We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

30

Effective agent-based geosimulation development using PLAMAGS

Tony Garneau, Bernard Moulin
Département d'informatique et de génie logiciel, Université Laval
Québec, Canada

Sylvain Delisle
Département de mathématiques et d'informatique, Université du Québec à Trois-Rivières
Québec, Canada

1. Introduction

During the past decade, the technology based on Software Agents (SA) has been applied to a large number of domains such as computer games, entertainment movies involving animated characters, virtual reality, user interface design involving personal agents, web-based interfaces and tutoring systems using virtual avatars, to name a few. Since the SA technology is now mature, new fields may benefit from it as, for example the field of Geographic Information Systems (GIS) applied to decision support, and more specifically the domain of geo-simulation. During the past decade, the number of software using digital geographic data has increased a lot, being popularized by applications such as web-mapping, assistants for route planning and monitoring using GPS (Global Positioning System), exploration of virtual cities and geographic territories using tools such as MapQuest and Google Earth. Besides these popular applications, GIS have been used for a long time by governmental and private organizations whose activities deal with the geographic space in a way or another. Most of these applications, however, are complex since they deal with spatial dynamic phenomena and usually involve large populations of individuals (persons, animals, insects, plants, etc.) and their interactions (Courdier et al. 2002).

There are numerous situations that decision makers from various sectors (governmental, economics (Fagiolo et al. 2007), military, industrial (Gnansounou et al. 2007), medical, social) need to monitor in order to insure human security (Foudil and Noureddine 2007) (in case of flood, earthquake or wild fire), the respect of public order (crowd monitoring, evacuation of people, peace-keeping activities) or the adequate use of infrastructures (i.e. monitoring of people and households transportation and shopping habits to better plan urban infrastructures). In such situations, GIS are very useful to gather data about geographic phenomena and to carry out spatial operations on it in order to generate various thematic maps that provide decision makers with an overview of the situation and its evolution.

However, GIS should be enhanced with other techniques in order to provide an enhanced support to decision makers. Geo-simulation (Beneson and Torrens 2004) is such an approach which became popular in geography and social sciences in recent years. It is a useful tool to integrate the spatial dimension in models of interactions of different types (economical, political, social, etc.) and is used to study various complex phenomena (CORMAS 2009), especially in the domain of urban dynamics and landcover planning.

Using Multi-Agent in Geo-Simulation
Since these phenomena usually involve large populations in which individuals behave autonomously, several researchers thought to take advantage of multi-agent simulation techniques (d'Aquino et al. 2003; Guyot and Honiden 2006; Gnansounou et al. 2007; Papazoglou et al. 2008), which resulted in the creation of the new field of Multi-Agent Geo-Simulation (Koch 2001; Moulin et al. 2003). However, most geosimulation applications deal with very simple agent models, mainly expressed in terms of simple behavior and decision rules, either attached to spatial portions (i.e. cells in cellular automata) or to simple agents moving around in a virtual geo-referenced space (Beneson and Kharbash 2005; Müller et al. 2005). Indeed, the degree of sophistication of agent models depends on the scale of the simulation. For example in the traffic simulation domain, different kinds of simulations are developed at macro-, meso- and micro-scales in order to respectively study traffic flows in regions of different extent (macro- or meso-level) or to create micro-models based on individual vehicles behaviors (Helbing et al. 2002; Bourrel and Henn 2003). Nevertheless, most models that drive such simulations of agents' movements in geographic space are either based on mathematical models (usually systems of differential equations) or on simple rules (Torrens and Benenson 2005; Levesque et al. 2008).

However, there is a large variety of phenomena in which individuals need to make informed decisions, taking into account the characteristics of the geographic environment as well as the effects of other agents' actions. Hence, there is a need for more sophisticated agents' models, akin to intelligent software agents' models, in order to carry out autonomous behaviours within geographic virtual environments (Crooks et al. 2007). Such a model was proposed by Moulin and colleagues (Moulin et al. 2003) in which agents have several knowledge-based capabilities: 1) perception (of terrain features, objects and other agents); 2) navigation (autonomous navigation with obstacle avoidance coupled with perception); 3) memorization (of perceived features and objects), communication (with other agents)); and 4) objective-based behaviour (based on interrelated goals and activity plans).

However, whatever the sophistication of the models, specifying agent behavior models is a difficult task and designers need efficient and user-friendly tools to support them. Some existing tools for agent-based simulations, such as CSF, GASP (GASP), HPTS (Donikian 2001), AI.Implant (AI.implant 2003; AI.implant 2009) and PathEngine (PATHEngine 2009), deal with the spatial aspects of agent behaviors by providing good navigation mechanisms for the characters. Unfortunately, they tend to neglect the proactive aspects of agents and their interactions with the environment. Other tools such as SimBionic (Fu et al. 2002) and SPIR.OPS (SPR.OPS 2009) offer sophisticated specification means for objects/agents behaviors based on models inspired by finite state machines (Fu et al. 2003). But, the use of finite state machines leads to complex graphs to represent relatively simple reactive

behaviors. Behaviors developed using these tools lead to reactive agents or "navigation driven" agents (Cutumisu et al. 2006). Hence, they are not sufficient for the development of geosimulations of social phenomena in which agents need to implement knowledge-based capabilities in relation with the space in which they evolve. In both cases, resulting agents do not have decision-making capacities. Moreover, since most of these tools do not provide perception mechanisms, agents cannot apprehend the virtual environment (act in the environment and interact with the object/agent contained in it).

To help solve these problems, we claim that software agents with space-related capabilities should be introduced in the virtual spatial environments associated with geo-simulations. We call these agents "spatialized SAs" (SSAs for short) and they are characterized by the following properties:

- Autonomous and individual perception mechanism
- Decision-making in relation to a geo-referenced virtual environment
- Proactive and autonomous behaviors taking into account their knowledge about the world (the virtual environment).

The specification of this type of agents is a difficult task and, to our knowledge, no existing simulation environment enables designers to specify SSAs. In the context of the PLAMAGS Project (Programming LAnguage for Multi-Agent Geo-Simulations), we have developed a high-level language and a complete development environment allowing a designer to quickly develop and execute multi-agent geo-simulations. The PLAMAGS toolkit was motivated by the need to provide a complete and real programming language dedicated to the specification and the execution of multi-agent geo-simulations.

Section 2 introduces the architecture and the main concepts on which the PLAMAGS language is based. Section 3 presents the main characteristics of the language. Section 4 is an overview of the IDE. Section 5 presents miscellaneous PLAMAGS features and Section 6 concludes the paper with a discussion and some future work.

2. Main models provided by the PLAMAGS architecture and language

This section aims to introduce the overall design we propose for MAGS' development. Section 2.1 presents the principles that guided PLAMAGS' development as well as how they are interrelated to form a coherent whole. Section 2.2 further describes the different models that PLAMAGS supports with respect to MAGS' specification and implementation.

2.1 The PLAMAGS architectural model

A fact that greatly contributes to make MAGS' development a challenge is the inherent necessity to work with two very distinct sets of concepts, GIS and Multi-Agent Based Simulations, which share neither the same problems nor the same concerns. Thus we have to conciliate these differences in order to allow the fruitful interactions necessary to the synergy we seek.

As a matter of fact, Figure 1 presents an overview of the model upon which PLAMAGS is based. As it can be noticed, the main components of a PLAMAGS simulation's model are: 1)

a VGE (Virtual Geographical Environment) that renders the simulation environment; 2) the agents and objects located in this environment (which are characterized by behaviors); 3) the simulation scenario and, ultimately 4) the results of the simulation. These four elements are in constant interaction and constitute the core of the system. However, on a more global level, PLAMAGS' architecture can be seen as two distinct parts; first the programming language for the MAGS, and, second, a set of tools to facilitate its use. These tools are bundled into an IDE (Integrated Development Environment) that simplifies as much as possible the language's usage. The IDE also provides the user with a development framework.

