Dynamically Configurable Multi-agent Simulation for Crisis Management

Fabien Badeig, Flavien Balbo, Mahdi Zargayouna

Dynamically configurable Multi-Agent Simulation for Crisis Management

Fabien Badeig[1] and Flavien Balbo[1] and Mahdi Zargayouna[2]

[1] ENS Mines
158 Cours Fauriel, F-42100 Saint-Etienne, France
[2] Université Paris-Est, IFSTTAR, GRETTIA
Boulevard Newton, F-77447 Marne la Vallée Cedex 2, France

Abstract. Multiagent-based simulation (MABS) is the processing of a multiagent model of a complex system by a simulation platform that controls its execution. The objective is the understanding of the dynamic of this complex system with the experimenting of different configurations for the same multiagent model. Following a scheduling process, an activated agent has to act according to his context, that is his current perceptible simulation state. In this paper, we propose to delegate the context computation process to the scheduling process. This approach has several advantages. The first is an optimization of the context computation, a single computation being used for several agents. The second advantage is a more configurable design process and a simplification of the reusability of agent behaviors in different simulations. The model that we propose gives a formal framework to support this context computation delegation while preserving agents' autonomy. We describe a crisis situation to illustrate the benefits of our model and compare our approach with a classical simulation scheduling approach.

Keywords: Simulation design; Scheduling policy; Multiagent environment; Crisis situation

1 Introduction

Multiagent systems (MAS) deal with the coordination of autonomous agents interacting in an environment to solve problems. This bottom-up modeling approach has often been used to model complex systems, for instance in social sciences [1], in the military domain [2], or in transportation [3].

Multiagent-based simulation (MABS) combines the advantages of the MAS paradigm, with the advantages of simulation. MABS is consequently a fitting solution to model complex processes and understand their dynamics. These features explain the uptake of MABS by several domains and why it is becoming one of the main directions for the deployment of multiagent systems [4]. However, the agent-centered design of MABS has two limitations. On the one hand, the simulation design is hardly configurable. Indeed, the MABS designer has two options to test new configurations, he can either modify the scheduler, and/or change the agent behavior associated to a context.

These two options often imply that the user of the simulation is either its designer or has only a set of predefined configurations he could use. The second limitation is related to the computational efficiency of the simulation process. Using their context, the agents decide which action they have to perform. For each agent, the context computation takes into account his available information including his own state, his accessible information about other agents, his local environment perception, etc. This context computation is automatic and repetitive for each scheduled agent. Indeed, in the majority of MABS frameworks, all agents must automatically compute their new context before performing an action, even if their context has not changed since their last execution. This computation is repetitive because the agents compute their context at each execution, even if they have the same context than others. This context computation process is time-consuming and is one of the barriers to increased use of MABS for large simulations.

In this paper, we propose an approach that improves the *configurability* of the simulation process by delegating the context computation to the multiagent environment. Thus, the designer can change the simulation behavior by modifying the agent context computation without modifying the agent's implementation. In addition, our approach is more *dynamic* than a classical approach because the modification of the agent and simulation behaviors are possible both offline (by the designer) and online (by the agents). Finally, our approach is *computationally efficient* because the same context computation can be used for several agents who desire to be activated with the same context conditions.

The remainder of the paper is organized as follows. Section 2 presents the issues related to the activation process in simulation. In section 3, we position our work in a practical context of crisis management. The agents design model is provided in section 4. Section 5 presents our experiments and results. The paper concludes with some perspectives to this work.

2 State of the Art

The execution of a multiagent-based simulation (MABS) model necessitates a scheduling process (performed by a scheduler) that synchronizes the agents execution. In this section, we give a presentation of the MABS frameworks focused on this activation process.

In the current MABS framework, we distinguish three ways for the scheduler to activate the agents. The first activation mechanism is the default activation mechanism. It consists in activating the agents with only one method. In this case, the scheduler keeps a minimal description of the agent like his internal clock or his identification information, and activates the agents either by calling a default method [5] or with a control message [6]. In this case, the scheduler responsibility towards the agents is minimal and the agents compute locally the perception-decision-action loop.

