Contextual activation for agent-based simulation

Fabien BADEIG[1,2] Flavien BALBO[1,2] Suzanne PINSON[2]

[1]INRETS - GRETIA team,
avenue du Général Malleret-Joinville,
F-94114 Arcueil Cedex, France
Email: fabien.badeig@inrets.fr

[2]University Paris-Dauphine - LAMSADE,
Place du Maréchal de Lattre de Tassigny,
F-75775 Paris 16 Cedex, France
Email: {balbo—pinson}@lamsade.dauphine.fr

Abstract—**When designing agent-based simulation, the choice of a coordination model is a key issue, since one of the difficulties is to link the activation of the agents with their context efficiently. Current solutions separate the activation phase from the action phase of the agents, and each action phase is based on local agent context analysis which is time-expensive. Moreover, because the link between the context and the action is an internal part of the agent, it is more difficult to modify the way the agent reacts to the context without altering the way the agent is implemented. Our proposal, called EASS (Environment as Active Support for Simulation), is a new approach for agent activation, where the context is analysed inside the environment and conditions the activation of the agents. The main result of contextual activation is to simplify the achievement of complex simulations and to decrease run-time. The EASS model has been implemented within the kernel of MadKit, a multi-agent platform, and the first results are given.**

Keywords—**agent-based simulation, environment, contextual activation**

Introduction

The general purpose of simulations is to model a complex reality [7] to reproduce, understand and evaluate it. The quality of a simulator is basically assessed using two criteria: how close it is to reality and how easy it is to interpret. The agent paradigm thus facilitates the modeling of independent entities interacting in an organized framework. This approach offers several interesting properties [8]: it supports structure preserving modeling of the simulated reality, simulation of proactive behavior, parallel computations, and dynamic simulation scenarios [4]. However, few models are based on the multi-agent paradigm because one of the difficulties in both the design and the understanding of MASs comes from the lack of central controls and the ensuing conflicting, uncertain, incomplete and delayed knowledge on the part of the agents.

One of the main problems in the design of a multi-agent simulation is the choice of a scheduling policy and more precisely the role of the simulator in the agent activation process. If the agents schedule their own behaviors, then the simulator has to synchronize them. This is well-adapted to simulations where agents do not have the same internal time models (discrete versus linear), or have different units of time (seconds versus years) [6], and a global virtual time has to be defined in order to enable coordination between agents. This constraint implies synchronization mechanisms that have to be considered during the modeling phase, and which limit the proactiveness of the agents. If the scheduler of the simulator manages agent activation, then the conflicts between agents have to be solved. Usually, the simulator applies one of the following scheduling policies: simple activation, double buffer and event-based [10]. In this paper, a new agent activation process where an agent is activated according to a context that it has chosen is proposed. Our proposal, called EASS (Environment as Active Support for Simulation), belongs to the last category of simulator and takes into account the different scheduling policies. The activation context of an agent can depend on a time cycle (simple activation) or on a specific event (event-based simulation). To take into account the double buffer approach, the model has to integrate a action conflict problem-solving engine.

The remainder of the paper is organized as follows. The next section shows the advantage of placing a medium, the *environment*, at the center of multi-agent simulation. This is followed by the presentation of an example that will be used throughout this paper to illustrate our proposal. The components of EASS, i.e. the simulation environment, the simulation agents and the simulation control, are then described and performance is assessed empirically. The papers concludes with general remarks and suggestions for future research.

Towards a contextual activation

In agent-based simulation platforms, the activation phase of the agents is separated from the action phase of the agents, and each action phase is based on a local agent context analysis. When there is a global scheduling policy, the standard process is to activate each agent at each time cycle, so that it computes its context locally in order to choose an action that it will perform. For example, in CORMAS [3] or MASON [9], for each agent the scheduler activates the same method that the multi-agent designer has to overload. In the *TurtleKit* simulation tool of MADKIT [5] largely inspired by Logo-based multi-agent platforms such as the StarLogo system (http://education.mit.edu/starlogo/) or NetLogo (http://ccl.northwestern.edu/netlogo/), an agent has an automaton that is applied by the scheduler. Each state of this automaton corresponds to an action.