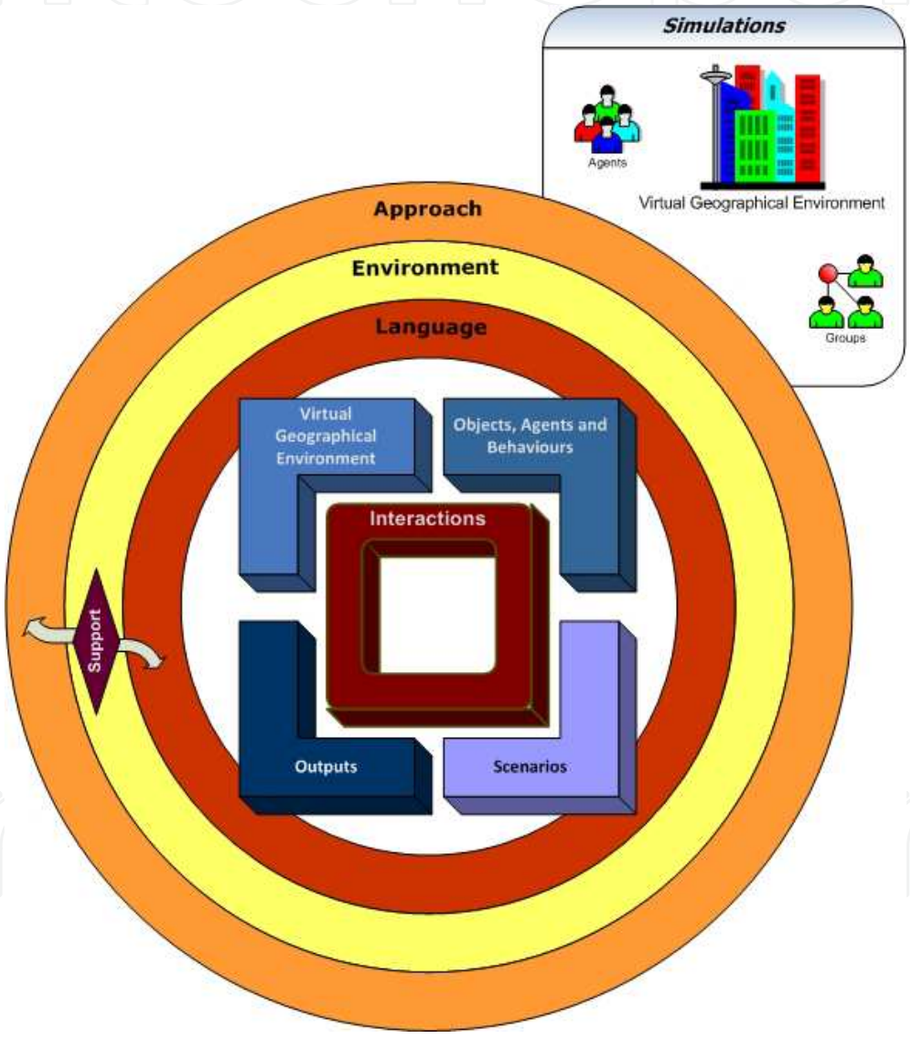


Fig. 1. PLAMAGS' architecture

Since we cannot present PLAMAGS' architecture in details in this chapter, we will briefly present an overview of the most important elements, such as the VGEs and scenarios, as

well as the specification of objects, agents and their behaviors. We will also present the main tools of the IDE, and we will show how the language can be used to retrieve and manipulate spatial information.

2.2 Objects and agents

Independently of their specific characteristics (be they passive objects, reactive agents or proactive/cognitive agents), SSAs participating in geo-simulations have common important characteristics: they are situated in a virtual geo-referenced space (they actually move in it), they must apprehend its content (they perceive the objects and agents located in the virtual space), and they must interact with the elements (objects and agents) contained in the virtual space.

As an illustration, we developed a simulation of a demonstration (see figure 2). In such a simulation agents need to perceive tear gas and to react to them. Agents also need to distinguish different types of agents such as police, rioter and demonstrator agents and to behave according to their perception of the other agents' actions. As another example, pedestrians participating in a peace walk must be able to perceive and follow the group walking along a planned route, adjust their own pace to the crowd's pace, avoid cars and other obstacles. We distinguish three categories of SSAs, depending on the complexity of their behaviors, reasoning and interactions with the virtual space: passive SSAs, reactive SSAs, and cognitive SSAs.

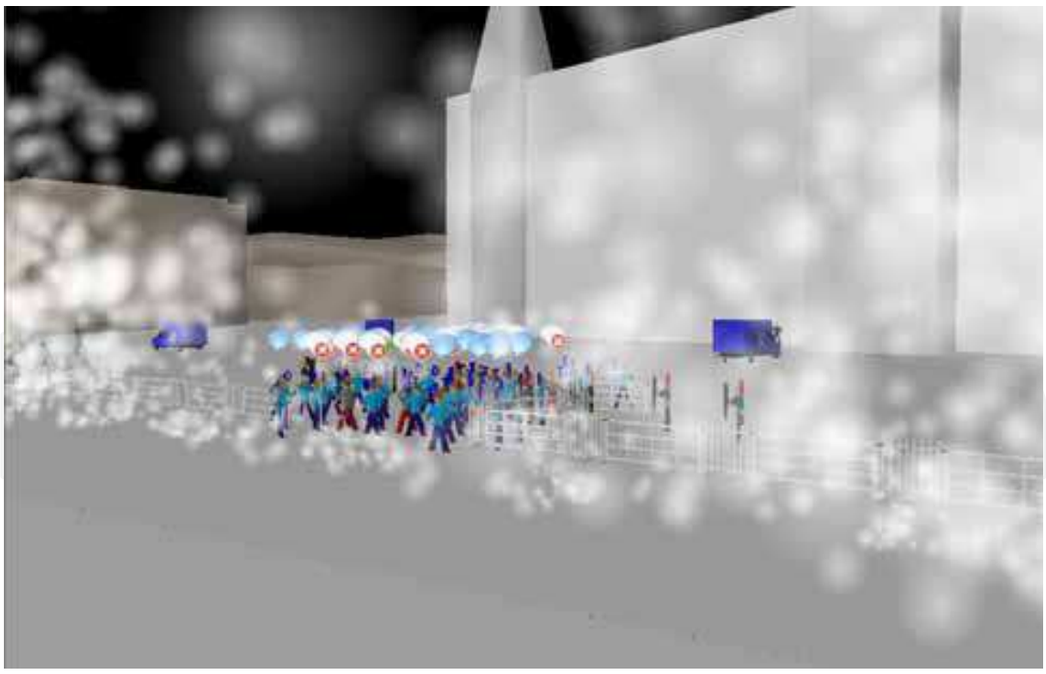


Fig. 2. Distant view of a hot spot of the event.

2.2.1 Static object (passive SSA)

In a simulation, numerous components are still objects having a physical presence in the VGE, but no behavior. These objects may have various properties such as a color, a weight and a dimension, but they do act. We may also need other types of objects that are data structures without any physical representation as for example, the representation of a position in the virtual environment.

In our model, we use a specific language structure to represent passive SSAs. This structure is called a *static object type* which is equivalent to an object-oriented (OO) class in an object-oriented language enhanced with a visual representation and a spatial/physical definition (if necessary). As for OO classes, a static object type has properties (which can be constant or modifiable) and methods. Methods can be used to attach processes to static objects as for example to change their location, their visual representation and change their properties as a result of the actions applied by agents on the objects. In addition to standard OO classes' capacities, static objects can own a visual representation and a spatial/physical definition and description (bounding volume, mass, etc.). In the demonstration simulation, fences are defined as static objects since they only have some properties and a physical presence, but no behavior (see figure 3).

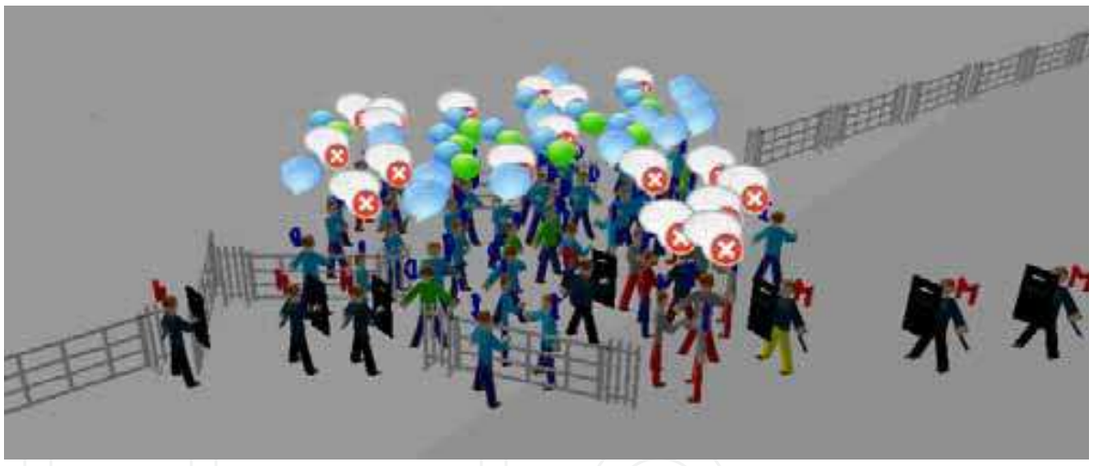


Fig. 3. Confrontation between the police and demonstrators.

