to make a personal choice on this matter. And this choice is painful: a MAS model does not define a particular technique itself, as it is precisely a **computational problem**. Thus the same model can be implemented, on this point, in multiple ways. Then it is simple to see that one single "paper model" can give different results according to the scheduling technique used to compute it, like it is shown in [3].

This is why, on the contrary of mathematics in digital simulations (on which the validity and constancy do not depend on a model), the scheduling policy used to compute a MAS model has a crucial impact on the output results.

In a conventional way, the experimentation and the sensibility analysis are used to evaluate the model quality. They participate to corroborate the model or to call it into question for refute. We think that within MAS simulation framework, it is imperative to include the execution policy employed when evaluating a model.

2.2 Usually Scheduling Techniques

In this section we present three kinds of synchronization techniques used in current simulation platforms : simple activation, double buffer and event-based.

" Discrete Time Simulation" This method consists in activating the agents (and possibly the simulation's objects) in a sequential way. Then the activation of the whole system corresponds to a time step for the simulation.

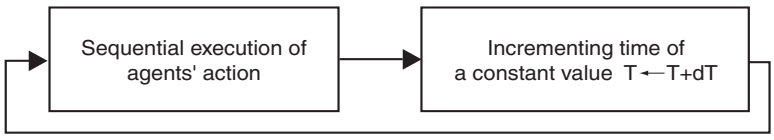

Fig. 1. simple activation loop

This technique has the advantage of avoiding the presence of conflicts in the access to the environment variables in the sense that each gets to execute exactly once during each clock cycle. This form of concurrency makes agents synchronized by default, allowing them to be simpler than they would have to be if they needed to achieve synchronization through explicit constructs. Then, it is by far the most used technique because of its simplicity.

Our object here is not to enumerate all the skews generated by this kind of operation. But, as these skews are related to the difficulty of modeling simultaneity in MAS, the problems raised by this kind of operation can easily be extended to others. So, in order to show the significance of activation structures in simulation courses, we will use two simple examples.

Our first example describes a simulation model known as " prey/predator ". We pose that a prey (a triangle) is captured when surrounded by four predators (circles). One will find a description more exhaustive as well as experimental results in[4].

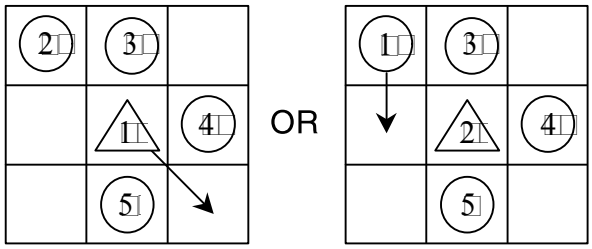

Fig. 2. the prey's survival depends on the ordering of agents

Figure 2 shows that the ordering of agents (represented here by a number) can change the result with the same initial situation. Here, the prey is living or dies. Randomize the ordering of agents to smooth the problem is a regularly used method (like [16]).

Our second example stresses the problem of time granularity of actions. This example is drawn from the StarLogo platform [5]. A turtle (agents' denominations in this model) can carry out, in each time step, a primitive action. Here the primitive considered is *"fd N"*. The parameter N means that the turtle goes N patches forward.

Figure 3 shows that, as N is not fixed, several turtles can have cross trajectories without being at no moment on the same patch, due to the fact that the movement speed has no direct link with time.

Double Buffer Extension of the precedent one, this technique aims at bringing a solution to simultaneity problem. The main idea is to make all the agents having the same perception of the world state at one moment t. To do this, the idea is that all action effects are delayed until the end of an entire cycle. So

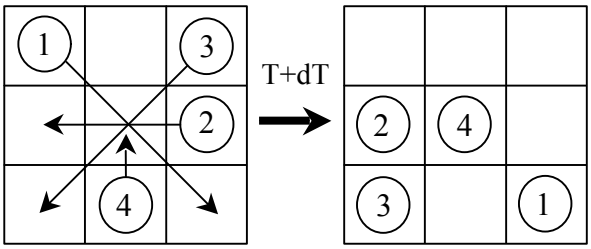

Fig. 3. the problem of action granularity

the agent actions are done on buffered variables and the world is not directly modified. Once all the agents carried out, the second phase takes place to make the new values current.