Context computation involves retrieving and analysing the data. The problem is that computing the context locally increases the complexity of the agent simulation design. Consequently the simulation designer has only two ways of changing the parameters of the simulation. The first is to modify the global scheduler, for instance in CORMAS the "*prepare and schedule*" interface is used to give the parameters of the scheduler. The second is to modify the behavior of the agents, i.e. their reaction to their local context, which often implies a modification of the way the agent is implemented. Our objective is to explicitly link each agent activation to a specific context. The reification of the link between context and activation of the agent is called *contextual activation* and is managed by the environment.

This principle is similar to the contextual interaction proposed by the EASI (Environment as Active Support of Interaction) model [1]. In EASI, the environment is a medium enabling interactions to be shared. EASI modeling is a direct application of the Property-Based Coordination principle [13] (PBC) for interaction. The PBC is defined as a coordination principle for multi-agent systems in which every entity composing the system, including the agents, exhibits observable properties, and an agent uses the observable properties to manage the interactions, perceptions and actions inside the system. Our proposal is to apply the PBC principle for the activation process of the agents. The objective is that the agents dynamically choose their activation context (the local scheduling policy) and leave the management of the simulation system (the global scheduling policy) to the environment. Moreover, the environment is the temporal reference of the MAS, the agents managing their own internal time scale.

The result is an improvement of the control of the simulation. The environment ensures a centralized control in which the agents involved can dynamically modify their link to the simulation. By centralized control, we mean a global scheduling policy in which each agent modifies its own scheduling policy according to its state. By link, we mean the choice of the elementary action that the agent has to perform.

Illustrative example

This paper presents an agent-based simulation of robots, inspired by the application "Packet World" [12]. The objective of the robot agents is to cooperate in order to shift packets. The robot agents and the packets are situated on a *grid*, i.e. a two-dimensional space environment. A robot agent can move *randomly* or in *a given direction*, and it possesses only one skill. It can either *carry* or *raise* a packet, but the two skills are required to shift a packet. Each robot agent has a field of vision and the context of a robot agent is related to its field. A robot agent is close to a packet if the distance between the packet and the robot agent is equal to 1. A robot agent is unavailable if it has found a packet.

A robot agent reacts according to three contexts. The first is the context *packet seeking* where the robot agent randomly moves on the grid. The second is the context *packet discovery* where the robot agent has found a packet in its field of vision. Then, if there is no available robot agent with the same skill close to the packet, the robot agent moves towards the packet. The third is the context *packet proximity* where a robot agent waits for the arrival of another robot agent with the complementary skill to shift the packet.

A robot agent can interact to establish a contract with another robot agent to shift a packet. A robot agent which is committed to another robot agent by a contract is unavailable. The robot agents have a field of perception which depends on the distance between the robot agent and message. This property limits the circle of messages that can be received by the robot agent. A robot agent has a specific behavior for interaction according to two contexts. The first is the context *contact*, where a robot agent close to a packet tries to contact another available robot agent with the complementary skill to shift a packet and waits for it to arrive. The second is the context *contact reiteration*, where it reiterates the contact-making operation if no robot agent has answered.

Simulation system environment

This section focuses on the modeling of the environment. The environment ensures the global scheduling policy and is the temporal reference of the MAS. In order to apply the PBC, the environment must contain the descriptions of the components of the MAS.