2.2.2 Active object (reactive SSA)

Reactive SSAs have relatively primitive behaviors that can be efficiently represented using a reactive agent approach. We represent reactive SSAs thanks to a specific language structure that we call active object. Active objects are equivalent to reactive agents in classical agent models. Their capabilities are similar to those of static objects' (properties, methods and visual/spatial/physical definition) but are augmented with a set of lists of rules used to specify their behaviors. Reactive SSAs have no elaborate decision-making capabilities: they only react to their inputs (most of the time obtained from their perception mechanism) thanks to the aforementioned lists of rules which are automatically triggered by the

simulation engine. Using these rules, the reactive agent has the possibility to interact with its environment and other SSAs during the simulation (Levesque et al. 2008).

In our simplified demonstration simulation (see figure 4), considering that squad members 'blindly' obey to the instructions of their squad leaders, we chose to specify them as reactive SSAs, using an active object type called *squad-member*.

Fig. 4. Formation line of squad members.

Figure 5 presents some parts of the *"squad-member"* type and the list of rules defined to represent its reactive behavior. In Figure 2, the method *"moveToward"* is a "managed action" (also called "perform"), a structure provided by PLAMAGS and explained in Section 3.

```
246 public active object Squad
247   attribute: ....
248       rules SquadRules trigger when [newDestination() == true]
249   method: ...
250
251 private rules SquadRules mode disjunctive
252   rule WP_1 mode conjunctive        ...
253   rule FL mode conjunctive
254       lhs IS_FL [operation] == [Commander.FORMATION_LINE] // other lhs...
255       rhs GO_FL call moveToward(destination.x, destination.y, 1, true) // other rhs...
256       ...
```

Fig. 5. Line 248 declares that "squad" objects use a list of rules called "SquadRules" (defined at line 251) which will be automatically triggered by the simulation engine.

2.2.3 Agent (cognitive SSA)

We consider cognitive SSAs as agents that behave autonomously. Such agents must be able to interact with their environment (virtual geographic environment, objects and other agents), make decisions with respect to their own states and preferences and act accordingly. A reactive approach is not sufficient to represent these behaviors. We thus defined cognitive SSAs thanks to a specific language structure that we call agent.

In addition, to all the attributes of an active object, an agent is characterized by a number of static and dynamic variables whose values describe the agent's state at any given time. Using these variables, the system can simulate the evolution of the agents' dynamic states and trigger the relevant objectives. An agent is also associated with a behavior which is represented by a set of multi-layered directed graphs, each one composed of a set of objectives that the agent tries to reach (see Figure 6).

2.3 Behaviors

Behavioral graphs are the most powerful and expressive behavioral data structures existing in PLAMAGS. They are used to define complex agent's behaviors of simulations' agents. The power of these graphs lies in their high flexibility, customizability and their ability to represent behaviors in different ways.

A behavioral graph in PLAMAGS is a multi-layered graph in which nodes represent objectives which can be either atomic or composed of sub-behaviors. The objectives are organized in hierarchies such that elementary objectives (called simple objectives) are associated with actions that the agent can execute (i.e. objectives "PresenceAct" and "CheckAround" in Figure 3). Each agent owns a set of objectives corresponding to its needs (Moulin et al. 2003). An objective is associated with rules containing constraints on the activation, execution and completion of the objective, called activation rules, execution rules and completion rules (Moulin et al. 2003; Garneau et al. 2008). Constraints are dependent on time, on the agent's state, and on the environment's state. Objectives are also linked to resources that must be either acquired or already owned for them to be executed. The selection of the current agent's objectives relies on the graph structure, on previously executed objectives, on the missing required resources and on priorities and activation/execution/completion rules related to the agent's objectives. The execution of an objective is always conditional to its execution rule being triggered and the required resources being available. An objective's priority is primarily a discriminating function or expression used to choose between potential future objectives. It is also subject to modifications brought about by the opportunities that the agent perceives in the environment and by the temporal constraints applying to the objective. Resources are agents' assets that can be assigned exclusively to an objective's execution. The allocation of resources between objectives for iteration is dictated by the objectives' priorities.

The structure of the multi-layered directed graph allows us to define a behavior at different levels of abstraction and to divide behaviors into sub-behaviors. An abstraction level can be added by inserting a compound or an aggregate objective in the behavior. A compound objective can be thought of as a decomposable structure representing a sub-behavior (its structure is similar to a behavior structure), see Figure 7. Aggregate objectives are also

decomposable structures, they are composed of a set of objectives, but those one are not interrelated. These objectives allow us to represent goals (or composite objectives) where neither a hierarchical structure nor a predefined sequence of objectives is needed. Compound and aggregate objectives are perfectly suitable to regroup an agent's objectives by goal. Since "non-simple" objectives are composed of other objectives, any number of abstraction levels can be specified. The decomposition stops when an objective is composed of actions (corresponding to simple objective). At this time, it is considered to be the "execution level" of the behavior.

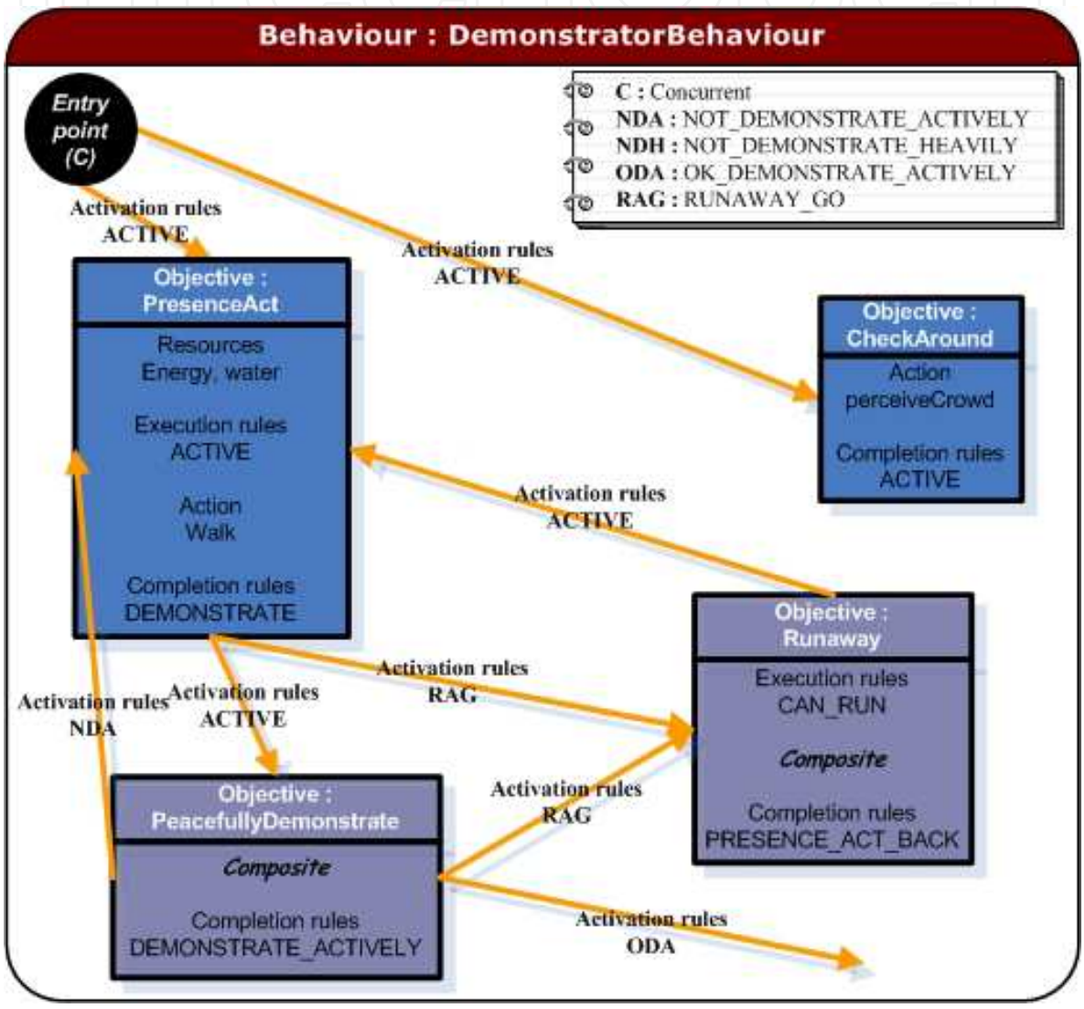


Fig. 6. A part of the demonstrator's behavior (in the demonstration simulation) similar as those displayed through PLAMAGS's graphical interface.