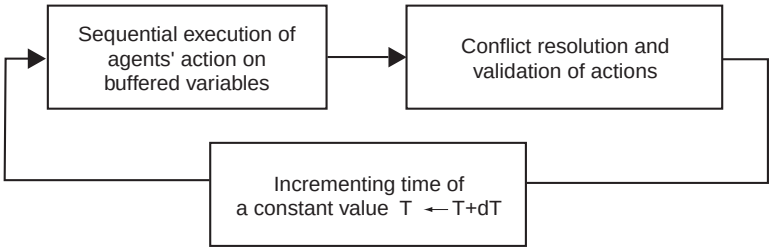

Fig. 4. double buffer simulation

For example the Conway's game of life is relied on this technique: the state of the world is updated once that all the cells calculated their next state. The Travers's, LiveWorld[13], platform provides this operating mode.

With the use of this method there is a new coherence problem that arises. A conflict appears when two agents or more specify different values for the same variable. Thus the conflicts that lead to an inconsistency of the world state should be solved.

In addition, as Travers saw himself, there is a serious drawback : if an agent modifies a variable, it cannot make use of it in further computation without obtaining an incoherent result, knowing that the variable value is only validated at the end of a cycle.

Without taking into consideration the difficulties related to conflict resolutions, we can simply notice that, in a way or another, we have to set up a priority mechanism between agents at the time of conflict resolution. Thus, even if this technique relies on an analysis of the situation rather than a preset list of scheduling, one can easily finds situations similar to those of our preceding examples.

Event-Based Simulation Rather than synchronizing all the agents at one moment t, the idea is to explicitly preset a chronological order between the agent actions. It consists in determining first, or in the simulation course, the future events, their date and their nature.

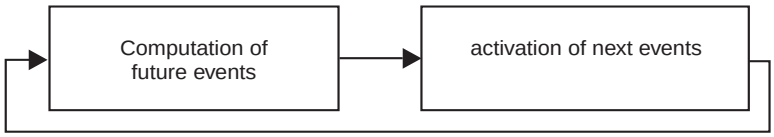

Fig. 5. principle of event-based simulation

For example, in his SIEME simulator, Magnin[9] uses a set of environmental rules to progressively determine the events to proceed : if $<$ *event condition* $>$ then $<$ *proceed event* $>$. Thus a causality law between events is his main worry.

On the contrary in the platform SWARM[17], interactions chronology is settled at first. A swarm is defined as a collection of objects together with a schedule of activity over those objects. It is this schedule of activity that defines the system dynamic. Although this kind of simulation seems far from the precedents, it does not remain about it that the same kinds of problems are found. Indeed, when several events are concurrent, the difficulties of simulating simultaneity remain the same ones.

3 Discussion

3.1 The Need of Flexibly Activation Stuctures

As we saw, implement a MAS simulation is somehow difficult when the scheduling problem arises. Especially knowing that a particular model is not related to a particular scheduling technique and that all these methods are, one way or another, skew producers. Nevertheless a choice should be made. But, in our point of view, the simple fact of making on the matter a unilateral and unchangeable choice is an error. It should be noticed that the simulation operation is always based on a single strategy.

Thus almost all the simulation agents are subjected to it. For example SWARM afford generic tools but the whole platform is based on a single scheduling policy.

From there, compute complex models that comprise several kinds of agents can be very tricky knowing that action granularity (movement, decision, etc.) and interaction semantic (collisions, coordination, etc.) can be very different.

Moreover it is not occasional that processes that come with the agents execution (displays, statistical investigation, etc) are also managed on the same guidelines. For example, in an event based system, the graphical display could be a particular event that should be managed like the others. Thus the fact that all the simulation processes are subjected to the same scheduling routine

contributes to make difficult, and at least unachievable, to really change the simulator's basic operations. So it could be very difficult to analyze the scheduling impact on the output results. Furthermore, it degrades the possibility of extension to other agents.