Formal description

The environment is a set of m entities, $\Omega = \{\omega_1, \omega_2, \ldots, \omega_m\}$, and a set of k filters, $\mathcal{F} = \{f_1, f_2, \ldots, f_k\}$. An entity ω_l is the description of a component of the MAS (agent, object, percept) using observable properties. In the example, the set of entities Ω corresponds to the set of robot agent descriptions A, the set of message descriptions IO and the set of packet descriptions O. A filter f_j is the description of constraints on the observable properties of the entities. $\mathcal{P}$ is the set of observable properties, noted $\mathcal{P} = \{p_i \mid i \in I\}$. I is a finite set without assumptions on the data type (string, integer, etc). An observable property p_i is a function which gives the value associated with an entity ω_l, where $\forall p_i \in \mathcal{P}, p_i : \Omega \to d_i \cup \{unkown, null\}$. d_i, the domain description of the function p_i, is a finite set of qualitative or quantitative data. The value *unknown* is used when an agent does not want to give information about this property, and the value *null* is used when the property does not exist for the agent. The value of a property can be modified dynamically at run-time, except if it is *null*. A *null* value expresses the absence of a property, and this cannot be changed at run-time. Figure 1 shows a simple instantiation of our environment modeling. Here there are four entities, $\Omega = \{\omega_1, \omega_2, \omega_3, \omega_4\}$ that are respectively the descriptions of *packet 1*, *robot agents A* and *B* and message $MsgB_1$. The set of observable properties represented in figure 1 is not complete. For A, the set of observable properties is $\mathcal{P}_{\mathcal{A}} = \{id, skill, available, position, vision, perception, id_p\}$; for IO, this set is $\mathcal{P}_{\mathcal{IO}} = \{id, sender, receiver, type, id_p\}$; and for O, this set is $\mathcal{P}_{\mathcal{O}} = \{id, position\}$. For example, the description domain $d_{available}$ of the property *available* is $\{true, false\}$. With this property, an agent announces if it is processing a packet as in the case of the entity ω_3 where the value $available(\omega_3)$ is $false$.

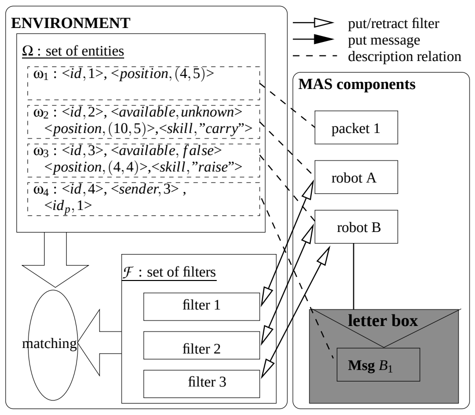
ENVIRONMENT
Ω : set of entities
ω₁ : <id,1>, <position,(4,5)>
ω₂ : <id,2>, <available,unknown> <position,(10,5)>,<skill,"carry">
ω₃ : <id,3>, <available,false> <position,(4,4)>,<skill,"raise">
ω₄ : <id,4>, <sender,3> , <id_p,1>
matching
ℱ : set of filters
filter 1
filter 2
filter 3
put/retract filter
put message
description relation
MAS components
packet 1
robot A
robot B
letter box
Msg B₁

Fig. 1. Overview of the environement

Here, the entity ω_1 does not have this property in its description, so the value $available(\omega_1)$ is *null*. In addition, the entity ω_2 does not wish to give its $available$ state, so the value $available(\omega_2)$ is *unknown*.

In the EASS model, two categories of filter exist with different uses. The filters that link the context to interactions are interaction filters, and the filters that manage the simulation process are simulation filters. The second category includes the activation filters that link the context to the activation of agents. In order to keep a homogeneous model for interaction and activation management, the definition of a filter is built using the Symbolic Data Analysis (SDA) theory [2] (for more details see [11]).

In SDA, a symbolic object is a coherent description of entities. An assertion is a special case of a symbolic object and is written $\forall \omega_l \in \Omega, as(\omega_l) = \wedge_{i=1,...,q}[p_i(\omega_l) R_i v_i]$ where R_i is a comparison operator and v_i is either a variable, or the value *unknown* or a value belonging to d_i. The value of an assertion for any particular entity $\omega \in \Omega$ is *true* if that assertion holds for that entity, or *false* if not. An example of an assertion is $as(\omega) = [skill(\omega) = "carry"] \wedge [available(\omega) = true]$; this assertion is the description of the *available* entities which have the skill *carry*.

