Optimizing context computation for multiagent simulations

Flavien Balbo[1], Mahdi Zargayouna[2], Fabien Badeig[1]
[1]Institut Henri Fayol, ENS Mines Saint-Etienne [2] Université Paris-Est, IFSTTAR, GRETTIA.
Saint-Etienne, France. Champs sur Marne, France.
{flavien.balbo, fabien.badeig}@mines-stetienne.fr hamza-mahdi.zargayouna@ifsttar.fr

***Abstract*—The execution of a multiagent-based simulation (MABS) model necessitates a scheduler that synchronizes the agents execution and simulates the simultaneity of their behaviors. In the majority of MABS frameworks the scheduler activates the agents who compute their context to decide the action to execute. This context computation process is time-consuming and is one of the barriers to increased use of MABS for large simulations. The context of an agent is computed based on all information he possesses about himself, the other agents and the objects of the environment that are accessible to him. One of the issues is to find this information and to identify the information subsets that make sense for the agent. In the majority of MABS, the context computation is hidden in the agent processes and there is no specific algorithm enabling to decrease its computing. Our proposal is the modeling of this subset of information identifying a context by a so called "filter" and an algorithm to find efficiently for each agent all their filters. The accessible information are given by the scheduler to the agents who browse a tree of filters to select the right information. With these filters, the agents know their context and decide which action to execute. Our algorithm is compared to a classical context identification. Promising results are presented and discussed.**

***Keywords*—Multiagent simulation, multiagent scheduling, context computation, agent modeling**

I. Introduction

One of the main functional objectives of a simulation is the reproduction of complex systems, some specific properties must therefore be guaranteed. The simultaneity of actions, which implies that several agents are activated at the same simulated time, is one of these properties. Therefore, similarly to multitasking on a computer, the execution of a MABS model necessitates a scheduling process (executed by a scheduler) that synchronizes the agents execution and simulates the simultaneity of their behaviors. The majority of the MABS frameworks follow a cooperative model, where the activation of agents is controlled by a scheduler and their interruption is controlled by the agents themselves. When activated by the scheduler, the agent executes his current behavior and decides to hand over to the scheduler, either because he has finished what he has planned to do or because he has been designed to halt at a certain step. The scheduler then activates another agent. Following a discrete temporal model, the current time value within MABS is updated when agents have terminated their current action. The activated agent has to compute his context to decide which action to execute and this computation is time-consuming.

The context of an agent is computed with all the information that are accessible by him including his local perception of the environment i.e. the other components of the MABS [4]. This computation can be divided in two steps. The first step is related to the identification of the perceptible information which cost depends of the organization of this information (grid, list, etc.). The second part is the identification of the subsets of information that are relevant for him. The definition of these subsets, that we call context, belongs to the knowledge of the agents because they condition their behavior. The issue is that the identification of the relevant context is embedded in the agent code: The agent executes an often costly (because of the often large size of the search space) ***if*** *context* ***then*** *action* evaluation process. To decrease this computation cost, the designer classically uses nested contexts and/or behavioral automata (NetLogo like MABS). From our point of view, nested contexts increase the complexity of the agent design and behavioral automata focus on the internal state of the agent, neglecting the other components of the agent context. Our proposal is the modeling of the contexts as "filters" to simplify the agent design without limiting the context computation possibilities. Each agent has his own filters and executes the algorithm that we propose in this paper to find all the filters related to the accessible information.

The remainder of the paper is organized as follows. Section II discusses the issues related to context computation. Section III presents an illustrative example that we follow all along this paper. Section IV provides the formal definition of our proposal. Our context selection algorithm is provided in section V. Section VI presents our experimentation and results. The paper concludes with a discussion and some perspectives to this work.

II. State of the art

The context computation issue is not only related to the multiagent based frameworks. Every system in which the components act following filtered information coming from their environment has to deal with these issues. In Distributed virtual environments (DVE), context computation is a major issue and [9] has proposed a classification of the different solutions. Indeed, DVE provide sensory information to multiple users allowing them to interact in a shared environment even if they are situated in different locations. [9] proposes the following classification of the data filtering schemes:

- Zone-based solution: the environment is decomposed in zones and each of them contains a subset of information. This space information becomes therefore the main filter to find the relevant information. Moreover

it is possible to limit for a component his access to the information.

- Aura-based solution: an aura represents the interests of the participants. When the aura overlaps, then the participants exchange information. It is a qualitative improvement of the zone-based solution: because the information is filtered following the participant's interests. This solution is time-consuming because the aura of all the participants have to be compared.
- Class-based solution: with this solution, the participants subscribe to a subset of components that are relevant for them. Therefore, the information filtering is done following the specific relation between the component and is not limited to a spatial relation.
- Hybrid solution: this is a combination of the two preceding solutions.