Since agents often need to simultaneously achieve more than one objective, we provide an execution mode allowing to concurrently activate several objectives. The "mode" declaration is specified for each objective because concurrent activation is not desirable everywhere in a behavior graph. This allows to locally control the activation of parts of the

behaviour graph. For example, in Figure 6 the behaviour entry point (a special simple objective) declares "C" as an indicator to specify that successor objectives (PresenceAct and CheckAround) can be activated concurrently (if their respective activation rules are correctly triggered). But, everywhere else, only one successor of an objective will be executed at a given time (note that when "C" is not specified, the execution is considered to be single).

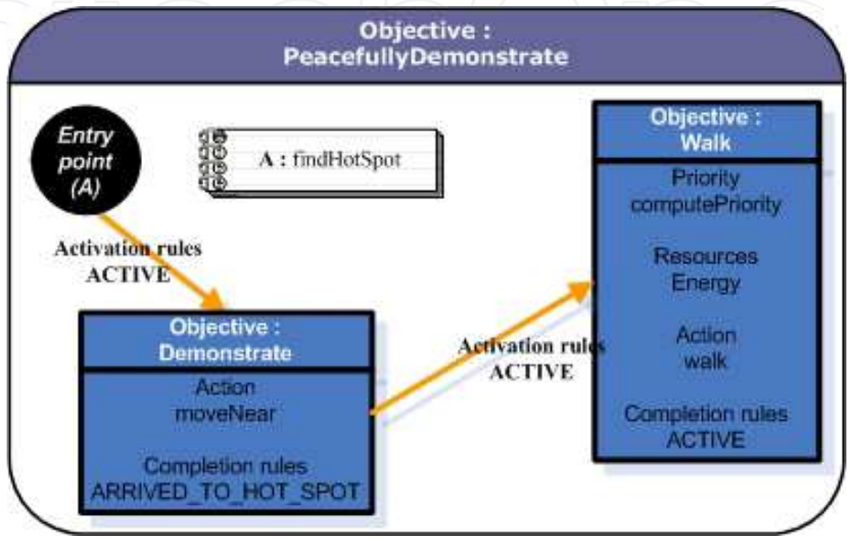


Fig. 7. Internal view of the PeacefullyDemonstrate objective used in the demonstrator agent's behaviour (in the demonstration simulation).

2.4 Other PLAMAGS models

Objects, agents and the agents' behaviors are the main concepts defining simulations' dynamism in PLAMAGS. However, there exist other concepts that further define a MAGS simulation namely scenarios and agent groups. On the one hand, *scenarios* allow specifying the VGE (including the 3D model, textures, coordinate system, elevation maps, collision matrix), simulation parameters (cycles per time unit, time unit to use), and the initial characteristics of agent populations. On the other hand, *agent groups* allow to describe the complex attributes inherent to a group such as the individual relations between members, influence of the group on its member (and conversely). Agent groups are perceived by the agents, objects and other groups of the simulation. A last simulation influencing type exists, but it is limited to model clouds of smoke (although thoroughly) using attributes like perceptibility, particles density, speed and acceleration of solid particles.

In this section, we presented the different types of SSAs and how to represent an agent's behavior. Section 3 will show how these concepts are defined and usable in the language.

3. The PLAMAGS language

This section presents the PLAMAGS language, a complete and expressive agent-oriented language providing standard procedural and object-oriented (OO) features. One of the main advantages of this language is that it facilitates the transformation of simulation designs into executable code. The data structure and features of this language are thus well adapted to the needs of MAGS developers.

In order to stay as close as possible to concrete needs for the development of MAGS, the language syntax includes all the tools required to easily create the VGE, to specify agents and scenarios. It is not possible to present all the language elements in this paper. We will mainly focus on the structures implementing the SSAs' characteristics presented in the previous section. Let us now define some technical aspects of the language.

3.1 Interpreted procedural, descriptive and declarative language

In essence, PLAMAGS is an interpreted language with a Java-based interpreter. This interpreter is included in a complete IDE. The language is used to write specifications, to implement and execute MAGS. Its syntax makes explicit all structures and parameters and is easily readable.

We can thus sum up this language as: i) easy to use for simulation descriptions; ii) within reach to non programming experts; iii) intuitive and easy to master for experts and developers.

In addition to its MAGS-oriented features, PLAMAGS also offers the syntactical structures found in mainstream structured and object-oriented programming languages such as Java, C++ and C#. Such structures must to be present in any programming. Unfortunately, there seems to be a trend in MAGS-oriented specification languages that provide adequate functionalities to specify MAGS, but lack the basic constructs of programming languages that are needed to develop complete systems. Breaking with that trend, PLAMAGS offers: 1) built-in data types; 2) user-defined data types (closely related to OOP classes); 3) ability to define methods and constructors; 4) support for overloaded and recursive functions; 5) control and loop structures; 6) set of logical and mathematical operators as well as a casting operator; 7) a set of access modifiers (inspired from OOP); 8) package-based modularity infrastructure; 9) instance, class-wide and global variables; 10) support for object composition.

3.2 Integrated support for MAGS

Notwithstanding the fact that the language supports structured programming much like general-purpose programming languages, either object-oriented or procedural, the appeal of PLAMAGS obviously comes from its integrated support for MAGS. This support is directly embedded into the language's basic structures and keywords and allowing for a simple specification of the simulation's attributes and components. All the main components of MAGS are mapped to keywords that are used to declare and define them: this facilitates the usage of the language in a purely descriptive way. Hence, a user can easily set up a simulation by completing some templates using only keywords and simple expressions.

The next two sub-sections present these language-embedded structures and the mechanisms offered by the language to retrieve and manipulate data related to the geo-spatial environment, as well as information generated by agents' interactions. These mechanisms are integrated in the language, thus enabling a completely seamless transition from specification of a MAGS to its implementation.

3.2.1 Simulation description

This section presents and justifies the structures chosen to implement the three types of SSAs introduced in Section 2.

Passive and reactive SSAs (i.e. static and active objects)

Since passive SSAs are inanimate components of the simulation having various properties or structures representing properties of other components, OO classes could be used to define such SSAs. However, our static object structure also provides the means to specify behaviors in the form of rules.

Reactive behaviors expressed by rules, allow a component to respond to stimuli (applying functions to these stimuli, to modify internal states, to carry out actions, etc). We implemented a rule list model that is directly inspired by traditional reactive approaches in which a rule is made up of a set of conditions. These conditions must satisfy certain constraints; and when these constraints are satisfied, a series of actions is triggered. In PLAMAGS' active objects (or reactive SSAs), the rule's conditions are relational operators that can be applied to both object properties and function calls. Most of the time, these function calls return computed data obtained from the perception. Whenever the rule conditions are satisfied, actions (method calls and spatialized actions) and properties modification are triggered. Figure 2 shows the definition of an active object and its rule list.

Rule list multiple usages

In PLAMAGS, rule lists are used in different situations. As we saw above, they are used as behaviors for active objects, but they can also be used to verify conditions when activating, executing and completing agents' objectives. However, independently of their use, they always have the same structure and the same execution logic. A PLAMAGS rule list contains one or several rules. Each rule is composed of a list of preconditions known as LHS (Left Hand Side) and a list of consequents known as RHS (Right Hand Side). A rule must have at least one LHS and one RHS. A precondition is always evaluated to "true" or "false". A consequent can result in a property modification or a method call.

Agents and their behaviors

As mentioned in Section 2.2.3, a cognitive SSA is characterized by a behavioral component which is represented as a layered graph composed of a set of objectives that the agent tries to reach (nodes). This sub-section introduces the main structures of the language implementing behaviors in PLAMAGS. The code presented in this sub-section represents the behavior and objectives of Figure 6 and Figure 7. An interesting feature of the behaviors is that they are directly transferable from the visual representation to PLAMAGS code. It allows for the modeling of the behavior in an intuitive way using a graphical representation. Thereafter, this model can quickly and easily be transferred into PLAMAGS executable

code. We will illustrate the various concepts by referring to the code specifying a demonstrator agent (Figure 8).

```
688 public agent Demonstrator
689   attribute: ...
690   behaviour DemonstratorBehaviour // optional : trigger every [...] or when [...]
691   method: ...
692 end agent
693
694 private behaviour DemonstratorBehaviour
695   entry execute concurrent
696       successor CheckAround activation rules ACTIVE
697       successor PresenceAct activation rules ACTIVE
698   end entry
699   exit .... end exit
700 end behaviour
```

Fig. 8. The "behavior DemonstratorBehaviour" declaration specifies that the demonstrator type will use a behavior called "DemonstratorBehaviour".