A filter f is a symbolic object. For agent a and context $C \subset \Omega$, an activation filter is $f(a, C) = \wedge_{i \in I_a}[p_i(a) R_i v_i] \wedge_{c \in C} (\wedge_{i \in I_c}[p_i(c) R_i v_i])$. $I_a \subset I$ (respectively I_c) are the indices ranging over $\mathcal{P}$ that are used in f for selecting the agent (respectively the context). R_i and d_i are respectively the comparison operators and the values of the descriptions that define the conditions to be held by a and C. An activation filter is triggered when the conditions on the description of agent a and context C are verified. For agent a, the interaction object $io \in IO$ and the context $C \subset \Omega$, an interaction filter is $f(a, io, C) = \wedge_{i \in I_a}[p_i(a) R_i v_i] \wedge_{i \in I_{io}} [p_i(io) R_i v_i] \wedge_{c \in C} (\wedge_{i \in I_c}[p_i(c) R_i v_i])$. $I_{io} \subset I$ are the indices ranging over $\mathcal{P}$ that are used in f for selecting the interaction object. This filter is the conjunction of at least two conditions, the first being related to the receptor a (first argument) and the second being related to the interaction object io (second argument). The third argument that describes the context is optional. This definition means that the same interaction object io can be perceived according to different contexts, or on the contrary that several interaction objects io can be perceived in the same context. Contrary to the activation filter, the trigger of an interaction filter depends on an interaction object io (here, io is a message). However, in the activation case, if the agent activation is conditioned by an interaction object io, then the conditions of the io description are added to the context C description of the filter. In our example, the interaction filter that enables the reception of a $request$ message is: $f_{request1}(a, m, \{a_1\}) = [available(a) = true] \wedge [sender(m) = ?x_1] \wedge [id(a_1) = ?x_1] \wedge [skill(a) = ?x_2] \wedge [skill(a_1) \neq ?x_2] \wedge [type(m) = request] \wedge [position(a) = ?x_3] \wedge [position(a_1) = ?x_4] \wedge [perception(a) >= pcc(?x_3, ?x_4)]$ where pcc is a function which returns the value of the shortest path between two points. This filter ensures that an agent a receives a $request$ message m if a is available with the skill which completes the skill of the sender agent a_1 and if the distance between a and a_1 is less than the perception field of the agent a, i.e. a can perceive the message.

One of the main advantages of using the same formal definition for all filters is that interaction and activation modeling is homogeneous, thus giving a simple way to describe the simulation process, as is shown in the next section which discusses filter-activated actions.

Environment specifications

The set of filters $\mathcal{F}$ contains the interaction and simulation filters that are inside the environment. When a filter is triggered, this activates an action for an agent to perform. This action depends on the filter category. In the case of interactions, this means that an agent can perceive an interaction object. This is ensured by the action *receive* that each agent must implement to receive interaction objects. It adds the interaction object with the information about its context to the "message box" of the agent. The objective of the activation filter is to trigger the appropriate reaction of the agent according to its context, which is why an action *act* is associated with each filter. This action is implemented by all the simulation agents and its parameters are the context and a label corresponding to the action associated to this context (see paragraph on Simulation Agents).

In addition, in the EASS model the environment ensures the simulation validation process including the time management. Consequently, the environment is the temporal reference for the simulation agents and each agent has its internal time. The environment time is discrete and is the global time of the simulation. At a given time cycle, the simulator ensures that the agents which are ready to act are activated; this is done only once per time cycle. This control is guaranteed by the environment thanks to the comparison of the global time with the internal time of the agent in each activation filter. Therefore, the internal time of agents must be observable, which implies the addition of the observable property *time* to the description of the agent. When the value of the observable property *time* is greater than the time of the environment then the agent is not activated. This control implies that the internal time of an agent is updated automatically when it is activated.

The activation of the agents depends on the filters inside the environment. In order to ensure a default activation, the filter $f_{default}$ is systematically added to the environment and its triggering depends on the comparison between the global time (t_E) and the internal time of the agent. This filter is written $f_{default}(a, \emptyset) = [time(a) < t_E]$ where $a \in A$.

In order to update the environment time, the filter f_{time} compares the internal time of the agents with the global time (t_E) and is written $f_{time}(null, A) = \wedge_{a \in A}[time(a) >= t_E]$. It is triggered when the internal time of all the agents is greater than or equal to the global time. This triggering process updates the global time t to $t + 1$. This filter is a simulation filter.