These solutions enable to compute which data should be sent to the participants and which data should be filtered. The participants compute their context based on these data. For instance in [7], in an aura-based teleconferencing system, a user receives data from the others if they use compatible medium, are close enough and he is interested by them. In this case, the user receives verified relevant data but he still has to compute his context (which users, what medium, etc.), which is a combination of this data. MABS has to take into account that the participants are artificial agents which have to compute their context following the accessible data. When an agent is activated, he is aware that he is executing a new simulation step. He can therefore compute his new context before to decide which action to execute. In this section, we discuss which information are provided by the scheduler to the agent when activated in a cooperative model. These information are used by the agent to choose which action to execute next. There are three options: 1) no information is given; 2) a reference to the action to perform; 3) some information about the simulation's context.

In the first option, the scheduler activates the agents either by calling a default method ([2], [10], [14], [1]) or with a control message ([13], [15], [12], [5]). The activated agent must then computes its context thanks to perceptible information. For instance in [1] the objects belonging to the activated agent perception field are given to him thanks to a perception event.

The second option gives a reference to the next action of the agent and assumes that the scheduler knows which behavior of the agent has to be activated. In [11], the scheduler activates the agents following an ordered list of actions. In the logo-based multiagent platforms such as the TurtleKit simulation tool of MADKIT [6] or STARLOGO[1], an agent has an automaton that determines the next action that should be executed. The scheduler activates the agents based on this information that is a predetermined context.

With the last option, the scheduler also computes for each agent which action to execute, but this computation is based on the current simulation and agent's context. To the best of our knowledge, the framework JEDI [8] and the Repast Simphony simulation platform [3] are the only two proposals where the choice of the action that is executed by an agent is computed by the scheduler. In the JEDI framework [8], the computation of the agent action decision is based on an interaction matrix where a cell is a conditioned interaction between two simulation components. 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 does not change during the simulation. This approach is based on a reification of the interaction and its computation can therefore be externalized. The advantage is that it can be optimized and simply modified contrary to an agent-based approach. Nevertheless, 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's dimension. Moreover this matrix cannot be modified by the agents and therefore the agent autonomy is limited. Repast Simphony natively uses the first scheduling options (i.e. with a default method 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 limited by the expressiveness of the watcher queries language to express the activation context. The queries are boolean expressions that evaluate the watcher and the watchee using primitives such as *colocated*, *linked_to* [*network name*], *within X* [*network name*], etc. (a network is a graph of agents relationships) and the logic operators AND and OR. It is not possible to integrate complex conditions about other components (other than the watcher and the watchee).

We propose a multiagent-based simulation process where the environment (our scheduler) activates the agents with their accessible information. We propose a data model and an adapted algorithm to compute agent' context. A context is modeled by condition on meta-information about the MAS components given by the environment. This standardized model of a context is called a filter. The activated agent processes the computation context thanks to our algorithm on its own standardized context model (his filters).

III. ILLUSTRATIVE EXAMPLE

To illustrate our proposal, we choose to simulate the behavior of a driver who enters a roundabout. Our objective is to highlight the richness of the context modeling for a complex simulation. The roundabout simulation is used to illustrate the components of our proposal while our experiments are based on a theoretical example (section VI). Indeed, our experimentation takes into account the efficiency of our proposal and we have to control all the parameters. The figure 1 represents a roundabout with agents (vehicles, pedestrians and bicycles) and objects (traffic signs).

The context of a vehicle agent is computed with the information that are perceptible by him: objects and agents in his perception field, his GPS device and his own state. Each of these sources gives numerous information that have to be combined to give to the vehicle agent (VA) a global

[1] http://education.mit.edu/StarLogo/

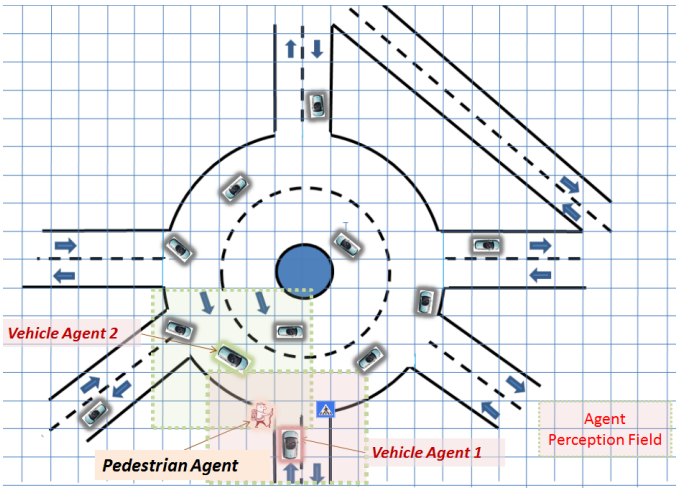

Fig. 1. Crossroad simulation example

status of his own situation. These sources give complementary information. For instance, the perception field gives the relative location of the mobile entities (vehicles, pedestrians, bicycles, etc.), the GPS device gives information about the network topology and the traffic flows and the state of the agent gives his direction, speed, the state of his turn signals, etc. A context is a combination of these perceptible information that are relevant for the agent. For instance, the pedestrian agent pa_1 is perceived by va_1 and va_2 (figure 1) but the resulting context is not the same for each agent: 1) va_1 context could be *"my speed is excessive and there is pa_1 crossing the street before me"*; 2) va_2 context could be *"I am about to cross entering traffic, which is blocked by the crossing pa_1"*. The information is the same but that is their combination that makes sense for agents. The issue is that these combinations are multiple and their computation is time consuming. Their definition are related to the expert domain as their use in the decision. After the identification of the relevant contexts, the agents have to decide which action to execute. Our proposal is positioned between the information acquisition and the decision process: a data model and an algorithm for information context processing.