The second activation mechanism consists in activating the agents directly with a reference to the next action to perform. In this case, the scheduler keeps,

in addition of the minimal description of the agent, an information about the next agent action to perform. The agent delegates to the scheduler the call of his actions. In the logo-based multiagent platforms such as the TurtleKit simulation tool of MADKIT [7], an agent has an automaton that determines the next action that should be executed. The action activation is delegated to the scheduler.

The third and last activation mechanism is called *contextual activation*. It consists in activating the agents with some kind of context information. In this case, the scheduler keeps a detailed description of the agents and computes each context for the agents according to the agent focuses and the accessible descriptions. The JEDI framework [8] and the Repast Simphony simulation platform [9] are the only two proposals where the choice of the action that is executed by an agent can be computed by the scheduler. In the JEDI framework, the computation of the agent action decision is based on an interaction matrix where a cell is a conditioned interaction between two agents. The interaction is conditioned following the MABS state, i.e. a specific context. For instance, an interaction is possible between two agents following their proximity. At each of these contexts, an action is associated and will be executed by the activated agent. This matrix is given by the designer and do not change during the simulation. This proposal is limited by the choice of the matrix to specify the interaction. Indeed, the number of components that can be taken into account to condition an interaction is limited to the matrix dimensions. Moreover, this matrix cannot be changed by the agents, their autonomy is therefore limited. The Repast Simphony simulation platform natively uses the first scheduling options (i.e. with a default agents' method to call and no information given to the agent), but it also allows a sort of *contextual activation* based on "watchers". Watchers allow an agent to be notified of a state change in another agent and schedule the resulting action. The designer specifies which agent to watch and a query condition that must be verified to trigger the resulting action. This activation process is coherent with our definition of a contextual activation. However, it is limited by the expressiveness of the watcher queries language to express the activation context. Indeed, the queries are boolean expressions that evaluate the watcher and the watchee using primitives such as *colocated* and *linked_to* and the logic operators AND and OR. More importantly, it is not possible to integrate complex conditions about other components (different from the watcher and the watchee).

We propose a simulation model called EASS (Environment as Active Support for Simulation) proposing a complete contextual activation. The context computation is delegated to the environment and where agent activation is based on context evaluation. Our proposal is based on the accessible description of the MABS components inside the environment and the computation of the agents contexts by the environment. The language to express matching conditions is defined such that it is the most expressive possible to define a context.

3 A Crisis Management Simulation

We consider a crisis management application in transportation where several emergency services must be coordinated to limit the crisis consequences. The various activities in crisis response are coordinated between services. For instance, the task of evacuation may first need authorization from the mayor, it could involve military personnel for transport, the fire brigade to clear the road and aid agencies to provide shelter and food. The problem is further complicated by the shifting goals of the various organizations when their priorities change due to the highly dynamic nature of crisis situations. A crisis situation is a dynamic and complex phenomenon characterized by the initial situation (location and time, impact on population and infrastructure), and this situation requires a specific organization to limit the negative effects. In [10], we propose an organization-based modeling of a crisis in transportation. With the victim evacuation example, we illustrate in this paper how the agent behavior is modeled by the designer and dynamically supported by the scheduling process.

This task of evacuation consists in coordinating different emergency agents to secure and carry the victims. We consider two categories of emergency agents. The first is related to the emergency agents playing the role of *physician*. The goal of the physicians is to make a clinical diagnosis to stabilize the victim and to define the suitable evacuation. The second category is related to the emergency agents who play the role *medical porter*. The goal of the medical porters is to transport a victim to an emergency vehicle. In our implementation, each agent is situated on a two-dimensional space environment. The emergency agents have a perception field that limits their perception of the environment. In this example, the victims are objects because they are not autonomous. Figure 1 depicts an example of crisis situation with the stakeholders.