This modeling offers two advantages: (1) an agent can choose to be inactive for a period in the simulation, thus enabling a gain in run-time; (2) the same agent cannot be activated more than once in the same time cycle.

Simulation agents

The activity of an agent is modeled using an automaton, called the behavior automaton (Figure 3). Each state of the automaton is a reference to a behavior which is a coherent sequence of actions. If the behavior is linked to one action, it is *elementary*, whereas if it is associated with several actions, it is *complex*. The transition from one state to another corresponds to the context. The context is perceived via the filters for the contextual activation and can also be identified through local processing which consists in recovering a partial state of the environment. Our model manages these two processes in the same simulation. Each agent has its own scheduler which manages the behavior automaton which specifies the next action to be performed. The activation filters use only one generic action *act* for the agent activation, and this action corresponds to the call of the agent scheduler. In order for the right behavior to be activated, a *label* is associated with each behavior of the agent and is a parameter of the action *act*. The definition of a *label* is a couple $(id, context)$ where *id* is the identifier of the *label* and *context* is the set of information related to the context. Thus the agent knows which behavior has to be carried out as well as information related to the activation context.

To build the behavior automaton of each agent, the designer first has to identify the activation contexts of each behavior, and then build the corresponding activation filters for each context. Each agent has a set of activation filters. A filter becomes active when the agent adds it to the environment, and is desactivated when the agent retracts it. The agent dynamically chooses the activation filters that it adds to the environment; each additional activation filter triggers a particular ac-

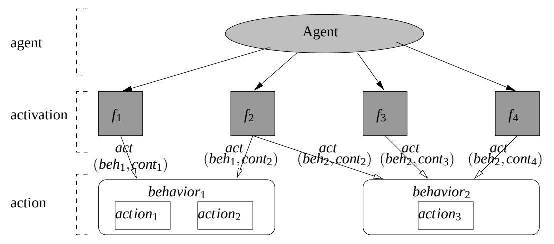

Fig. 2. Relation between agent behaviors and activation filters

tivation context for the agent. In this way of modeling the activation process, we put context analysis into the environment. The same behavior can be activated from various filters and thus in various contexts. Conversely, the same filter can be used for several behaviors, enabling an agent to modify its reaction to the same context. This approach facilitates the implementation of various scenarios. Figure 2 depicts the relation between the agent behaviors and its activation filters.

When trying to take the consequence of the execution of a complex behavior by an agent into account, the difficulty depends on whether or not it is possible to interrupt it. If a complex behavior can be interrupted, the link between the agent and its environment is not modified because it is the standard execution of the simulator, and the agent is in charge of the correct execution of its behavior. If a complex behavior cannot be interrupted, the agent uses its observable property *time* to simulate an appointment. Thus, if *time* is equal to the sum of current time and duration of the complex behavior, then no filter (contextual activation or $f_{default}$) can be triggered. Therefore it is necessary to add a new filter f_{free} that activates the agent to keep the execution of the complex behavior at each time cycle. In order to take this characteristics into account in the agent model, we have added an observable property *complex* to the agent description. The filter is $f_{free}(a) = [complex(a) = true] \wedge [time(a) >= t_E]$ with $a \in A$.

Simulation control

As in the interaction model EASI [11], the filters belong to the set $\mathcal{F}$. However, as described above, there are more than one kind of filter in the EASS model. Some of these filters are necessary for interaction, others for activation. To design a simulation using EASS, the designer has to schedule when each filter will be evaluated. From our point of view there are two scheduling levels: the global scheduling level which controls the execution of the simulation, and the local scheduling level which controls the behavior of an agent.