IV. Definition of the Model

A. Data model

Our main objective is to decrease the execution time cost of a simulation. Classically, an agent executes a sequence at each simulation cycle: *perception - decision - action*. The result of the perception step is the computation of information about the perceptible MAS components (agents, objects, etc.). How this computation is processed is independent of our proposal since we only take into account the result of this process. Based on these information, the agent has to select those that are relevant. We assume that the scheduler provides to each agent a set of $\langle property, value \rangle$ pairs for each perceptible entity. An entity is an agent or an object and a description is the list of pairs characterizing the entity (a formal definition is given below). Entities with identical properties belong to the same *category*.

The figure 2 gives two alternatives for the recording of descriptions. In the first case, the descriptions are stored in a 2D grid and each cell may contain a component of the environment. In this case, an agent has to access each cell that is accessible to him then collects the available descriptions. In a 2D grid, the descriptions are ranged following their position and their category has to be found. The complexity to construct the description list is $O(C*n^2)$ with C the number of categories and n the size of the part of the 2D grid that is perceived. The second alternative consists in organizing the descriptions by category and at each cycle find the descriptions that are accessible to the activated agent. The complexity to construct the description list is $O(C*m)$ with C the number of categories and m the average cardinality of a category. There exists other alternatives and we assume that our algorithm uses a unique data structure: a list of categories for which there exists at least one perceptible description. This assumption is critical but is not difficult to respect since this is a quite basic model to represent the information.

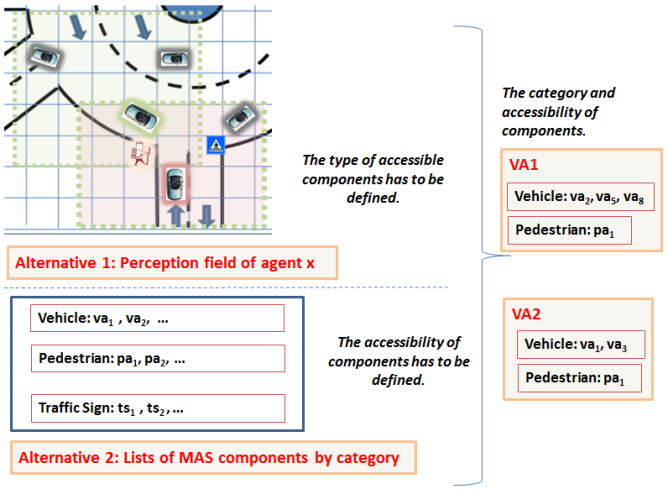

Fig. 2. Alternatives for simulation data organization

The elementary component of our proposal is called a property and gives a specific information about an entity.

Definition 1 (Property): A property $p_i \in \mathcal{P}$ is a function, which description domain $d_j \in \mathcal{D}$ is quantitative, qualitative or a finite set of data. A property is noted $p_i : \Omega \rightarrow d_j$, with Ω the set of descriptions.

For instance, the property $speed : \Omega \rightarrow \mathbb{R}$ gives the speed of an entity or the property $street : \Omega \rightarrow \{$ *streets*, *roundabout lanes*$\}$ gives the location of the entity following the network's topology.

Definition 2 (Description): A description is a set of pairs $\langle property, value \rangle$ that characterizes a MAS entity (agent or object).

For instance, the description of a vehicle agent could be (we note $\Omega_{\mathcal{A}} \subset \Omega$ the set of agents):

- $speed : \Omega_{\mathcal{A}} \rightarrow \mathbb{R}$;
- $location : \Omega_{\mathcal{A}} \rightarrow \mathbb{N}$: the distance from roundabout entry or a relative value for a roundabout lane;
- $direction : \Omega_{\mathcal{A}} \rightarrow \{$ *to roundabout, from roundabout*$\}$;
- $turnSignal : \Omega_{\mathcal{A}} \rightarrow \{left, right, off\}$: gives the state of the turn signals

- ...

Definition 3 (Category): A Category represents the entities that have an identical set of properties and that are semantically similar.

In our example, vehicle agents (VA), pedestrian agents (PA) and traffic signs (*TS*) are instances of categories.

We propose to model a context as a filter that formalizes conditions on the entity's descriptions that are perceived by the agent. A filter generates processed information from raw information (description of the MAS components) and is by construction an aggregate information.

Definition 4 (Filter): A filter $F_j \in \mathcal{F}$ is a tuple $F_j = \langle f_a, f_C, n_f \rangle$ with:

- $f_a : \Omega_{\mathcal{A}} \rightarrow \{true, false\}$ a mandatory assertion that expresses constraints on the agent who owns the filter;
- $f_C : 2^{\Omega} \rightarrow \{true, false\}$ an optional set of assertions expressing constraints about others components that complete the context;
- n_f the filter name.