The evacuation process requires to follow sequentially two actions: 1/ A physician makes a diagnosis, 2/ Two medical porters carry the victim. The second action needs a synchronization of the two medical porters. Each medical porter has a specific capacity among *medical monitoring* or *victim handling*. The first capacity allows the medical porter to monitor the victim and to inform the physician of the victim state evolution. The second capacity is necessary to carry the victim. To transport a victim, two medical porters with complementary capacities are required. In this evacuation process, physicians and medical porters have to cooperate: a diagnosis of the victim has to be done before to transport him. Medical porters have to coordinate themselves to transport a victim together.

4 Agents Design

One of the main roles of the MAS designer is to define the behaviors of the agents. In Eass, this is done by defining and composing "filters". A Filter is a set of parameterized conditions, i.e. the context, and the associated action to be

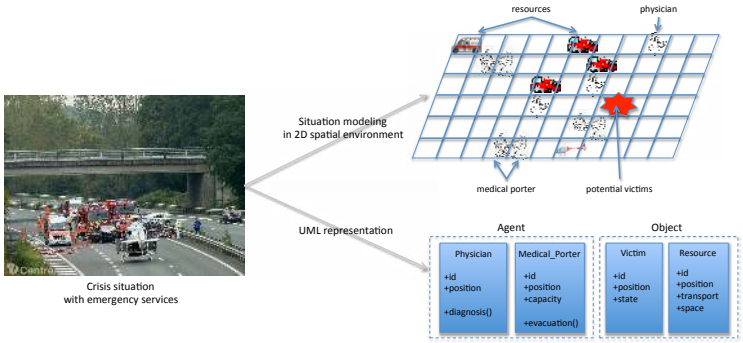

Fig. 1. An example of transportation crisis situation with emergency services to evacuate victims.

performed [3]. A context is a set of constraints on the descriptions of the MAS components (agents, messages or objects). For instance, a filter could express the following context: "if there is a victim nearby (condition 1), and a free medical porter nearby (condition 2), then carry the victim (action)". The context here is made of conditions 1 and 2.

We propose to the designer an agent design process of three steps (cf. Figure 2). The first step, called *enumeration* step, consists in specifying all the possible filters that could be useful to each type of agent (e.g. the *physician* and the *medical porter* in our example). The second step, called *behavior* step, is to design the behaviors of the agents, i.e. how the state of the agents is updated, and which filters are relevant in each state. The last step, called the *simulation* step, is to execute a simulation and to ensure its coherence.

4.1 The Enumeration Step

The *enumeration* step takes as input the description of the MAS model, and provides as output a filters library. The MAS model is composed of the description of the system entities (the categories of agents and objects) and the possible actions of the agents. For each agent type, the designer enumerates all the relevant contexts and associates, to each context, one or several actions. Then, according to these associations between contexts and actions, the designer defines the filter library.

For instance in the crisis simulation, one possible action for the physician is *move* (moving towards a victim) and the context of this action is the perception of a victim in the physician field of view, which is not handled by another physician or a medical porter. With these two information (the action and the context), the designer defines the filter that link the context with the relevant action.

[3] a formal definition of filters can be found in [11]

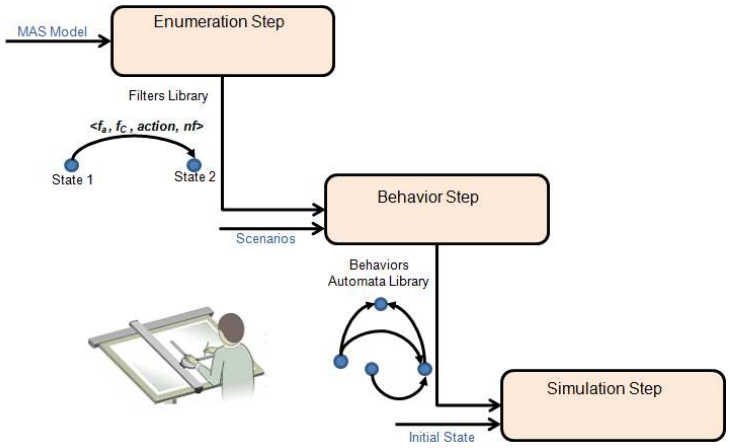

Fig. 2. The three steps of the agents design process

The output of the enumeration step is a library of filters, associating contexts to actions, which is one of the two inputs of the behavior step.