At the global scheduling level, the problem is that the agents must be activated under the same simulation state. Our proposal is to control execution of the simulation by an execution automaton where a state is a package of filters. A package groups filters that can be triggered at the same cycle of the simulation. By default, the execution automaton is composed of

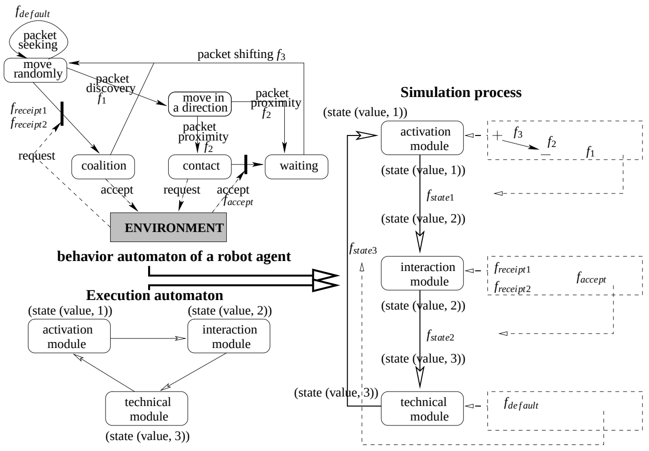

Fig. 3. Agent behavior automaton and simulation process

two states: one state is related to the "technical" management of the simulation, the other to the activation of the agents. For example, f_{time} is triggered when the simulation is in the "technical" state. The advantage of grouping the interaction filters in a specific state of the automaton is that the model of activation and interaction is homogeneous with its automaton which defines the order of filter execution. The transitions between states of the execution automaton are triggered by the value of a variable *state*. The value of this variable is managed by a filter f_{state} that is designed according to the needs of the designer.

At the local scheduling level, the problem is that several filters could be triggered in the same time cycle for the same agent. Our proposal is to control the behavior of an agent by a priority level given to each filter. Let *priority* be an application that gives the importance of a filter, where $priority : \mathcal{F} \to \mathbb{N}$. The filter with the highest priority is evaluated before a filter with a lower priority in the same filter package. For an agent, the trigger of the activation filter with the highest priority desactivates the others, using the update of the value given by the property *time*. If an agent has no activation filter that has been triggered at the current time cycle, then the filter $f_{default}$ is used. This filter belongs to the activation state and has the lowest priority. It is thus triggered by default for an agent only if its internal time has not been updated.

We have two levels of simulation control: (1) control of the simulation process using the execution automaton; (2) control of the activation of the agents using the priorities of the filters. These two levels of control define a new way to control the behavior of a simulation system. For our example, figure 3 depicts the relation between the behavior automaton and the execution automaton.

Results with the example "Packet World"

The simulation shows in figure 3 was assessed using six scenarios as shown below. Figure 3 gives the behavior automaton of robot agents. A filter is associated with each activation context, but the behavior triggered by a filter may change according to the scenario. For example, using the filter f_2 related to the context *packet proximity*, the associated behavior is the *waiting behavior* for the scenarios without any communication (S1, S2) and the *contact behavior* for the scenarios with communication (S3, S4, S5, S6). The filter $f_{default}$ is related to the behavior *randomly move*, and corresponds to the context *packet seeking*. The filter f_1 is related to the context *packet discovery* and enables a robot to move towards the closest detected packet if no available robot with the same skill is close to the packet. This filter is written $f_1(a, e \in O) = [position(e) = ?x_1] \wedge [position(a) = ?x_2] \wedge [vision(a) >= pcc(?x_1, ?x_2)] \wedge_{px \in O} ([position(px) = ?x_3] \wedge [pcc(?x_1, ?x_2) <= pcc(?x_3, ?x_2)]) \wedge_{a1x \in E(A_1)} ([position(a1x) = ?x_4] \wedge [pcc(?x_4, ?x_1) \neq 1]) \wedge [time(a) < t_E]$ where $E(A_1)$ is the set of entities $\omega \in A$ which satisfy $[skill(a) = skill(\omega)] \wedge [id_{packet}(\omega) = id(e)]$. This filter activates an agent a which detects a packet e in its field of vision and this packet is the closest one of the agent a. The filter f_2 related to the context *packet proximity* is written $f_2(a, e) = [id_p(a) = id(e)] \wedge [position(a) = ?x_1] \wedge [position(e) = ?x_2] \wedge [pcc(?x_1, ?x_2) = 1] \wedge [time(a) < t_E]$. This filter is triggered when the distance between an agent a and a packet e is equal to 1. The filter f_3 is related to the context *packet shifting* and triggers the moment when the two agents shift the packet.