A filter identifies by unification the agent's description and the context (a subset of descriptions) that matches the associated assertions. A filter is valid for any tuple $\langle a \in \Omega_{\mathcal{A}} \subset \Omega, context \subset \Omega \rangle$ such that $f_a(agent) \wedge f_C(context)$ is evaluated to true. When a filter is valid, the associated context is valid for the agent a, i.e. the tuple $\langle context, n_f \rangle$. A context being formalized as constraints on the descriptions of the MAS components, the *context* and n_f information are complementary to characterize the MAS context. It means that the same description's subset can valid several contexts and a context can be validated by several description's subsets.

Let $\langle f_a, f_C, warning \rangle$ be a filter dealing with the detection of a warning related to the potential movement of vehicles. A filter belongs to an agent and is therefore built from his point of view. For the *warning* filter, the vehicle agent is on the central lane of the roundabout and a slower vehicle agent before him in the other lane turns on his left turn signal. The filter triggering depends on: i) the location of the agent (assertion f_a), ii) the perception of another agent with a perceptible property (assertion f_C). The filter *warning* has the following definition:

- $a \in \Omega_{\mathcal{A}}$: $f_a : [speed(a) = ?s_a] \wedge [street(a) = centralLane] \wedge [location(a) = ?l_a]$
- $b \in \Omega_{\mathcal{A}}$: $f_C(b) : [speed(b) < ?s_a] \wedge [location(b) < ?l_a + 2] \wedge [turnSignal(b) = left] \wedge [street(b) = externLane]$

The symbol "¿' before an expression identifies a variable and the operator "=" is the comparison operator. With this filter, the agent a, when he is in the central lane, is interested by the b slower agents who are in the external lane and that are up to two units before him, if their left turn signal is on. The scheduler has already filtered the perceptible entities based on the perception field of the agent a. The agents' filters concern locations only if they have additional space constraints other than the perception field, like in this example.

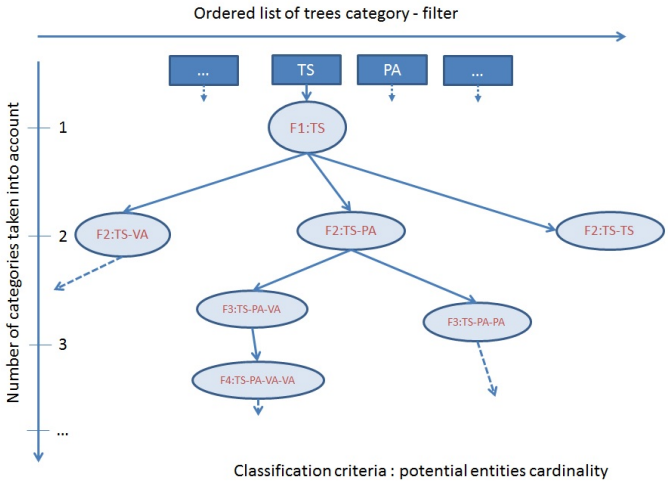

Fig. 3. ordered list of filters

B. Data structure

We propose to organize the agents filters as a list of trees of filters. Figure 3 is a part of a list of tree of filters for a vehicle agent. An agent has his own instance of the proposed data structure and he processes an algorithm (described later in this paper) to browse it and find the filters that match the accessible descriptions. The objective of this structure is to test fewer filters given the number of categories that are perceived by the agent. The basic idea is the following: *The evaluation of a filter is conditioned by the existence of at least one entity for each category that it has to test.*

A tree of filters is identified by the name of the category that is tested by these filters having the minimum of entities. We assume that the number of entities by category is defined by the MABS designer or he is at least able to rank the categories. In our roundabout example, the designer gives the following order $\cdots < |TS| < |PA| < |VA| < \ldots$ if the simulation concerns rush hours with a great traffic activity, or $\cdots < |PA| < |VA| < |TS| < \ldots$ if the simulation concerns night time with low traffic.

The list of filters' trees is ranked following the order given by the MABS designer. For instance, following the rush-hour order, there are more vehicle agents (category *VA*) than traffic signs (category *TS*). Therefore the first element of the list of filters' trees is related to the category *TS* and the last element is related to the category *VA*. In the following our structure is built following this order.

For a given category, we obtain the filters that concern it and that do not contain a category of inferior cardinality. For instance, in figure 3, we obtain from category TS a tree containing the filters verifying it and the categories which cardinality is superior. It means that this tree contains the filters where the category TS is tested alone or with the categories *PA* and *VA*. From category *PA* (the second element of the list) we obtain the filters where the category *PA* is tested alone or along with the category *VA*. In that way, it is possible to eliminate the tree of filters (the tree is not browsed) if there is no description belonging to this category in the perceptible information.

A node is a set of filters which f_C validation concerns the same set of categories. To distinguish these filters, we append a letter to the end of the filter's name. For instance, the node *F4:PA-TS-VA-VA* (figure 3) contains all filters where f_C is validated with the description of a pedestrian agent, a traffic sign and two vehicle agents (in addition of the vehicle agent owner of the list of filters' trees). The filter *warning* belongs to the node *F1:VA* since f_C is related to one vehicle agent.