3.2 Related Works

Some approaches are not related to a particular implementation and aim at providing a formal framework for MAS design. For example, DESIRE [18] is a framework for the design and formal specification of compositional systems and relies on a task-based approach. A task hierarchy is used to define components distinguished within a formal specification. Interaction between components is formally specified by *information links* between components to model complex behaviour. As DESIRE does not define a representation of time, the modeler is not limited to a particular one. For example, this compositional development method was used to design the Generic Agent Model, GAM [19], that abstracts from specific application domain. Within this model, several components (asynchronous processes) are composed and linked to model the whole *agent task control*.

SDML [15], which is a strictly declarative language that corresponds to a fragment of Konoliges strongly grounded autoepistemic logic, is also not limited to a particular scheduling method. Within SDML, agents incorporate rules that determine their behaviour. SDML's rules are fired in respect of declarative clauses contained within agents' databases that can be shared. So it supports representations of interaction and communication by agents to assert clauses to one anothers databases. It also supports nested time levels and, within time levels, agents can act in parallel for instance. So it enables the user to finely specify a particular scheduling mechanism using different agent types (*serial*, *parallel* and *merging composite agent*) and different grains of time levels.

3.3 Position of our Approach

Concerning us, we will focus our attention on a higher level of abstraction. Our main goal is not to provide to users a complete solution for designing MAS simulations but sound means to do it. Both SDML and DESIRE, or other formal approaches like [20], are generic in the sense that they formally specify MAS, using logical formalisms, in an implementation -and domain- independent manner at a high level of abstraction. None the less, the way they model MAS is quite advanced and they propose a particular solution for modeling agents' actions and interactions: *information links* between components in DESIRE, rule-firing and assertion of clauses in shared databases in SDML. So it is difficult, within these kind of approaches, to incorporate other ways of modeling agent interactions like the *influence/reaction* [21] approach.

As we aim at providing tools that can be helpful to develop every kind of simulators, that is to say every way of simulating MAS (from empirical ones to formal approaches like SDML), we must make the fewest assumptions as possible

on how a MAS can be simulated. On the matter we can say one thing: most of the time, MAS simulation is about organizing method invocations over objects. If such a work can be done using an implicit methodology, we believe that the inherent use of it will be in some way useful.

4 The Simulator as a Particular MAS

4.1 Organize-and-Conquer

Our idea is simple and besides it is not new in the guidelines. It is about dividing the global scheduling problem into as many of sub problems that necessary. Indeed, it is enough to notice that - whatever the method - synchronize all agents does not have a real meaning when considering several kinds of agents with different scheduling requirements. Thus it is more operative to concentrate on an agent group where synchronization is crucial for the simulation process. Each group can then be independently handled in order to identify a scheduling protocol adapted to the situation. These groups then define naturally an organizational structure that reflects explicitly the simulator logic.

4.2 The AALAADIN model and the MADKIT platform

We present here very briefly the organizational model AALAADIN[6] and the MADKIT[1][22] platform which relies on it.

This organizational model is based on three core concepts : *agent*, *group* and *role* (AGR). Figure 6 presents a diagram of this model.

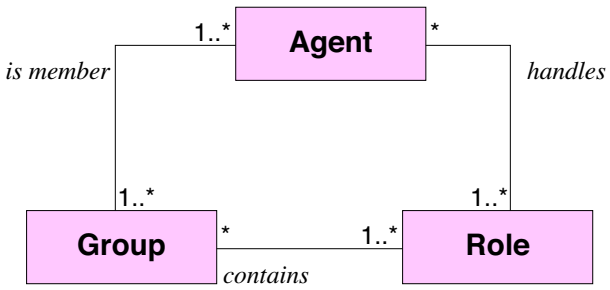

Fig. 6. Agent/Group/Role Model

Agent This model places no constraints on the internal architecture of agents and does not assume any formalism for individual agents. An agent is only specified as an active communicating entity which plays roles within groups.

[1] refer to [23] for a detailed description of its formal operation semantics.