The behavior

The "behavior" is the global structure representing the whole behavior graph. It allows for identifying the "entry point" (see top left in Figure 3 and line 695 in Figure 8) of the behavior which can be viewed as the initialization of the behavior (this one will be executed only once). It also specifies that after the execution of the entry point, objectives "CheckAround" and "PresenceAct" will be executed concurrently thanks to the "execute concurrent" declaration (each one will be executed at each behavior execution).

The objectives

Objectives are the main components of the behavior. They are used to manage goals that an agent tries to reach. These behaviors can take different forms and they can be divided into three categories: simple objectives, compound objectives and aggregate objectives. Each objective has some basic elements. This section describes common elements to all objectives.

Basic objective elements

Whatever its type, an objective is characterized by some basic elements: a state (implicit), a list of successors (optional), required resources and priority (optional), an execution rule list (optional) and a completion rule list (mandatory). This section quickly describes these elements.

Objective states

The state of an objective is an implicit attribute representing the current context of an objective for a certain agent. This attribute can be consulted or modified using activation, execution and completion rules (the rule list example will show how to access this attribute). The runtime engine uses objective states to determine which actions it must undertake: to change the current running objective, to add or withdraw an objective from the execution process, etc.

Successors
A successor links an objective to a potential objective (its successor) that may be activated if the execution of the behavior of the first objective is successfully completed. A successor is composed of a destination objective (mandatory), an activation rule list for the destination objective (mandatory) and a priority function (optional) to discriminate objectives when necessary. Lines 696 and 697 of Figure 8 declare two successors named "CheckAround" and "PresenceAct" (these objectives must be defined elsewhere).

Activation rules
Activation rules are used to influence the state of a potential successor objective. For example, after the execution of the entry point in the demonstrator's behavior (see Figure 8), the behavior runtime engine has to choose the next objective to execute (in the next iteration). In this case, since the execution of the entry point's successors is specified to be concurrent, the behavior engine will have to execute all active successors. But, to check if an objective must be activated, the engine triggers the activation rules of each successor. Thereafter, the engine checks the state of each successor and it will schedule for execution all successors whose state is "active". Figure 9 shows an activation rule list used to verify if the objective "Runaway" must be activated. It checks whether some properties (anxiety, bravery and formation level of the crowd) satisfy certain levels. If it is the case, the state of the "Runaway" objective is set to "ACTIVE".

```
private rules RUN_AWAY_GO
  rule CHECK_STATES
       lhs S_1 [anxiety] >= [Crowd.ANXIETY_LEVEL]
       lhs S_2 [bravery] < [Crowd.BRAVERY_LEVEL]
       lhs S_3 [Crowd.getFormationLevel()] >=
               [Crowd.SCARING_FORMATION_LEVEL]
       rhs K set objective.this.STATE = [objective.ACTIVE]
end rules
```

Fig. 6. Rule list called "RUN_AWAY_GO" used in the "PresenceAct" and "PeacefullyDemonstrate" objectives.

Execution rules
The execution rules of an objective are rules which are automatically triggered immediately before each execution of an objective. These rules are used to control the execution of an objective by modifying its state. Once the rule list is triggered, the behavior engine uses the state value of the objective to determine if the objective needs to be executed at the current iteration, or if the system needs to wait and reevaluate them at the next behavior execution.

Completion rules
The completion rules of an objective are rules which are automatically triggered immediately after each execution of an objective. These rules are used to control the execution of an objective by modifying its state. Once the rule list is triggered, the behavior engine uses the state value of the objective to determine the action to be carried out: either re-execute the objective, or choose a successor, or stop the branch's execution (this graph's section). Note that the completion rules are only used to determine the action to perform on the currently executed objective. If the action consists in choosing another objective (if the

objective's state is set to "successfully terminated"), then the activation rules of successors will be used to determine which successor will be chosen.

Resources and priorities
Resources and priorities are used to automatically monitor the exclusive execution of concurrent objectives at any iteration.

Objective types
As previously mentioned, objectives are divided into three types: simple, compound, and aggregate. Simple objectives are the most basic objectives. Their body is composed of a list of actions that are iteratively triggered when the objective is executed. In the demonstrator's behavior (Figure 6), "PresenceAct" and "CheckAround" are two simple objectives. Figure 10 shows the code of the "CheckAround" objective.

```
private simple objective CheckAround
        action perceiveCrowd()
        completion rules ACTIVE
end objective
```

Fig. 10. The "CheckAround" simple objective implementation.

Compound and aggregate objectives
A compound objective is a decomposable structure representing a sub-behavior. Figure 4 shows the internal view of the "PeacefullyDemonstrate" compound objective. The implementation and the execution of a compound objective are similar to a behavior. It consists in executing the sub-graph (the sub-behavior). An aggregate objective is another decomposable structure. But, contrary to a compound objective in which the inner objectives are linked together by successors, an aggregate objective is composed of a set of objectives without any direct relation between them. The execution of an aggregate objective triggers the execution of the objectives composing it which are in the active state.

3.2.2 Interactions between objects/agents and the VGE

Typically in a MAGS, the interactions between the objects/agents and the VGE are very frequent since an interaction is required every time an agent needs to query or act upon its environment. PLAMAGS includes various mechanisms, structures and an appropriate syntax to facilitate these interactions. Let us now present the main interactions.

Sending commands to the 3D environment
PLAMAGS provides a simple mechanism to send commands to the 3D environment that is which looks a classical method call. Collectively, these commands are called "perform" functions. The majority of the "performs" are what we call "managed actions" that are provided to ensure the spatial coherence of the virtual environment. The language offers movement and displacement actions such as "moveToward", moveNearAvoidObstacles", etc., allowing the displacement of components without worrying about the spatial constraints.

Information retrieval (feedback) from the 3D environment

We can specify many more interactions between simulation components and the VGE than just sending commands. Indeed, the agents and objects must also be kept aware of their situation in the environment so that they can behave as expected by the designer. Two types of information are generally required by an agent or an object from the VGE: its "spatial situation" within the VGE (its location, orientation, elevation, etc.) and its current perception of the environment.

Object/Agent spatial situation

The language introduces the keyword "percepts" whose sole purpose is to query the VGE regarding the "space related" attribute (position, orientation, elevation angle, etc.) whose name follows the keyword. This technique very effectively hides the complexity of the underlying operations.

Perception information retrieval

Agents and active objects are not only aware of their situation but are also aware (possibly partially) of the situation of other agents/objects located within the range of their senses. Retrieving this kind of information from the VGE is taken care of the keyword "references". Processing massive quantity of perceived facts can be a relatively inefficient and costly task. Hence, we split the perceived facts are into categories: i) agents and objects; ii) perceived gases; iii) groups; iv) projectiles. Retrieving perceptions by categories limits the number of facts passed to the simulation engine, and also coarsely filters facts that are obviously not relevant for an agent in its current context.

Configuring physical and spatial capabilities

In Section 1, we insisted on agents' space-related capabilities that a MAGS tool must automatically handle (perception, collision detection, obstacle avoidance, etc.). Most of these capabilities are directly integrated and configurable in the PLAMAGS language using what we call "*mapped items*".

Mapped items are simple language declarations specifying spatially/physically/visually related characteristics of agents and objects. Once these characteristics are specified, they are automatically managed by the simulation engine. Figure 8 shows how to specify some mapped items: the bounding volume and the field of view of the agent.

```
map boundingVolume : "automatic"
map fieldOfView : "20, 180, 0"
...
private void perceiveCrowd()
        local r : references = [references.Sight]
        for [i : int = 0;i < r.size(); i = i + 1] ... end for
end method
```

Fig. 11. Two "mapped items" and a method recovering perceived objects/agents.

In the demonstration simulation example, demonstrator agents have their own perception mechanism. To assign perception capabilities to a reactive or cognitive SSA, we only need to add a property "*fieldOfView*" with a radius and an angle in degrees to specify the angular

extent of the visual field. Once the *fieldOfView* property is set, it is possible to recover the elements perceived by the agent any time. Figure 11 shows the code to recover perceived elements.

The language offers more than 90 different features corresponding to most of the basic capabilities needed to easily and automatically handle the agents' physical interaction with the spatial environment. It is easy to use these features, and consequently the user can pay attention to the specification of the cognitive aspects of the SSA's behaviors without the burden of programming them and verifying their coherence at the spatial level. For instance, with the four properties of figure 12, we define 1) the 3D model to be used for the EVG (i.e. the site); 2) the system of coordinates of the environment (0-500 units on the X axis, 0-500 units on the Y axis, and 0-125 units on the Z axis); 3) the file in which the height map of the 3D model will be saved; and 4) the initial position of the camera. Figure 13 shows the resulting view.

```
19  map mapModel : "res/models/quebec_city.3ds"
20  map coordinates : "0, 500, 0, 500, 0, 125"
21  map heightMap : "res/maps/heightMap.map"
22  map camera : "106, 26, 85, 27, -88"
```

Fig. 12. Definition of four essential properties of the EVG.