An arc is an inclusion relation between subsets of filters. Following this relation, the depth of a node provides the number of categories that a filter has to test. An arc between two nodes indicates that the deeper node (the child) contains the filters which evaluation requires a category more to test than the shallower node (the parent). For instance, the children of the node *F1:TS* are *F1:TS-VA*, *F1:TS-PA* and *F1:TS-TS* with respectively the addition of the categories *VA, PA* and *TS*.

The naming convention of the filters respects these constraints and indicates the depth of the filter in the tree and the order of the category taken into account. For instance, figure 3, *F2:TS-VA* specifies a filter that is at the depth 2 and concerning the categories *TS* and *VA*. This naming convention has two advantages. The first is algorithmic because respecting this order allows us to look efficiently for filters that can be evaluated (section V). The second advantage is practical since the respect of this order insures the uniqueness of filters. For instance, the filter *F2:TS-VA* does not belong to the tree of the category *VA*. Nevertheless, for clarity's sake, we use also a more explicit naming (like *warning* for a filter *F1:VAx*) in our explanation when the position of the filter in the tree is not discussed.

For a given depth, we order the filters in decreasing order of categories cardinality. Indeed, for a given node, these children are explored if the additional category belongs to the perceptible categories (section V). Therefore, processing in priority the children that have potentially the most chances to have descriptions increases the possibility to have a valid context and to stop the search. For instance, there are potentially more vehicle agents than pedestrian agents that are perceived by a vehicle agent. For instance, in figure 3, the filters belonging to the node *F3:TS-PA-VA* are tested before the filters belonging to the node *F3:TS-PA-PA* if the category *VA* belongs to the set of perceptible categories. If the objective is to retrieve all the possible contexts of the agents, like in our experiments, then the ordering of the nodes has no consequence.

If a child node has no parent, i.e. there exists no filter concerning the parent's categories, then the parent node is created but is empty.

Starting from this filters' structure and the data related to the descriptions that are accessible from the environment, we design an algorithm, which identifies efficiently the possible filters.

V. Context computation algorithm

A. Algorithm global overview

Figure 4 illustrates our explanation of the foundations of the proposed algorithm. The general principle is for the activated agent to test the only filters for which there exists perceptible descriptions. This information is provided by the environment and used by the agents to explore their list of filters' trees. For a tree, the agent has to test the filters contained in the root then in each of its children if the added category exists in the descriptions' list provided by the environment. It is noteworthy that a child node may have validated filters because accessible descriptions validate its conditions while its parent does not contain any valid filter. For instance, a filter belonging to the node *F3:TS-PA-PA* can be valid even if no filter in the node *F3:TS-PA* is valid. Our data structure is built to exploit the perception of the entities and not their value.

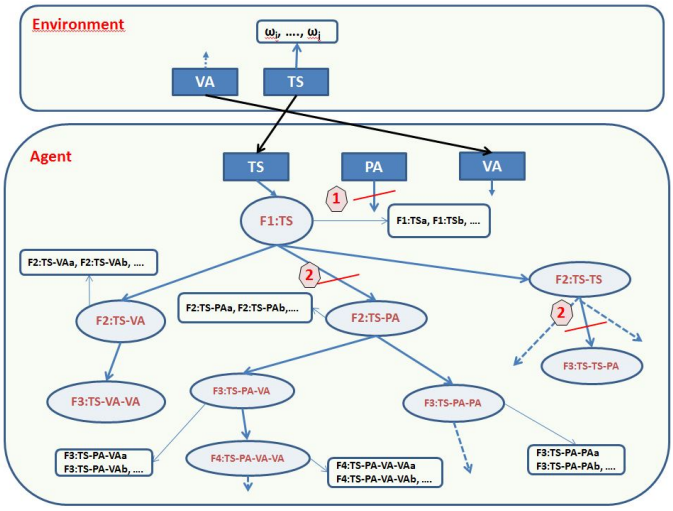

Fig. 4. Global overview of data model

We propose a classical scheduling algorithm (algorithm 1) for which the number of time ticks is fixed (T). At each tick, we activate the agents of the simulation by providing a dictionary which keys are the categories and the values the accessible entities' lists (algorithm 1-(4)). This dictionary contains the only categories, which list is not empty. It is not sorted because our algorithm aims to provide all the possible contexts; it is then necessary to explore all the possible trees. The prefixed notation indicates the access to the members of the concerned element.

Algorithm 1 Simulation scheduling algorithm

Require: $T > 0$
1: $t \leftarrow 0$
2: **while** $t < T$ **do**
3: **for all** $a \in \Omega_{\mathcal{A}}$ **do**
4: $AccessibleWorld \leftarrow DescriptionComputation(a)$
5: $a.activate(AccessibleWorld)$
6: **end for**
7: $t \leftarrow t + 1$
8: **end while**

B. List of filter' trees browsing

When an agent is activated, he executes the *perception - decision - action* loop. Our proposal concerns the *perception* step. A part of this step is already performed since the agent has the accessible descriptions (*AccessibleWorld*) and has to find the possible contexts. The browsing of the list of

categories belonging to *AccessibleWorld* is already a selection of the potential filters since the filter's tree referred by a category that does not belong to it is not explored. In Figure 4, the list of trees of the categories *VA* and *TS* are explored but not the filters' tree of the category *PA* (following the selection labeled with the number 1).