4.2 The Behavior Step

Following Figure 2, the behavior step takes as input the filters library defined in the enumeration step and the scenarios that the designer wants to simulate. The output of this step is a library of behaviors. A behavior is an organized collection of filters that is related to a scenario. Each behavior is defined in the form of an automaton where the nodes are agent states and the edges are filters from the library.

The scenario describes the workflow of a part of the simulation. The designer uses this workflow to define which filters are taken into account and when. For instance, consider a crisis scenario "priority to evacuation". In this scenario, the main objective is to evacuate the victims. However, a victim cannot be evacuated until it is handled by a physician who provide a first diagnosis. When designing the behaviors of physicians, the designer translates this scenario to a behavior automaton.

In this scenario (cf. Fig. 3), from state 1, where the physician is looking for a victim, the filter $Filter_1$ might match and the physician would go to state 2 where he would have found a victim without a porter. For this filter to match, the agent a has to be free, i.e. not engaged to handle another victim, the victim should be in the perception field of a and the victim should not be already handled by another physician or a medical porter. This filter is of priority 1. From state 1, there is another filter ($Filter_2$) that could match and take the physician to another state (state 3), in which he would have found a victim with a medical porter. The filter linking state 1 and state 3 (priority 2) is of higher priority than the filter linking state 1 and state 2 (priority 1). Indeed, the victims

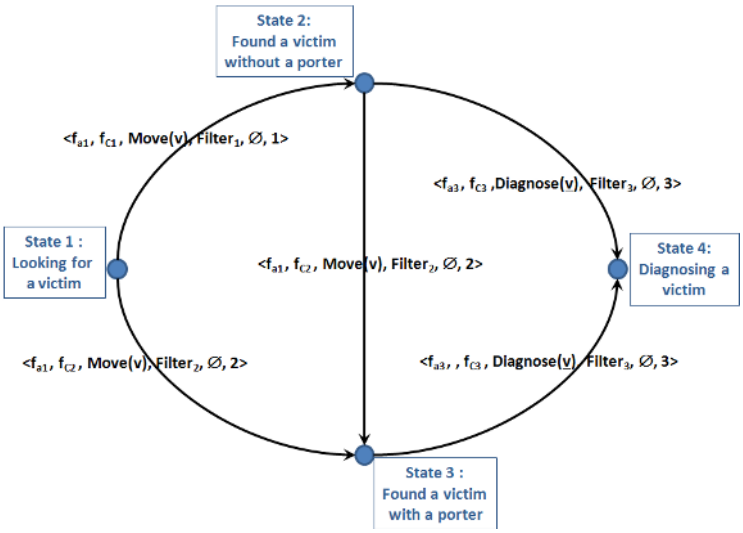

Fig. 3. Example physician behavior (priority to evacuation scenario)

that are already handled by a medical porter would be evacuated immediately after the physician's diagnosis, while the others would still wait for a medical porter.

In state 2, the physician is engaged for handling a victim $\underline{v}$, but he has not reached it yet. Two other filters matching might change the state of the agent. The first is the same filter $Filter_2$ linking state 1 and state 3, i.e. another victim is found that is not handled by another physician but is however handled by a medical porter. A filter of higher priority ($Filter_3$) links state 2 to state 4, which is related to the case the physician has reached the victim, he should then diagnose it. The context assertion is the co-location of the physician and the victim. The same filter links state 3 to state 4.