Group We define groups as atomic sets of agent aggregation. Each agent is part of one or more groups. In its most basic form, the group is only a way to tag a set of agents. In a more developed form, in conjunction with the role definition, it may represent any usual MAS. An agent can be a member of n groups at the same time. A major point of AALAADIN groups is that they can freely overlap.

Role The role is an abstract representation of an agent function, service or identification within a group. Each agent can handle several roles, and each role handled by an agent is local to a group.
The MADKIT platform implements these concepts and adds two design principles: a micro-kernel architecture and the agentification of the services.

4.3 The Scheduler Agent of MADKIT

Our idea is to have transposed the MADKIT's design principles to simulation. We thus implemented a micro-kernel dedicated to simulation called synchronous engine. So, a MADKIT micro-kernel special agent, the *Scheduler* agent, offer services related to the design of heterogeneous scheduling methods.

Its role consists in handling execution policies on which no constraint is supposed. Thus this agent is associated with a generic tool object called *Activator*. In its simplest form, an Activator is simply a mean for the Scheduler to identify a particular agent set given a group and a role. For example, the Scheduler can create an Activator on the role *agent* within the group *simulation*, or *displayer* within *Graphical interface*.

The idea is to specialize subclasses of *Activator* in order to define a particular scheduling procedure that could then be applied punctually, by the scheduler, on various groups of agents. The advantage lies in the fact that the same Scheduler agent can create as many activators as necessary. Its task is simply summarized in organizing the activation of its activators in order to define the whole simulation process.

Figure 7 shows an example where a Scheduler S comprises two different kinds of activators (two different scheduling policies) on three different groups. Moreover this figure shows that an agent can belong to several groups simultaneously.

The principal advantage of dividing the global scheduling lies in the possibility of modifying, or replacing, an activator without having to touch the whole structure : we have uncoupled the simulation management problems (the Scheduler's work) from the agent synchronization problem (The Activator's work). An activator is the place where the modeler have to deal with the problems illustrated in figures 2 and 3. So, one can notice that a Scheduler is not responsible for the quality of a particular activator[2]. Additionally, a Scheduler agent can, as a regular one, be scheduled by an agent hierarchically higher.

[2] Thus, the designer is also responsible of the simulation complexity in terms of time, knowing that the MADKIT's micro-kernel is responsible for maintaining correct information about group members and roles handled

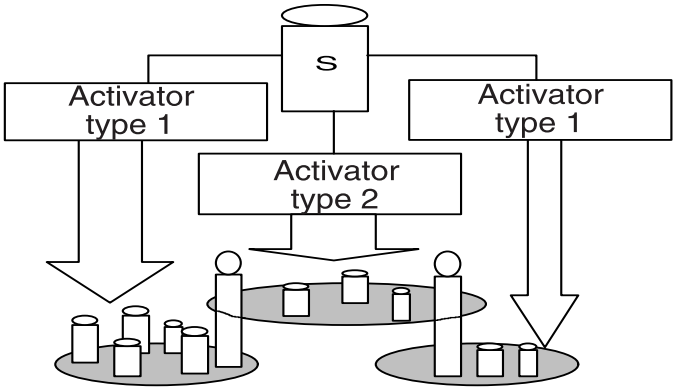

Fig. 7. An organizational simulator

Another advantage is that this mechanism is not related to the internal architecture of agents. Thus an agent (a particular Java class) can be substituted by another one (another class) without having to change the scheduler or its activators. By the way, an agent can enter a simulation process just by playing the right role in the right group as an Activator overlooks this particular couple. So you can compile an agent during a simulation and then put it within the simulation without having to stop it.

4.4 Design Methodology of an Organizational Simulator

We suggest a design methodology of MAS simulators that consists in three stages:

1. Express the simulator organizational structure in terms of groups and roles.
2. Work out new kinds of Activators if necessary for the considered simulation (i.e local scheduling policies).
3. Define the whole simulator logic by ordering the activators execution.