Algorithm 2 Activate: agent activation algorithm

Require: *AccessibleWorld*
1: **for all** $Category \in AccessibleWorld$ **do**
2: **for all** $f \in self.Filter[Category][1]$ **do**
3: **if** $f.valid(self)$ **then**
4: **if** $f.trigger(self, AccessibleWorld)$ **then**
5: $self.validFilter.add(f)$
6: **end if**
7: **end if**
8: **end for**
9: **for all** $t \in self.Filter[Category][2]$ **do**
10: $self.recursiveFilterTriggering(t, AccessibleWorld)$
11: **end for**
12: **end for**
13: $self.decision()$
14: $self.action()$

In algorithm 2, the agent explores the filters' trees of each category belonging to *AccessibleWorld* (the notation $self$ refers to the agent himself). The filter's trees (*PotentialContext*) are recorded in a ordered dictionary with the category name as a key and the filters' trees as value. The algorithm explores the filter's tree in two steps:

1) It explores the filters of the current node (value 1 in algorithm 2-(2)): it tests the f_a part of the filter (condition on the state of the agent) before to test f_C (the conditions on the concerned descriptions). This order avoids to browse the related categories if the current state of the agent makes the filter not adapted. For instance, for the filter *warning*, it is not useful to test all the perceptible vehicle agents if the activated vehicle agent is not in the central lane of the roundabout. If the agent uses a behavioral automaton, his current state can be used here to reproduce a logo-based simulation.
2) It explores the children saved in a sublist (value 2 in algorithm 2-(9)): the exploration of the child is performed following a recursive process applying the same principles as for the root.

If the filter is valid given the state of the agent (algorithm 2-(3)) and the necessary descriptions (algorithm 2-(4)) then it is saved in a list of valid filters of the agent. This list is made of sublists containing the name of the filter and the list of descriptions validating it. We choose not to compute all the combinations of perceptible descriptions of a given filter and to only select the first successful.

The input parameters of the recursive algorithm are the part of the filters' tree that is explored and the perceptible descriptions. A partial filters' tree is a list of lists with, for each imbrication level, three information:

1) The name of the new category taken into account. For instance *VA* for the fist call following the filters tree given figure 4;
2) The list of filters of the node. For instance the filters belonging to the node *F2:TS-VA* figure 4;
3) The list of children that reproduce this structure. For instance the structure related to the filters' tree with *F3:TS-VA-VA* as a root.

Algorithm 3 Recursive filters' tree exploration

Require: *partialTree*
Require: *AccessibleWorld*
1: **if** $partialTree[1] \in AccessibleWorld$ **then**
2: **for all** $f \in partialTree[2]$ **do**
3: **if** $f.valid(self)$ **then**
4: **if** $f.trigger(self, AccessibleWorld)$ **then**
5: $self.validFilter.add(f)$
6: **end if**
7: **end if**
8: **end for**
9: **for all** $t \in partialTree[3]$ **do**
10: $self.recursiveFilterTriggering(t, AccessibleWorld)$
11: **end for**
12: **end if**

With this information, the algorithm tests the existence of the category (algorithm 3-(1)) and if successful, it tests the nodes of the filter (algorithm 3-(2)) then accesses the children nodes (algorithm 3-(7)). If the category does not belong to perceptible categories then this part of the filters' tree is not explored. For instance, the filters' tree with the category *PA* (figure 4-cut 2) are not explored because this category does not belong to perceptible categories.

VI. Experimentation

A. Simulation settings

To validate our algorithm, we choose a theoretical framework in which we set categories and filters. Our environment is a 2D grid that contains 60.000 entities distributed in 6 categories in addition of 100.000 agents. For each category, we set a relative number of entities (table I) to have categories that are poorly represented ($C4$) or well represented ($C9$). For each description, we generate random values between 0 and 20 for five properties. We simulate agents situated on a grid with a size varying from 500 × 500 to 5000 × 5000. The agents have to decide which action to perform based on the MAS entities that are present in their perception field. The position of the entities is random.

TABLE I. Cardinality of the different categories

name	cardinality
$C4$	2.500
$C5$	5.000
$C6$	7.500
$C7$	10.000
$C8$	15.000
$C9$	20.000

In each filter, we define two conditions on the properties by category (f_C) and the evaluation part of the agent (f_a) comprises three conditions. For instance, for a filter $F2-78$

of depth 2, there will be 7 conditions (4 conditions for f_C and the 3 conditions of f_a).

A filter is designed with nested loops and each loop is related to one category. The nested loops are ordered following the increasing category order. For instance, for a filter $F2-78$, there are two loops and the loop for the category $C8$ is nested within the loop for the category $C7$. The objective is to minimize the cost of the filter computation: there is *a priori* less entities belonging to the category $C7$ to test. During evaluation, when a description of the current loop validates the conditions (for $F2-78$, filter $\omega_i \in C7$), then the nested loop is processed for the next category to test. Again, if a description of the current loop validates the conditions (for $F2-78$, filter $\omega_j \in C8$) and if there is no nested loop, then the filter is valid for the entities' tuple (for $F2-78$, filter $(\omega_i \in C7, \omega_j \in C8)$) and the trigger function returns true (algorithm 1-(4); algorithm 3-(4)).