Discussion
The keywords « perform », « percepts », « references » and « map » are powerful tools that allow simple and seamless interactions between the simulation's components (agents/objects) and the 3D environment. Using these keywords (and their associated functions), a component can send commands to the 3D engine, retrieve information about its own situation and about other component it perceives. Developers thus have all the required tools to integrate geographic information into the decision-making process of the agents.

Fig. 13. View of Quebec City obtained using the properties of figure 12.

4. The PLAMAGS IDE

PLAMAGS IDE is an application that aims to help developers in using PLAMAGS by combining all the tools required to design, implement and execute a PLAMAGS program into a single piece of software. Figure 14 shows a global view of the PLAMAGS IDE when a project is loaded. In "A", we can see the built-in text editor displaying *Commander.bdl,* a PLAMAGS source file. In "B", there is the Project Manager which allows browsing the project's files and opens them into the text editor. In "C", we have the context tree of the currently editing file which sums up the attributes, object and other structures defined in the file. The component in "D" is a tabbed pane showing the output of the compiler and the execution engine (when either is loaded).

4.1 Integrated Development Environment

The IDE provides all the tools that are necessary to allow developers to create applications efficiently using the PLAMAGS language. Features are: i) a project manager; ii) a text editor; iii) a visual file structure navigator; iv) a real-time syntax checker; v) a Java classes browser; vi) a graphical configuration tool (sets, for instance, the JDK, fonts, etc.); vii) a built-in versioning system. Figure 9 shows some of these features in action.

Type creator (using stubs)

The MAGS related structures can be defined with stub files that enable the instantiation of objects of the corresponding type. The *Type Creator* provides stub files' templates for static objects, active objects, agents, behaviors, atomic objectives, complex objectives, compound objectives, aggregated objectives, rule lists, as well as scenarios. These templates accelerate the development by sparing the developer the tedious and error prone part of writing a stub file.

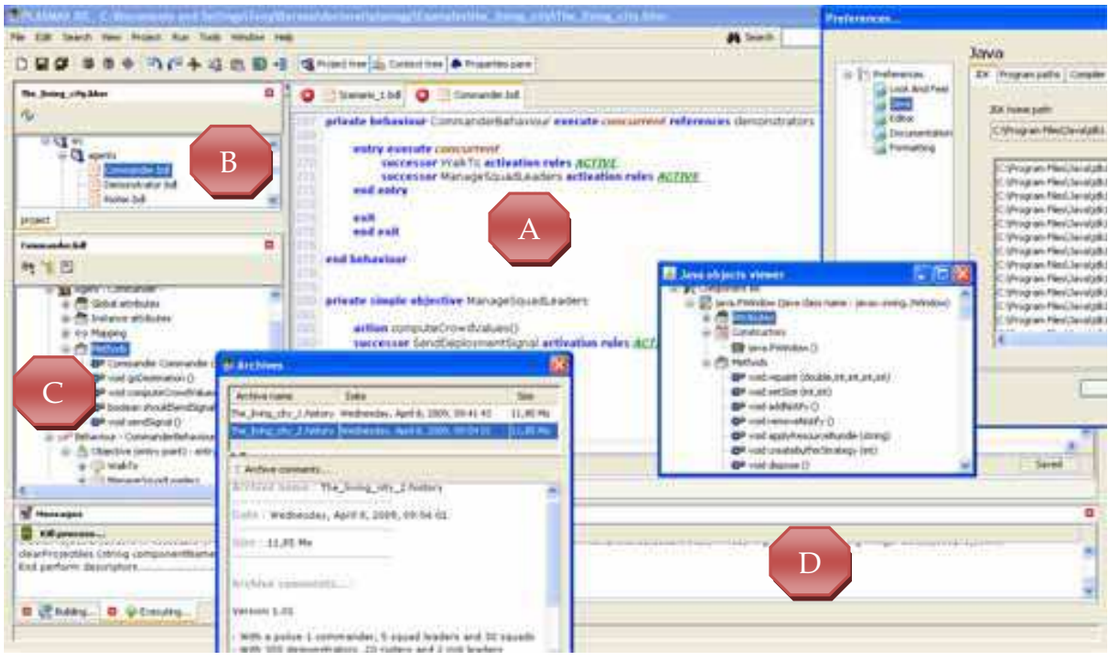


Fig. 14. Global view of the PLAMAGS IDE.

4.2 Language Support Suite

In addition to the development tools, PLAMAGS also provides tools to verify and execute the source files written in the language. These tools are either integrated in the IDE or provided as stand-alone applications.

Syntactic and Semantic Language Checker

As previously mentioned, the IDE includes language checker that verifies the content of every file of the current project in real-time and detects both syntactic and semantics errors.

PLAMAGS Language Interpreter

The Language Interpreter allows translating in real-time a PLAMAGS program into a set of Java instructions. The Java Runtime then takes over and evaluates each expression and executes the associated code when needed. The interpreter is responsible for the main following tasks: i) to evaluate the mathematical, boolean and relational expressions; ii) to create the structures specified using the language; iii) to evaluate and to dereference of attributes; iv) to execute method calls; v) to open the channels between Java and the execution engine; vi) to route the attribute query toward the requested object. However, when a behavioral structure has to be executed, the interpreter delegates this task to the Behavior Execution Engine.

Behavior Execution Engine

An execution engine might sound quite the same as an interpreter, but it is not the case. The main difference between the two is that the interpreter analyzes and evaluates conditional expressions whereas the execution engine executes pre-built structures such as rule lists and

behaviors. In fact, if the execution engine encounters a conditional expression, it halts and calls the interpreter to evaluate the expression and recovers the control afterwards.

The execution engine is totally separated from PLAMAGS behavioral features: it can be thought of as a plug-in that handles several tasks. It executes instructions, handles the transitions between components taking into account the constraints and maintains each agents' structures sound. This separation would allow for modifying or even changing the execution engine without any coupling problems.

4.3 3D Engine

The 3D engine bundled with PLAMAGS is a module whose main responsibilities are to render 3D scenes and to manage the consistency of spatial and physical relations between the objects which are part of the simulation.

For instance, it must ensure that agents and active objects do not go through buildings' walls or another agent body that the laws of physic prevail in the virtual world when collisions occur, etc. The computation of available perceptions and other properties influenced by the spatial situation of an agent are also handled by the 3D engine since such tasks require computing a large number of spatial constraints.

Considering that the visual 3D rendering and the management of spatial constraints and physics are complex tasks and often computationally demanding, the engine relies on two specialized external libraries, JPCT (JPCT 2009) to accelerate the computations and PhysX (PhysX 2009) to manage the physics in the virtual world.

4.4 Simulation's Control and Visualization Interface

In order to simplify the execution of MAGS, the IDE includes a tool to visualize and to control the execution of the simulations.

4.4.1 Direct Control

As we can see in Figure 15, the user interface allows a user to control a running simulation. The simulation can run continuously (launched by the 'Run' button) or step by step. These steps are: i) loading the 3D map on screen; ii) instantiate the scenario; iii) initialize the scenario; iv) start the simulation; v) pause the simulation; vi) continue the simulation; vii) execute next step, etc.

4.4.2 Custom Interfaces for Agents

PLAMAGS also offers the possibility to have a personalized execution interface by cfrating a personalized window in which the simulation can be displayed. It is also possible to add *listeners* that are used to interact with the simulation by exchanging 'events' with the simulation.