The statement of the third stage can make one believes that we are dealing with the same problems that we raised in the second section. In fact, an activator, related to a group, defines a particular scheduling method (event based, double buffer, etc.) for agents whose synchronization is regarded as necessary. On the contrary, the activators activation simply describes simulator logic, that is to say the specific order in which the various key groups (simulated agents, displays, observations, etc.) intervene in the simulation course.

Figure 8 illustrates this aspect of our approach. It describes an example where four different activators *A1*, *A2*, *A3* and *A4* are used. Each activator is related to an agent set identified by a group and a role. The scheduling policies of used by the various activators are independent. For example *A1* can use a double buffer method whereas *A2* defines an event-based principle

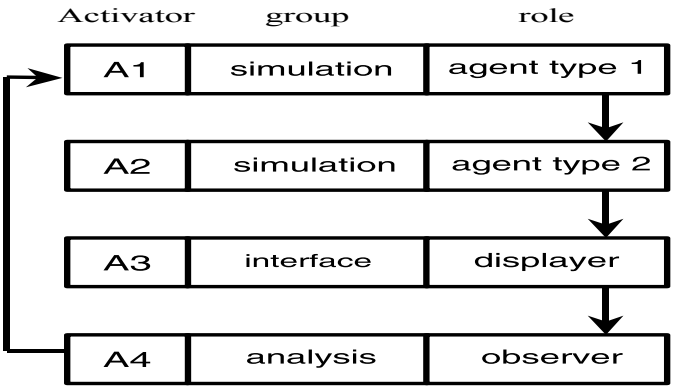

Fig. 8. Principle of simulation using activators

However it is obvious that the ordering of the activators can influence the results. Thus if one reverses *A1* and *A2*, one can obtains different results since these activators handle agents that can modify the world. Nevertheless it is not a synchronization problem here : this split into two activators implies that these various groups of agents should not intervene at the same time.

On the other hand it is completely possible to consider, after analysis of the results, that the synchronization of two groups is finally necessary. It is then very interesting to be able to change locally the simulator operation. For example, we can imagine the fusion of *A1* and *A2* in only one more adequate activator, *A3* and *A4* remain valid.

The Activator interchangeability obtained thanks to the organizational sight, finds all its meaning here: it is now possible to quickly constitute several simulations based on various scheduling methods without having to rewrite the simulation agents.

We claim that such a methodology is well suited to build very complex scheduling procedures that remain intelligible thanks to the problem division. Moreover, being able to locally modify the simulator's operations permits the realization of simulations increasingly complex comprising a great number of strongly heterogeneous agents.

4.5 Implementation Code Example

We present here a simple example of the scheduler's implementation code related to our approach. In the following example, a Scheduler uses one of the simplest activator that we have developed: the *SingleMethodActivator*. This activator is used to invoke a particular method over an agent set. First the scheduler must instantiate needed activators.

The first constructor's parameter represents the method's name, the second one is the group and the third is the role.

```
observers =
      new SingleMethodActivator("watch","simulation","observer");
rabbits =
      new SingleMethodActivator ("doIt", "simulation", "rabbit");
coyotes =
      new SingleMethodActivator ("doIt", "simulation", "coyotes");
guis =
      new SingleMethodActivator ("display", "simulation", "gui");
```

Then add the activators to the system:

```
addActivator(rabbits); addActivator(coyotes);
addActivator(guis); addActivator(observers);
```

Then the scheduler has just to execute the activators during its life cycle to define the whole simulation process:

```
While(true) {
rabbits.execute();
coyotes.execute();
observers.execute();
guis.execute(); }
```

A first variant of this simulation is to reverse the ordering of the two first activators for example. However, if one is interested in trying out a special activation structure, one can freely define a new kind of activator that fulfill the needs. It can be about trying to solve the simultaneity matter for instance. So the rabbits and coyotes activators can be replaced by one more complex activator. Call it *swarm* for instance. Then the two first lines become the following one:

```
swarm = new SwarmActivator("simulation", "animat");
```

and the Scheduler's life cycle becomes:

```
While(true) {
swarm.execute();
observers.execute();
guis.execute(); }
```

This implementation code just defines the simulation's guidelines. But if one is interested in further understanding, it is obvious that one has to take a look inside the *SwarmActivator*'s implementation. In this object, one will find only what is of interest: the local activation structure used to schedule the simulated animats.

