12

PLAMAGS: A Unified Framework and Language for Efficient Multi-Agent Geo-Simulation Development

Tony Garneau and 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

The micro-simulation of social and urban phenomena using software agents in geo-referenced virtual environments is a field of research whose popularity has strongly grown recently. Geo-simulation (Benenson and Torrens 2004) is 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 (Fagiolo et al. 2007), political, industrial (Gnansounou et al. 2007), medical, social, etc.) and it is thus used to study various complex phenomena, especially in the domain of urban dynamics (Foudil and Noureddine 2007) and land cover planning.

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 (Benenson 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, 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 HPTS (Donikian 2001), AI.Implant (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 mechanisms to specify objects/agents behaviors based on models inspired by finite state machines. 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 geo-simulations of social phenomena in which agents need to implement knowledge-based capabilities in relation to the space in which they evolve. In both cases, the 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 objects/agents 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. These agents, that we call "spatialized agents", 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 fully specify spatialized agents. In this chapter we present the PLAMAGS Project in which we developed an agent-oriented language, a development environment and a 3D visualization engine completely dedicated to the development and the execution of multi-agent geo-simulations (MAGS), involving spatialized agents.

Section 2 presents the PLAMAGS methodology that we propose to develop MAGS. Section 3 introduces the architecture and the main concepts on which the PLAMAGS language is based. Section 4 presents the main elements that are composing a PLAMAGS simulation. Section 5 discusses the characteristics of the language and its IDE and Section 6 concludes the paper with a discussion and some future work.

2. PLAMAGS method

Contrary to the majority of existing MAGS and MABS methods which generally put the emphasis on modeling and design, our PLAMAGS method is tightly linked to the specification language that was designed to support each step of the development cycle, using a syntax that is both declarative and procedural. The user can thus easily carry out the modeling, the implementation, the execution and the validation of the simulation within a single framework. Our method is composed of 12 steps shown in the Figure 1.

The PLAMAGS method proposes a generic and progressive approach which aims at supporting a designer when creating a MAGS, which is specified and tested in an

incremental way, thanks to the PLAMAGS Development Environment and the associated language.

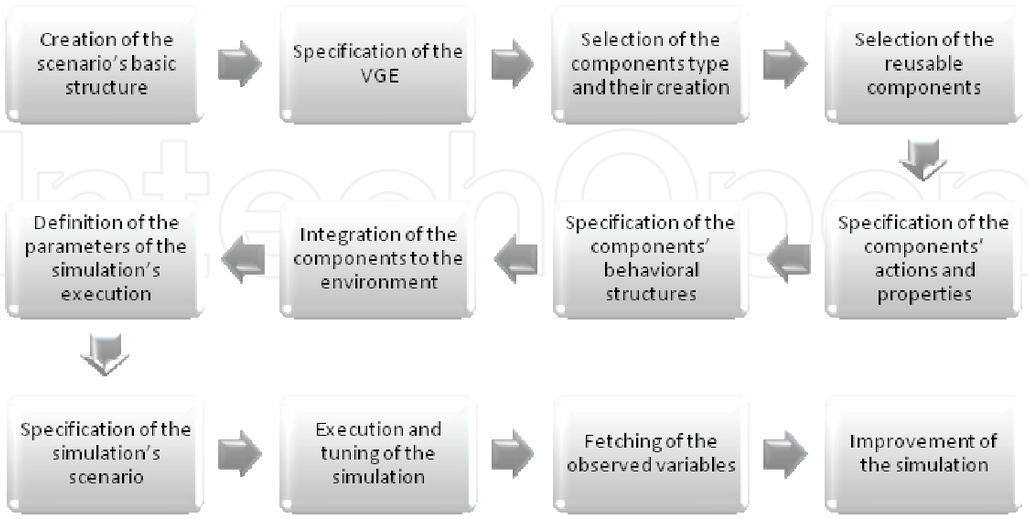


Fig. 1. Overview of the PLAMAGS' method.

As soon as the designer has specified the elements required at a given step of the methodology, he can implement these elements using the PLAMAGS language. Then, the PLAMAGS environment allows him to execute this partial simulation, so that the designer can observe the results and eventually detect any anomaly/error resulting from the specification. Indeed, he can make the required corrections and run the partial simulation again. In that way, the designer can readily get partial results of the simulation under construction, and make all the adjustments that are required to make sure that the specification is correct and yields the expected results. Each step of the method is supported by a set of statements of the PLAMAGS language and by specific components of the Development Environment. The method supports a 2-level iterative design process during which it is possible to come back to the previous steps (or sub-steps) when needed.
Each of the method steps leads the user to carry out an action sequence which aims at getting a partial simulation result, when completed. Figure 2 presents the action sequence (of sub-steps) that applies to most of the steps of Figure 1. These sub-steps can also be carried out in an iterative way and can be refined by the user if needed.