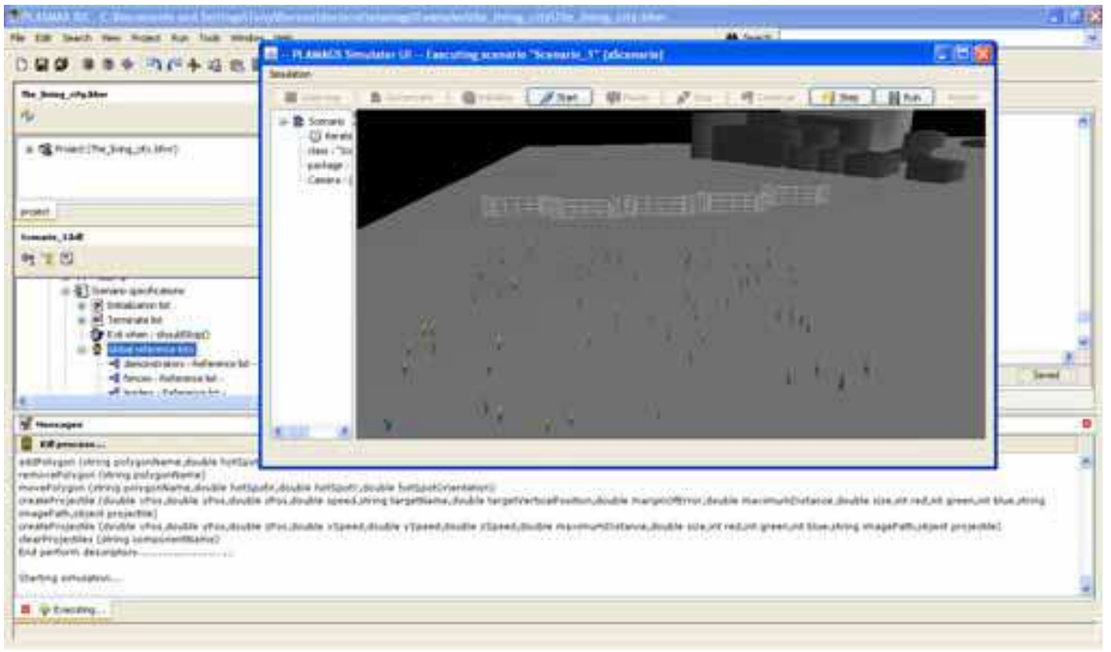


Fig. 15. Simulation interface running through the IDE.

Let us emphasize that the language checker, the interpreter, the execution engine, the libraries (3D and physics) and the visualization interface are all integrated in the IDE, but that they all can be used outside of it as well.

5. PLAMAGS features

In this section we discuss some features that contribute to the expressiveness, the simplicity and the effectiveness of PLAMAGS.

5.1 Graphical specification of graph and translation into PLAMAGS

The last component of PLAMAGS IDE which is still under development is an interface that will allow to greatly simplify the development and understanding of behavioral graphs (Figure 16). The Behavior Creator will allow a user to visually create and bind the various objects that together constitute a behavioral graph, as well as to specify the properties of an objective and the rules used by this objective.

When finished, the Behavior Editor will allow users to define agent behaviors in a simple, intuitive and visual way, thus effectively avoiding the syntactic and structural complexity that underlines the definition of multi-layer directed graphs. Furthermore, users will be allowed to shift back and forth between the source code and the visual representation instantly.

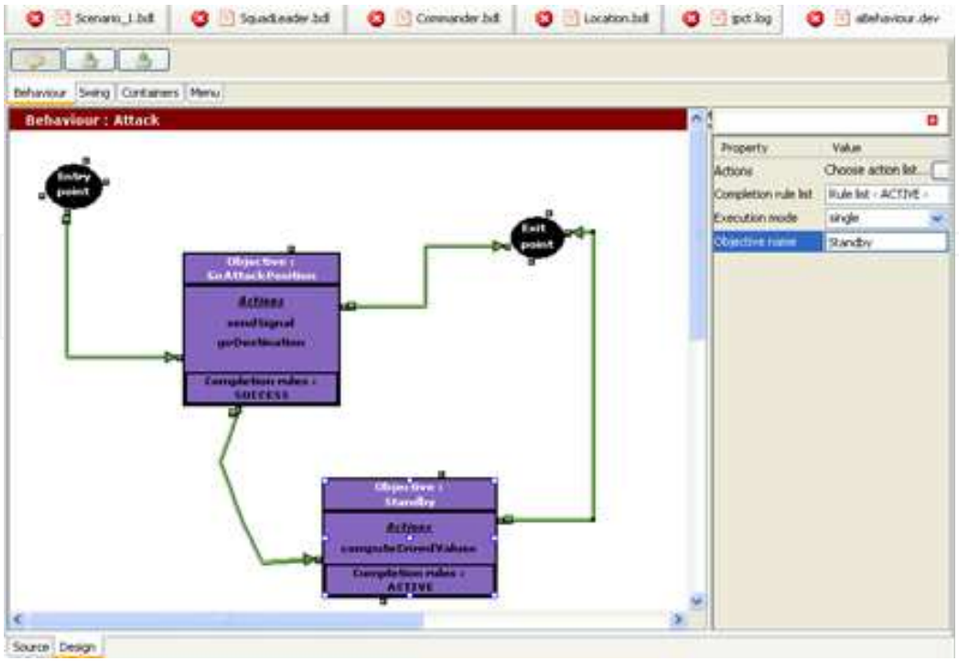


Fig. 16. Visual tool for behaviour creation.

It seems rather obvious that the visual representation of a behavioral graph is easier to understand than its textual equivalent since the visual part, the graph, even blank, carries semantics that one should write in a textual context. The difference with respect to complexity only grows as the size of this graph rises.

5.2 Compiled execution Mode

As we previously mentioned, PLAMAGS was originally designed as an interpreted language, which proved to be too inefficient for development and execution of large MAGS. This motivated the decision to add to the IDE a tool that compiles the interpreted code and then executes it. This tool is not completed yet; it currently supports simulations in which the user can specify active objects, scenarios and the use of gas. The part which would compile agent specifications is still under development.

The compiling technique that this tool implements is composed of three steps: first, the Java source code is generated from the PLAMAGS definition; second, the Java code is compiled; and third, the compiled code is executed using an execution engine. Splitting the technique in such steps enables the user to verify and modify the Java code before it is complied. This technique is a definitive solution to any eventual shortcomings of the language since the injected instructions are not subject to any limitations or restrictions. We empirically compared the interpreted and complied execution engines: the compiled engine was over 300 times faster than the interpreted one.

5.3 PLAMAGS' extensibility

Although in most situations, it is possible to develop simulations using PLAMAGS as a stand-alone development tool. However it might be sometimes desirable, if not necessary, to interact with external software such as external libraries or to use some code written in Java. We now introduce the techniques to so in both directions.

Using external Java classes and libraries

From the onset of the PLAMAGS Project, we designed the system so that communication between Java and PLAMAGS would be bidirectional. However, accessing PLAMAGS from Java should be allowed in specific cases.

PLAMAGS allows the use of any Java class and the technique is the same regardless of where the class comes from (JDK, libraries, folders). Each external class used in PLAMAGS requires to be assigned a name specified in a dedicated section of its source file. This name is used to reference the class in the PLAMAGS source code. Nothing prevents the use of the actual name of the class as its PLAMAGS name. Then these "new" types of static objects (along with their java properties and methods) are directly usable in the language, seamlessly using PLAMAGS syntax.

Accessing the simulation from Java

It is also possible to reference a PLAMAGS simulation from a Java application. To ensure a simple and safe interaction, the communication must go through what we call an *Externalizor* which exposes the relevant functions while preventing reckless interactions. The externalizor allows for sending "perform" commands to the 3D engine, accessing the components and the scenario of the simulation. This feature is used to carry out any operation outside PLAMAGS, as needed.

6. Conclusion

To sum up, the PLAMAGS environment provides: 1) a program editor (with real-time error checking); 2) a project management tree; 3) a contextual tree (describing the components of the file); 4) a language validation engine (similar to a compiler); 5) a runtime engine (an interpreter); 6) a 3D engine to visualize the simulations; 7) a visual programming tool (to graphically develop behaviors). In addition, it provides a Java objects browser for mapped types, an integrated help mechanism, a simulation start-up and a Java runtime configuration. It is executable both under Windows and Linux.

As previously mentioned, most of the multiagent geosimulation tools used to specify agents' behaviors are either strongly centered on the specification of behaviors' spatial aspects, or based on models inspired by finite state machines, which usually leads to create reactive agents. Similarly to tools which have "navigation or spatialized driven" behaviors, PLAMAGS offers a lot of predefined navigation actions. Although we consider spatialized and navigation behaviors as an important part of a geosimulation, they do not suffice to develop advanced multiagent geosimulations where agents possess cognitive capacities. To address this need, we introduced behaviors using multi-layered directed graphs. Contrary to other models inspired by finite state machine, PLAMAGS behavior's graphs manage concurrent execution and multiple concurrent, infinite decomposition of agents' goals (sub-

behavior layers), expressive and powerful transitions between nodes proceeding into two phases: activation and completion (using rule lists and priorities), execution control of objectives using resources, priorities and execution rules. Behavior-related structures are at the very heart of PLAMAGS' architecture and language.

In addition to offering a language and an IDE, PLAMAGS provides a configurable tool allowing tracing the execution of simulations. The tool even allows a designer to inspect step by step the execution of the behavior runtime engine and to get details about its decision choices.