The interactions are ensured by the filters $f_{receipt1}$, $f_{receipt2}$ and f_{accept}. The filter $f_{receipt2}$ is a specialization of $f_{receipt1}$ where the receiver is the agent nearest to the sender. The filter f_{accept} is triggered when an agent adds an *accept* message inside the environment in response to a *request* message. With $f_{receipt2}$ and f_{accept}, our interaction protocol is similar to a Contract Net Protocol; the filter $f_{receipt2}$ is used to find the best receiver and the filter f_{accept} closes the "negotiation".

For the evaluation of the model, we consider six scenarios:

S1: $\{f_1\}$ + Local Agent Context Analysis (LACA),
S2: $\{f_1, f_2, f_3\}$,
S3: $\{f_1\}$ + LACA + $\{f_{receipt1}, f_{accept}\}$,
S4: $\{f_1, f_2, f_3, f_4, f_5\}$ + $\{f_{receipt1}, f_{accept}\}$,
S5: $\{f_1\}$ + LACA + $\{f_{receipt2}, f_{accept}\}$,
S6: $\{f_1, f_2, f_3, f_4, f_5\}$ + $\{f_{receipt2}, f_{accept}\}$

S1 and S2 show the difference between a scenario with an activation phase and a local context analysis (S1), and a contextual activation inside the environment (S2). The scenarios S3 and S5 are the extensions of scenario S1 with communication. S4 and S6 illustrate the unified modeling of the contextual activation and the interactions. We have chosen two different interaction policies in order to show the facility with which one can generate different simulations without modifying the architecture of the agents.

We have run various simulations according to the number of robot agents, the number of packets and the value of the field of vision on a *grid* (150 × 150) and have evaluated our model in terms of run-time.

On average, S2 is faster than S1. A larger field of vi-

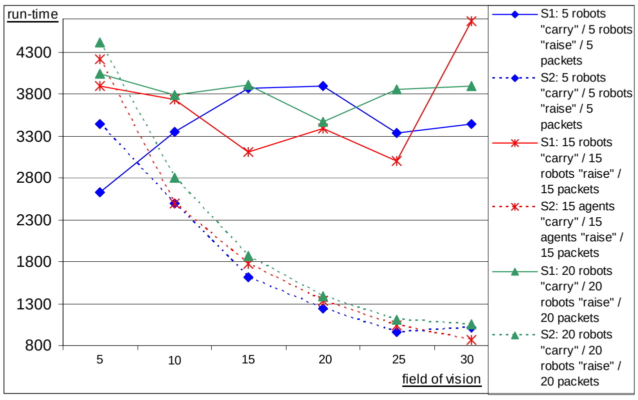

Fig. 4. Execution time with strategies S_1 and S_2

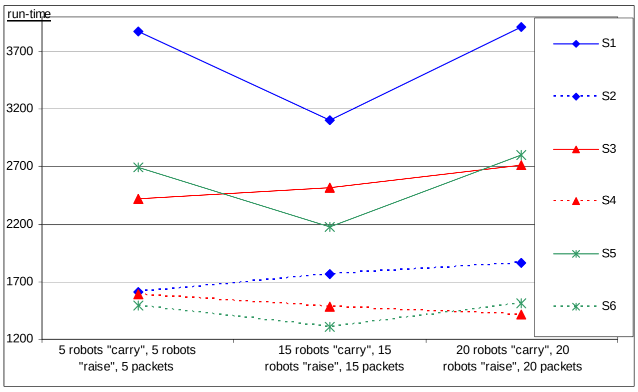

Fig. 5. Run-time with a field of vision = 15

sion improves the efficiency of the robot agents for S2. For S1, the same improvement barely offsets the cost of the local context analysis which increases linearly according to the size of the field of vision (figure 4). In addition, the difference in run-time between the scenarios with or without contextual activation is smaller than with the scenarios with communication (figure 5). When a coalition is formed the robot agent checks its internal state and moves directly towards the related packet without local context analysis. This is a direct call of one behavior, and the cost is close to that of behavior activation by a filter.