4.6 Advantages and Drawbacks

Additionally to the advantages about which we have already discussed, we believe that the simplicity of Schedulers' implementation code allows our methodology to be easily understood by external project users. Moreover, these tools are only a part of the MadKit platform. So, a particular simulation platform is only regarded as a particular MAS within MadKit. Thus, simulator designers benefit from other works done within the platform. Even those that are not related to simulation (system agents, interface agents, test agents).

This advantage is also a drawback. Even if these tools are made to be the more generic as possible, they rely on a specific organizational model and on the platform related to it. And users have to define several group and role in respect of the Aalaadin model. And, although organization seems to be a key point in representing MAS, Aalaadin is only one point of view.

Another shortcoming is that, for now, our approach does not provide any formal support like Sdml for example. In fact, we believe that the level of abstraction on which we focus by now is an explanation to this.

4.7 Applications

By now, all the simulations produced by our MAS team rely on these tools. From toy simulations to more complex ones like the TurtleKit of MadKit.

The last one is a library of simulation for reactive agents that mimic a part of the functionalities of StarLogo[5]. Although TurtleKit looks like a scaled-down StarLogo clone, the *turtles*, and all the agents of the simulation (*launchers*, *observers*, *viewers*) are actual MadKit agents written in Java. Knowing that the MadKit platform does not impose any constraint on the agent architecture, a turtle or an observer can be freely defined. Thus a *turtle* is not limited to its basic commands. Moreover, as MadKit messaging engine works using the Agent/Group/Role model, the simulation's agents have the ability to communicate using messages with any other agent running in MadKit and not only with these of their own simulation. Thus it is also possible to use all the usual functionalities of the platform and its tools like the *MessageTracer* agent of MadKit.

Far from this last example, Simonin[24] is currently using the synchronous engine to simulate autonomous mobile robots. Another proof that the synchronous engine is not limited to a particular domain and can be easily reused.

5 Conclusions

In this paper we have seen that the scheduling is a key problem within the framework of MAS simulation. So, as our main goal was to provide generic simulation tools, we have proposed ones that are not related to a particular scheduling method. Moreover, as these tools are a part of the MadKit's kernel, they place no constraint on the internal architecture of agents.

We also argued that an explicit use of MAS organizational structures can be very helpful to conceive MAS simulators. The methodology presented here is based on this idea. We claim that this methodology can give rise to complex scheduling policies which remain comprehensible (thus analysable) thanks to the localization of synchronization problems.

So, our future works will consist in try out new scheduling methods. Especially, it would be very interesting to use more complex computation models of action that have not yet been used in MAS simulations like the influence/reaction model[21].

On the other hand, we have seen that a MAS model must not be evaluated independently of the scheduling technique used to implement it. In this respect, we argue that the possibility of modifying locally the simulator, is a facility for investigating MAS models' behaviours.

We hope that it is the first step towards a thorough and more profitable analysis of multi-agent simulation mechanisms.