We used PLAMAGS in several simulation projects and have identified three main shortcomings for which we intend to find solutions in our short term future work. First, the definition of a simulation's initial properties, VGE, objects and agents, for a specific scenario, is a demanding task when performed at the programming level. A graphical specification tool would facilitate the task from the user's perspective. Second, the JPCT library seems to reach its performance limits when confronted with complex 3D models. Indeed, such models require too much time for graphical rendering and tend to use too much memory. This is why we use very simple models for the representation of objects and agents. We may try to find a replacement library. Third and last, our numerous experiments involving various simulations have shown that it would be very convenient to add a "save simulation state" function in order to save a simulation, modify one or more elements of its structure, and then resume the simulation in the state saved before the changes.

We are currently developing the visual specification interface for behaviors. Once this module will be completed, the user will be able to specify agent behaviors in a visual programming mode. Thereafter, it will be possible to automatically generate the corresponding PLAMAGS code. The user will also be able to do the opposite operation and go from the textual programming mode to the visual one. We are currently extending the compiled version of PLAMAGS so that it can support agents and their behaviors. We are also investigating the possibility to integrate some behavior validation mechanisms using graph theory.

Acknowledgements: Parts of this project have been financed by Geoide, the Canadian Network of Centers of Excellence in Geomatics. The first author also benefited from a scholarship from the Natural Sciences and Engineering Council of Canada.

7. References

AI.implant (2003). AI-implant: A game-AI derived general scalable model for life-form simulation in MOUT-based applications.

AI.implant. (2009). from http://www.presagis.com/.

Beneson, I. and V. Kharbash (2005). Geographic Automata Systems:From The Paradigm to the Urban Modeling Software. AGILE 2005 and GISPlanet 2005, Estoril, Portugal.

Beneson, I. and P. M. Torrens (2004). "Geosimulation: Automata-Based Modeling of Urban Phenomena."

Bourrel, E. and V. Henn (2003). Mixing micro and macro representations of traffic flow: a hybrid model based on the LWR theory. 82nd TRB Annual Meeting (Transportation Research Board), Washington, USA.

CORMAS. (2009). from http://cormas.cirad.fr/.

Courdier, R., F. Guerrin, F. H. Andriamasinoro and J.-M. Paillat (2002). "Agent-based simulation of complex systems: application to collective management of animal wastes." Journal of Artificial Societies and Social Simulation **5**(3).

Crooks, A., C. Castle and M. Batty (2007). Key Challenges in Agent-Based Modelling for Geo-Spatial Simulation. Geocomputation 2007, National Centre for Geocomputation (NCG), National University of Ireland, Maynooth, Co. Kildare, Ireland.

Cutumisu, M., D. Szafron, J. Schaeffer, M. McNaughton, T. Roy, C. Onuczko and M. Carbonaro (2006). "Generating Ambient Behaviors in Computer Role-Playing Games." IEEE Intelligent Systems **21**(5): 19-27.

d'Aquino, P., C. Le Page, F. Bousquet and A. Bah (2003). "Using Self-Designed Role-Playing Games and a Multi-Agent System to Empower a Local Decision-Making Process for Land Use Management: The SelfCormas Experiment in Senegal." Journal of Artificial Societies and Social Simulation **6**(3).

Donikian, S. (2001). HPTS: a behaviour modelling language for autonomous agents. International Conference on Autonomous Agents, fifth international conference on Autonomous agents, Montréal, Québec, Canada, ACM Press.

Fagiolo, G., A. Moneta and P. Windrum (2007). "A Critical Guide to Empirical Validation of Agent-Based Models in Economics: Methodologies, Procedures, and Open Problems." Computational Economics **30**(3): 195-226.

Foudil, C. and D. Noureddine (2007). "An autonomous and guided crowd in panic situations." Journal of Computer Science & Technology **2**(2): 134-140.

Fu, D., R. Houlette and S. Henke (2002). "Putting AI in Entertainment: An AI Authoring Tool for Simulation and Games." Intelligent Systems **17**(4): 81-84.

Fu, D., R. Houlette, R. Jensen and S. Henke (2003). A Visual Environment for Rapid Behavior Definition. 2003 Conference on Behavior Representation in Modeling and Simulation, Scottsdale, Arizona.

Garneau, T., B. Moulin and S. Delisle (2008). PLAMAGS: A Language and Environment to Specify Intelligent Agents in Virtual Geo-Referenced Worlds. Proceedings of the 19th IASTED International Conference on Modelling and Simulation., Quebec City, Canada.

GASP. (2009). from http://www.irisa.fr/prive/donikian/behavioral_programming_environment.html

Gnansounou, E., S. Pierre, A. Quintero, J. Dong and A. Lahlou (2007). "Toward a Multi-Agent Architecture for Market Oriented Planning in Electricity Supply Industry." International Journal of Power and Energy Systems **27**(1): 82-91.

Guyot , P. and S. Honiden (2006). "Agent-Based Participatory Simulations: Merging Multi-Agent Systems and Role-Playing Games." Journal of Artificial Societies and Social Simulation **9**(4).

Helbing, D., A. Hennecke, V. Shvetsov and M. Treiber (2002). "Micro- and Macro-Simulation of Freeway Traffic." Mathematical and computer modelling **35**(5-5): 517-547.

JPCT. (2009). from http://www.jpct.net/.

Koch, A. (2001). Linking Multi Agent Systems And GIS - Modeling And Simulating Spatial InterActions –. Angewandte Geographische Informationsverarbeitung XII, Beiträge zum AGIT-Symposium.

Levesque, J., F. Cazzolato, J. Perron, J. Hogan, T. Garneau and B. Moulin (2008). CAMiCS: civilian activity modelling in constructive simulation. SpringSim 2008.

Moulin, B., W. Chaker, J. Perron, P. Pelletier, J. Hogan and E. Gbei Fonh (2003). MAGS Project: Multi-Agent GeoSimulation and Crowd Simulation. Conference on Spatial Information Theory (COSIT'03), Ittingen, Switzerland, Springer-Verlag.

Müller, J.-P., C. Ratzé, F. Gillet and K. Stoffel (2005). Modeling And Simulating Hierarchies Using An Agent-Based Approach. MODSIM05 : International Congress on Modelling and Simulation. Advances and Applications for Management and Decision Making Melbourne, Australia.

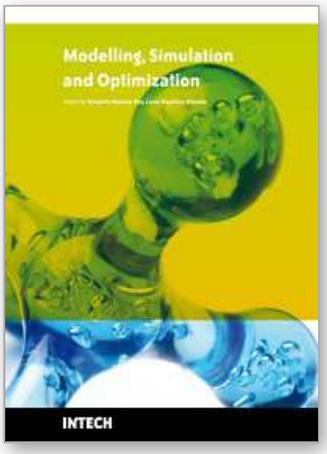
Papazoglou, P. M., D. A. Karras and R. C. Papademetriou (2008). On the Multi-threading Approach of Efficient Multi-agent Methodology for Modelling Cellular Communications Bandwidth Management Agent and Multi-Agent Systems: Technologies and Applications. **4953/2008**.

PATHEngine. (2009). from http://www.pathengine.com/.

PhysX. (2009). 2008, from http://www.nvidia.com/object/physx_9.09.0408_whql.html.

SPR.OPS. (2009). from http://www.spirops.com/.

Torrens, P. M. and I. Benenson (2005). "Geographic Automata Systems." International Journal of Geographical Information Science **19**(4): 385-412.



Modelling Simulation and Optimization

Edited by Gregorio Romero Rey and Luisa Martinez Muneta

ISBN 978-953-307-048-3
Hard cover, 708 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

Computer-Aided Design and system analysis aim to find mathematical models that allow emulating the behaviour of components and facilities. The high competitiveness in industry, the little time available for product development and the high cost in terms of time and money of producing the initial prototypes means that the computer-aided design and analysis of products are taking on major importance. On the other hand, in most areas of engineering the components of a system are interconnected and belong to different domains of physics (mechanics, electrics, hydraulics, thermal...). When developing a complete multidisciplinary system, it needs to integrate a design procedure to ensure that it will be successfully achieved. Engineering systems require an analysis of their dynamic behaviour (evolution over time or path of their different variables). The purpose of modelling and simulating dynamic systems is to generate a set of algebraic and differential equations or a mathematical model. In order to perform rapid product optimisation iterations, the models must be formulated and evaluated in the most efficient way. Automated environments contribute to this. One of the pioneers of simulation technology in medicine defines simulation as a technique, not a technology, that replaces real experiences with guided experiences reproducing important aspects of the real world in a fully interactive fashion [iii]. In the following chapters the reader will be introduced to the world of simulation in topics of current interest such as medicine, military purposes and their use in industry for diverse applications that range from the use of networks to combining thermal, chemical or electrical aspects, among others. We hope that after reading the different sections of this book we will have succeeded in bringing across what the scientific community is doing in the field of simulation and that it will be to your interest and liking. Lastly, we would like to thank all the authors for their excellent contributions in the different areas of simulation.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China

51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen

IntechOpen