The filters' tree for our tests is the one described in figure 5. Filters are chosen to respect a homogeneous dispatching between categories in order not to introduce bias. Hence for a category, there exists 3 filters of first level (F1), 6 filters of second level (F2) and a filter of third level (F3) for a total of 41 filters. An agent of the simulation has the same filters' tree.

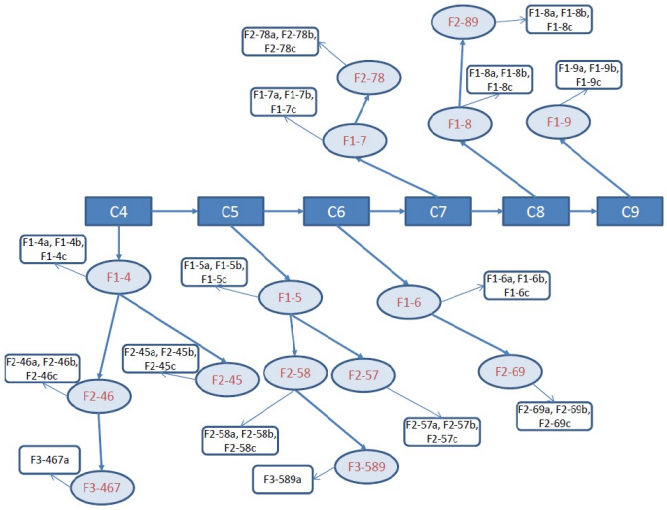

Fig. 5. General organization of the data

We compare our proposal with a solution in which filters are not organized and are explored iteratively. The objective is to compare our proposal with an algorithm computing the context with conditional branching but that remains generic. We call this proposal a *iterated* algorithm while ours is called *structured* algorithm.

We perform 50 simulations of one time cycle and measure the time spent to generate the possible filters. To ensure a similar behavior of the two algorithms (same world state during evaluation), at each cycle the activated agent executes both algorithms and measures their execution time before modifying the state of the world.

Our algorithm have been developed in Python 3.3 and processed on a PC with an Intel Core i7-2600 CPU@3.4GHz and 8 GB memory.

B. Results

We propose two parameters for the evaluation:

- The size of the perception field: the variation of this parameter enables to know when the decrease of the context computation runtime becomes negligible according to the time needed to explore the grid that contains the perceptible descriptions.
- The size of the grid: the variation of this parameter enables to modify the number of potential entities that are perceived by the agent with a constant grid exploration cost.

We vary the perception field value from 5 to 20 and measure the percentage that the context computation process represents within the total perception time step for the iterated algorithm. The results are given in table II. For instance, if the perception field value is 5 and the size of the 2D grid is 3000×3000 then 31.47% of the perception step processing time is related to the context computation and therefore 68.53% to the browsing of the 2D grid that is perceived by each agent. We observe that the context computation represents half the execution time of the perception process when the perception field is small and it decreases quickly (down to 16% for a 5000×5000 grid and a perception field of 20). If the perception field is greater than 20 then the runtime related to the context computation process becomes negligible. The increase of grid size causes a decrease of the context computation runtime process because the runtime to explore the grid remains stable while the context computation runtime decreases (there are less entities to process).

TABLE II. STRUCTURED ALGORITHM: RELATIVE COST OF CONTEXT COMPUTATION W.R.T PERCEPTION PHASE

	Perception field		
Grid size	5	10	20
500×500	52.34%	41.01%	32.48%
1000×1000	51.48%	42.45%	29.92%
3000×3000	43.95%	31.47%	22.84%
5000×5000	41.76%	18.39%	16.43%

The second test is a comparison between the structured algorithm and the iterated algorithm w.r.t. the context computation runtime. The table III provides the improvement percentage when using our algorithm w.r.t to the perception field and the size of the grid. It means that if the perception field value is 10 and the size of the 2D grid is 3000×3000 then the necessary time to compute the context is 53.56% less with the structured algorithm than with the iterated algorithm.

TABLE III. STRUCTURED VS. ITERATED ALGORITHM: RELATIVE PERFORMANCE OF CONTEXT COMPUTATION COST

	Perception field		
Grid size	5	10	20
500×500	14.28%	8.92%	1.61%
1000×1000	30.7%	9.06%	8.63%
3000×3000	81.77%	53.56%	22.63%
5000×5000	90.83%	73.38%	25.03%

Our algorithm is always better than the iterated algorithm but this advantage decreases conversely to the increase of the perception field. This result is coherent with the principle of our algorithm: the more entities the agent perceives, the less

there are empty categories. Nevertheless, we saw with the first series of experiments that the perception field has to be limited to 20, because with a superior value, the context computation runtime becomes negligible according to the browsing of the grid that contains the perceptible entities.