This step provides the simulation with a library of behaviors that the agent might adopt during the simulation.

4.3 The Simulation Step

Finally, the simulation step is the process taking as input the behaviors library and the initial state of the world, and which runs the simulation. During the simulation execution, the agent activation has to be managed, as well as the priority and the pooling of the filters.

In our framework, the environment plays a main role as a dynamic component of the simulation system. Indeed, the environment manages the filters built during the modeling phase, as well as the descriptive information about the simulation components (agents and objects).

4.4 An Example of a Crisis Application Execution

This section illustrates the global scheduling policy execution through a scenario of crisis simulation example. During the initialization phase, the MAS components (physicians, medical porters and victims) are registered in the environment. For each registration of an agent, a description is generated by the environment process where the pairs $(id, value)$ and $(time, 0)$ are added to this description. The agent identifier is automatically added to the list $potentialAgents$ of the default filter. After this registration, each agent, following his behavior automaton, adds his filters in the environment. In this example, there are three physician agents P_1, P_2 and one victim V_1. The physician agent P_1 adds the filers $F_{victim\ detection}$ and $F_{diagnosis}$ in the environment ($Fdetect$ and $Fdiag$). These filters do not exist, so the environment process adds the associated filters and initializes the parameter $potentialAgents$ with the identifier of P_1 for each of them. The physician agent P_2 adds the same filters and the environment process updates the parameter $potentialAgents$ with the identifier of P_2. At this moment, P_1 and P_2 share the same filters.

Once the simulation environment is initialized, the simulation starts. The current simulation time is $t_E = 1$. The filters with the highest priority are evaluated first and they are triggered according to the matching with the descriptions in the environment. Let's consider the case where one physician agent P_1 sees the victim V_1 while the physician agent P_2 does not see a victim.

P_1 is activated by the filter $F_{victim\ detection}$ to perform the action *move in a direction*, the filter variable unification gives the description of the victim V_1 with the information about *position* and *id* which are necessary for the action execution. During the action execution by the agent P_1, his internal time is updated to 1.

P_1 has an internal time that is inferior to t_E and he is not activated. Consequently he is activated by the default filter to perform the default action *move randomly*. During the action execution, their internal time is updated to 1.

All the agents have their internal time superior or equal to t_E (Recall that t_E = 1). Consequently, the time update filter F_{time} is triggered and the simulation time t_E is incremented to 2.

5 Experiments

This section presents the assessment of the EASSmodel. Our objective is to compare the cost of the context computation by the environment versus the context computation by the agents. A prototype of our ABS framework has been implemented as a plugin for the multiagent platform Madkit [7]. The plugin is composed of an environment component with an API that enables the agents to add/retract/modify their descriptions and filters. We have executed the same agent behavior with a contextual activation and with a classical activation and compared the execution time. In the latter case, the scheduler activates the agents alternately and each of them computes locally his context. The results

show that our contextual activation is always more efficient than the classical activation. The first explanation of this result is the use of the RETE algorithm that enables to optimize the matching process of the shared filters. The second explanation is related to the context computation process itself. With a classical scheduler, the context computation is based on the "exploration" of the perceived environment while it is based on an event (environment modification) evaluation process in our solution. The consequence is that the cost of the matching process in our solution increases with the number of entities that are taken into account by the filters (figure 4). The cost increases following the number of entities that compute their context and the size of the perceived environment in the classical solution.