Conclusion

This paper has presented a new contextual activation processing for agent-based simulation. Our proposal is based on the Property-Based Coordination principle which argues that the components of a multi-agent system have to be observable through a set of properties. The main objective is to make it easier and more efficient to share their context. As far as simulation, there are theoretical and practical advantages. From a theoretical viewpoint, the same unifying framework proposed for interaction and activation simplifies the development of complex multi-agent simulations. From a practical viewpoint, because the environment processes some of the tasks usually performed by the agents, the design of the agents is made easier, thus improving the run-time of the simulation. Moreover, our model proposes a new way to parameterize the model through the dynamic management of the filters, i.e. the link between the simulation process and the agent behavior. All changes are made without modifying the way the agents are implemented. Not only is the interpretation of the result facilitated but, in addition, formalization using concepts from Symbolic Data Analysis gives powerful tools to design and analyze the results of the simulation. Our proposal has been implemented and tested, the first results are encouraging. This evaluation needs to be improved using a more complex example including both cognitive and reactive agents, like the crisis management.

References

[1] F. Balbo. A new interaction model for agent-based simulation. In *18th European Simulation Multiconference*, 2004.

[2] L. Billard and E. Diday. *Symbolic Data Analysis: Conceptual Statistics and Data Mining (Wiley Series in Computational Statistics)*. John Wiley & Sons, 2007.

[3] F. Bousquet, I. Bakam, H. Proton, and C. Le Page. Cormas: Common-pool resources and multi-agent systems. In *IEA/AIE*, pages 826–837, 1998.

[4] P. Davidsson. Multi-agent-based simulation: Beyond social simulation. In *MABS*, pages 97–107, 2000.

[5] J. Ferber and O. Gutknecht. Madkit: A generic multi-agent platform. In *4th International Conference on Autonomous Agents*, pages 78–79, 2000.

[6] E. Fianyo, J-P. Treuil, E. Perrier, and Y. Demazeau. Multi-agent architecture integrating heterogeneous models of dynamical processes: The representation of time. In *MABS*, pages 226–236, 1998.

[7] D. Hill. *Object-Oriented Analysis and Simulation*. Addison-Wesley, 1996.

[8] N. Jennings and M. Wooldridge. Applications of intelligent agents. In *Agent Technology: Foundations, Applications and Markets*, pages 3–28, 1998.

[9] S. Luke, C. Cioffi-Revilla, L. Panait, and K. Sullivan. Mason: A new multi-agent simulation toolkit. In *SwarmFest Workshop*, 2004.

[10] F. Michel, G. Beurier, and J. Ferber. The turtlekit simulation platform: Application to multi-level emergence. In *Agent Based Simulation*, 2003.

[11] J. Saunier, F. Balbo, and F. Badeig. Environment as active support of interaction. In *E4MAS*, pages 61–78, 2006.

[12] D. Weyns, A. Helleboogh, and T. Holvoet. The packet-world: A test bed for investigating situated multiagent systems, 2005. Whitestein Series in Software Agent Technology.

[13] M. Zargayouna, J. Saunier, and F. Balbo. Property based coordination. In *Artificial Intelligence: Methodology, Systems, Applications*, pages 3–12, 2006.

Author Biographies

Fabien BADEIG is a Ph.D. student at the National Institute for Transport and Safety Research (INRETS) and at the LAMSADE laboratory of the University of Paris-Dauphine. His research concerns agent-based simulation applied to dynamic contexts such as crisis management in transport.

Flavien BALBO is Assistant Professor in Computer Science at the University of Paris-Dauphine. He received an M.S. in Applied Mathematics and a Ph.D. in Computer Science from the University of Paris-Dauphine. He is an associate researcher at INRETS. His primary research focuses on Real-Time Decision-Making Systems. His other research interests include Multi-Agent Systems and, in particular, interaction.

Suzanne PINSON is Professor of Computer Science at the University of Paris-Dauphine. She is Head of the Research Group on Artificial Intelligence and Decision Processes at the CNRS-LAMSADE laboratory. She received a M.S. and a Ph.D. in Computer Science from the University of Paris 6 as well as an M.S. in Computer Science and Operational Research from Northwestern University, Evanston, Il. Her research areas include Distributed Decision-Making and Multi-Agent Systems, more precisely, coordination and cooperation issues.