The table IV highlights the fact that our algorithm improves the simulation execution time whatever the perception field size. For instance if the perception field value is 10 and the size of the 2D grid is 3000×3000 then time related to the perception step is 16.85% less with the structured algorithm than with the iterated algorithm. This gain is not anecdotal since the average gain for the duration of a single simulation cycle is 1.25 second.

TABLE IV. STRUCTURED VS. ITERATED ALGORITHM: RELATIVE PERFORMANCE OF SIMULATION TIME

	Perception field		
Grid size	5	10	20
500×500	7.48%	3.66%	0.52%
1000×1000	15.8%	3.85%	2.58%
3000×3000	35.94%	16.85%	5.17%
5000×5000	37.93%	13.5%	4.11%

Our algorithm is dependent of the cardinality ordering given by the designer. To test the consequence of a wrong order, we have executed the same simulations with the same list of filters' trees (figure 5), but with an inverse order of the cardinality given in table I (for instance, we have set 20.000 entities of category $C4$ and 2.500 entities of category $C9$). The improvement of the structured algorithm compared to the iterated algorithm in the case of a wrong order is given in table V.

TABLE V. STRUCTURED VS. ITERATED ALGORITHM: RELATIVE PERFORMANCE OF CONTEXT COMPUTATION COST (WRONG ORDER)

	Perception field		
Grid size	5	10	20
500×500	11.88%	7.36%	1.4%
1000×1000	26.55%	12.39%	7.63%
3000×3000	76.78%	43.76%	18.31%
5000×5000	88.67%	66.98%	28.05%

The incorrect order penalizes the structured algorithm, since the runtime improvement becomes less important (compare with table III). However, the structured algorithm remains more efficient that the iterated algorithm. Indeed, even if the filters' tree is more deeply explored when the categories' ordering is incorrect, the structured algorithm still behaves better than the iterated algorithm.

VII. CONCLUSION AND PERSPECTIVES

In this paper, we proposed a solution to decrease execution times of a multiagent simulation. This issue is important because high execution times risk unfortunately to circumscribe the use of the multiagent paradigm to small-size simulations. Our proposal is focused on the optimization of context computation. We propose to model a context as a filter, which enables us to propose a filters' structure as a tree and an algorithm for the agent to process it efficiently. The proposed structure exploits the *a priori* cardinality of the different categories that an agent can take into account in the evaluation of his context. Our first set of experiments shows that, even when this *a priori* order is wrong, our proposal remains more efficient than an iterated algorithm.

Our future work concerns the introduction of new data structures, such as lattices, in the organization of filters. In addition, we plan to enrich the evaluation of our proposal with several real world applications.

REFERENCES

[1] F. Béhé, S. Galland, N. Gaud, C. Nicolle, and A. Koukam. An ontology-based metamodel for multiagent-based simulations. *Simulation Modelling Practice and Theory*, 40(0):64 – 85, 2014.

[2] F. Bousquet, I. Bakam, H. Proton, and C. L. Page. CORMAS : Common resources and multi-agent systems. In Springer-Verlag, editor, *Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 826–837, 1998.

[3] N. Collier. Repast: An extensible framework for agent simulation. *The University of Chicagos Social Science Research*, 36, 2003.

[4] A. K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer, 1999.

[5] R. Evertsz, F. E. Ritter, P. Busetta, M. Pedrotti, and J. L. Bittner. Cojack-achieving principled behaviour variation in a moderated cognitive architecture. In *Proceedings of the 17th conference on behavior representation in modeling and simulation*, pages 80–89, 2008.

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

[7] C. Greenhalgh and S. Benford. Massive: a collaborative virtual environment for teleconferencing. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 2(3):239–261, 1995.

[8] Y. Kubera, P. Mathieu, and S. Picault. Interaction-oriented agent simulations : From theory to implementation. In M. Ghallab, C. Spyropoulos, N. Fakotakis, and N. Avouris, editors, *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, pages 383–387. IOSPress, 2008.

[9] E. S. Liu and G. K. Theodoropoulos. A continuous matching algorithm for interest management in distributed virtual environments. In *Proceedings of the 2010 IEEE Workshop on Principles of Advanced and Distributed Simulation*, PADS '10, pages 13–22, Washington, DC, USA, 2010. IEEE Computer Society.

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

[11] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system: A toolkit for building multi-agent simulations. Santa Fe Institute Santa Fe, 1996.

[12] M. Sierhuis, W. J. Clancey, and R. J. Van Hoof. Brahms: a multi-agent modelling environment for simulating work processes and practices. *International Journal of Simulation and Process Modelling*, 3(3):134–152, 2007.

[13] C. van Dinther. Agent-based simulation software. *Adaptive Bidding in Single-Sided Auctions Under Uncertainty: An Agent-based Approach in Market Engineering*, pages 85–122, 2007.

[14] G. Wagner. Aor modelling and simulation: Towards a general architecture for agent-based discrete event simulation. In *Agent-Oriented Information Systems*, pages 174–188. Springer, 2004.

[15] T. Warden, R. Porzel, J. D. Gehrke, O. Herzog, H. Langer, and R. Malaka. Towards ontology-based multiagent simulations: The plasma approach. In *24th European conference on modelling and simulation (ECMS 2010). European Council for Modelling and Simulation*, pages 50–56, 2010.