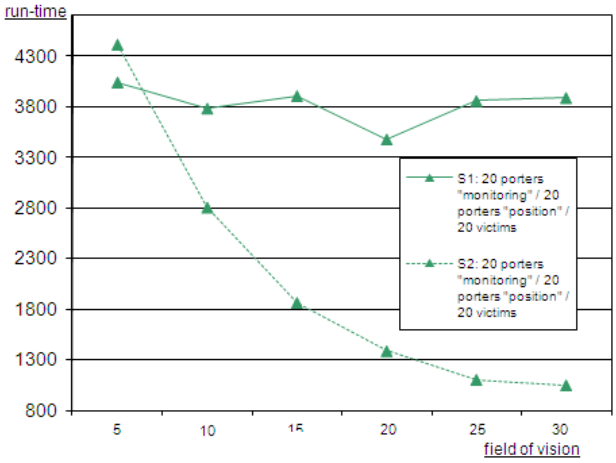

Fig. 4. Simulation run-time comparison between contextual activation (S_2) and classical activation (S_1)

6 Conclusion

In this paper, we discuss the role of the scheduling process in multiagent simulation. We have proposed a new model, called EASS (Environment as Active Support for Simulation), where the context computation is externalized to the environment and where agent activation is based on this context evaluation. Whereas classical approaches are agent-centered, our proposal is environment-centered because it delegates a part of the activity of the agents to the MAS environment. Our proposal is grounded on the PbC coordination principle [12] which argues in favor of MAS entities that have to be partially observable through a set of properties. For future work, we intend to test our platform on a complex, more

realistic and bigger scenario of crisis management in transportation networks (as in [13]).

References

1. López-Paredes, A., Edmonds, B., Klügl, F.: Special issue: Agent based simulation of complex social systems. Simulation **88**(1) (2012) 4–6
2. Cil, I., Mala, M.: A multi-agent architecture for modelling and simulation of small military unit combat in asymmetric warfare. Expert Syst. Appl. **37**(2) (March 2010) 1331–1343
3. Mandiau, R., Champion, A., Auberlet, J.M., Espié, S., Kolski, C.: Behaviour based on decision matrices for a coordination between agents in a urban traffic simulation. Applied Intelligence **28**(2) (2008) 121–138
4. Pěchouček, M., Mařík, V.: Industrial deployment of multi-agent technologies: review and selected case studies. Autonomous Agents and Multi-Agent Systems **17**(3) (2008) 397–431
5. Wagner, G.: Aor modelling and simulation: Towards a general architecture for agent-based discrete event simulation. In: Agent-Oriented Information Systems, Springer (2004) 174–188
6. Evertsz, R., Ritter, F.E., Busetta, P., Pedrotti, M., Bittner, J.L.: Cojack-achieving principled behaviour variation in a moderated cognitive architecture. In: Proceedings of the 17th conference on behavior representation in modeling and simulation. (2008) 80–89
7. Ferber, J., Gutknecht, O.: Madkit: A generic multi-agent platform. In: 4th International Conference on Autonomous Agents. (2000) 78–79
8. Kubera, Y., Mathieu, P., Picault, S.: Interaction-oriented agent simulations : From theory to implementation. In Ghallab, M., Spyropoulos, C., Fakotakis, N., Avouris, N., eds.: Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08), IOSPress (2008) 383–387
9. Collier, N.: Repast: An extensible framework for agent simulation. The University of Chicagos Social Science Research **36** (2003)
10. Boissier, O., Balbo, F., Badeig, F.: Controlling multi-party interaction within normative multi-agent organisations. In: Coordination, Organization, Institutions and Norms in Agent Systems VI (LNAI 6541), Springer-Verlag (2011)
11. Balbo, F., Zargayouna, M., Badeig, F.: A tree-based context model to optimize multiagent simulation. In: German Conference on Multiagent System Technologies, Springer (2014) 251–265
12. Zargayouna, M., Trassy, J.S., Balbo, F.: Property based coordination. In Euzenat, I.J., Domingue, J., eds.: Artificial Intelligence: Methodology, Systems, Applications. Volume 4183 of Lecture Notes in Artificial Intelligence. Springer Verlag (2006) 3–12
13. Badeig, F., Balbo, F., Scémama, G., Zargayouna, M.: Agent-based coordination model for designing transportation applications. In: Proc. of the 11th Int. IEEE Conf. on Intelligent Transportation Systems, IEEE Computer Society (2008) 402–407