References

1. Axtell R.L.: Effects of Interaction Topology and Activation Regime in Several Multi-Agent Systems. In the Second Workshop on Multi Agent Based Simulation. MABS-2000 (LNAI 1979), July 2000.
2. B. A. Huberman and N. S. Glance. Evolutionary Games and Computer Simulations. Proceedings of the National Academy of Science USA, 90(August):7716-7718, 1993.
3. Lawson B.G. and Park S.: Asynchronous Time Evolution in an Artificial Society Mode. In Journal of Artificial Societies and Social Simulation vol. 3, no. 1, http://www.soc.surrey.ac.uk/JASSS/3/1/2.html
4. Ferber J.: Multi-Agent Systems An Introduction to Distributed Artificial Intelligence. Addison-Wesley, 1995.
5. Resnick M.: New Paradigms for Computing, new Paradigms for Thinking. Computers and Exploratory Learning, edited by A. diSessa, C. Hoyles, and R. Noss. Springer-Verlag, 1995.
6. Ferber J. and Gutknecht O.: A meta-model for the analysis and design of arganizations in multi-agent systems. In Proceedings of Third Internationnal Conference on Multi-Agent Systems (ICMAS'98), pp. 128-135, IEEE Computer Society, 1998.
7. Parunak H.V.D., Savit R., and Riolo R. L.: Agent-Based Modeling vs. Equation-Based Modeling: A Case Study and Users' Guide. In Proceedings of Workshop on Modelling Agent Based Systems. MABS'98, Paris, 1998.
8. Drogoul A. and Ferber J.: Multi-Agent Simulation as a Tool for Modeling Societies: Application to Social Differentiation in Ant Colonies. In Proceedings of MAAMAW'92, Viterbo, 1992.
9. Magnin L.: Modélisation et simulation de l'environnement dans les systèmes multi-agents: application aux robots footballeurs. Doctor Thesis. université Paris VI, 1996.
10. Bousquet F., Bakam I., Proton H., and Lepage C.: Cormas: common-pool ressources and multi-agent systems. Lectures Notes in Artificial Intelligence 1416, pp. 826-837, 1997.

11. Hraber P. T., Jones T., and Forrest. S.: The Ecology of Echo. In Artificial Life Vol.3, No.3, pp. 165-190, 1997.
12. MacKenzie D., Arkin R.C., and Cameron R.: Multiagent Mission Specification and Execution. Autonomous Robots, Vol. 4, No. 1, pages 29-52, January 1997.
13. Travers M. D.: Programming with Agents: New metaphors for thinking about computation. Thesis, Massachusetts Institute of Technology, pp. 127-137 1996.
14. Horling B., Lesser V., and Vincent R: Multi-Agent System Simulation Framework. In 16^{th} IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation, EPFL, Lausanne, Switzerland, August, 2000.
15. Moss S., Gaylard H., Wallis S., and Edmonds B.: SDML: A Multi-Agent Language for Organizational Modelling. Computational and Mathematical Organization Theory, CMOT-98 v. 4, pp. 43-70, 1998.
16. J.M. Epstein. et R. Axtell. Growing Artificial Societies. Brookings Institution Press, Washington D.C. 1996.
17. Minar N., Burkhart R., Langton C.,and Askenazi, M., "The Swarm Simulation System, A Toolkit for Building Multi-Agent Simulations", June 1996, http://www.santafe.edu/projects/swarm/overview/overview.html.
18. Brazier F.M.T., Keplicz B.D., Jennings N.R., and Treur J.: Formal Specification of Multi-Agent Systems: a Real World Case. In First Int. Conference on Multi-Agent Systems, ICMAS-95, MIT Press, pp. 25-32, 1995.
19. Brazier F.M.T.,Jonker C., and Treur J.: Compositional Design and Reuse of a Generic Agent Model. In Applied Artificial Intelligence Journal, pp. 491-538, 1999.
20. Hilaire V., Koukam A., Gruer P., and Müller J.P.: Formal Specification and Prototyping of Multi-agent Systems. In Engineering Societies in the Agents' World, ESAW'00, (LNAI 1972) pp. 114-127, 2000.
21. Ferber J. and Müller J.P.: Influences and Reactions: a Model of Situated Multiagent Systems. In Proceedings of Second Internationnal Conference on Multi-Agent Systems (ICMAS'96), pp. 72-79, 1996.
22. Gutknecht O. and Ferber J.: The MadKit Agent Platform Architecture. 1^{st} Workshop on Infrastructure for Scalable Multi-Agent Systems , Barcelona, June 2000.
23. Ferber J. and Gutknecht O.: Operational semantics of a role-basd agent architecture. In Proceedings of the 6^{th} Int. Workshop on Agent Theories, Architectures and Languages. Springer-Verlag, 1999.
24. Simonin O. and Ferber J.: Modeling Self Satisfaction and Altruism to handle Action Selection and Reactive Cooperation. In the proceedings supplement of The Sixth International Conference on Simulation Adaptative Behavior (SAB-2000), pp. 314-323, Paris, Septembre 2000.