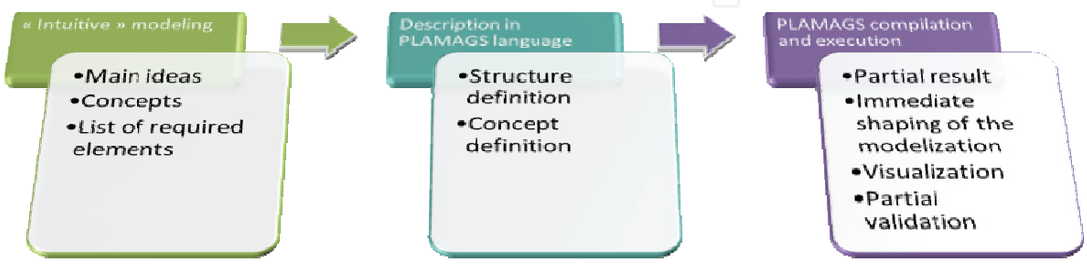


Fig. 2. Sub-steps that apply to each step of the method.

For example, in a first iteration of the development process, at the VGE specification step (second step in Figure 1), the user might carry out the following actions.

- "Intuitive" modeling: the user chooses the place where the simulation will occur, the dimensions of the virtual geographic environment (VGE), etc.

- Description in PLAMAGS language: the designer uses the language to create a scenario and specifies the 3DS model of the ground, the textures, the system of coordinates, the dimensions, the model orientation on the screen, etc.

- PLAMAGS compilation and execution: the user compiles and executes this initial scenario and visually validates into the simulator that the VGE model is suitable to the simulation, that it has been positioned correctly, that the proportions are realistic, etc.

2.2 Advantages of the PLAMAGS approach

The PLAMAGS approach (Method + Language + Development Environment) allows the designer to get a partial result that can be executed at every step of the development process, which brings many advantages compared to other theoretical or conceptual methods. Here are some of them:

- Early detection of modeling or design mistakes
- Validation of the simulation result at every step
- Transparency between the conceptual and the implemented models
- Easy approach and quick materialization
- Progressive development and refinement of the simulation
- Flexibility in the definition

Developers are often forced to use simulation development tools that do not directly support the concepts that are defined in the conceptual models, which inevitably leads them to modify the models in order to implement them. Indeed, such modifications lead to an additional work-load and increase the risks of introducing errors and discrepancies with the conceptual models. Moreover, even when the tools allow for a direct translation of the specification to the simulation code, the fact that the code is written in a general programming language that is not dedicated to the simulation, the implementation task is slow, complex and propitious to mistakes of various kinds. The great advantage of the PLAMAGS Approach is that it provides a unique and complete language for specification, definition, implantation and execution of the models, and consequently eliminates the translation process between the theoretical/conceptual models and their implementations.
Compared to other methods, the PLAMAGS two-level process allows the user to make sure that the partial model is implemented, tested and validated at the end of each step. This allows him to progressively carry out the model validation during its creation. This kind of validation is usually impossible when using other simulation design methods.

3. PLAMAGS' Architecture

This section presents a synthesis of the relations that exist between the different elements that we use in the development cycle of MAGS. We will present the general principles of the PLAMAGS framework and show how they are related.

A fact that greatly contributes to make a MAGS development a challenge is the necessity to work with two very distinct sets of concepts expressed either using a Geographic Information System (GIS) or a Multi-Agent Based Simulation, which do not address the same modeling problems. Thus, we need to conciliate these differences in order to simplify the work of designers and developers. To this end, the PLAMAGS Approach proposes a way to easily model, specify and implement the Virtual Geo-referenced Environment (VGE) as well as the links and interactions between the agents and the VGE. Indeed, the PLAMAGS' architecture deals with the following issues: i) the characterization of the relations between the Geographic Information (GIS) and the multi-agent based simulation (MABS); ii) the description of the VGE within the simulation; iii) the interaction between the VGE and the simulation's components (objects and agents); iv) spatial and physical consistency; v) sensorial capacities of the agents and objects; vi) the VGE's influence on the agents and objects during the simulation.
As a matter of fact, Figure 3 illustrates the principles on which the PLAMAGS Approach is based. As illustrated, the main components of a PLAMAGS simulation's model are: 1) a VGE that renders the simulation environment; 2) the agents and objects located in this VGE (which are characterized by specific behaviors); 3) the simulation scenario and, finally 4) the results (or outputs) of the simulation. These four elements are in constant interaction and constitute the core of the system.

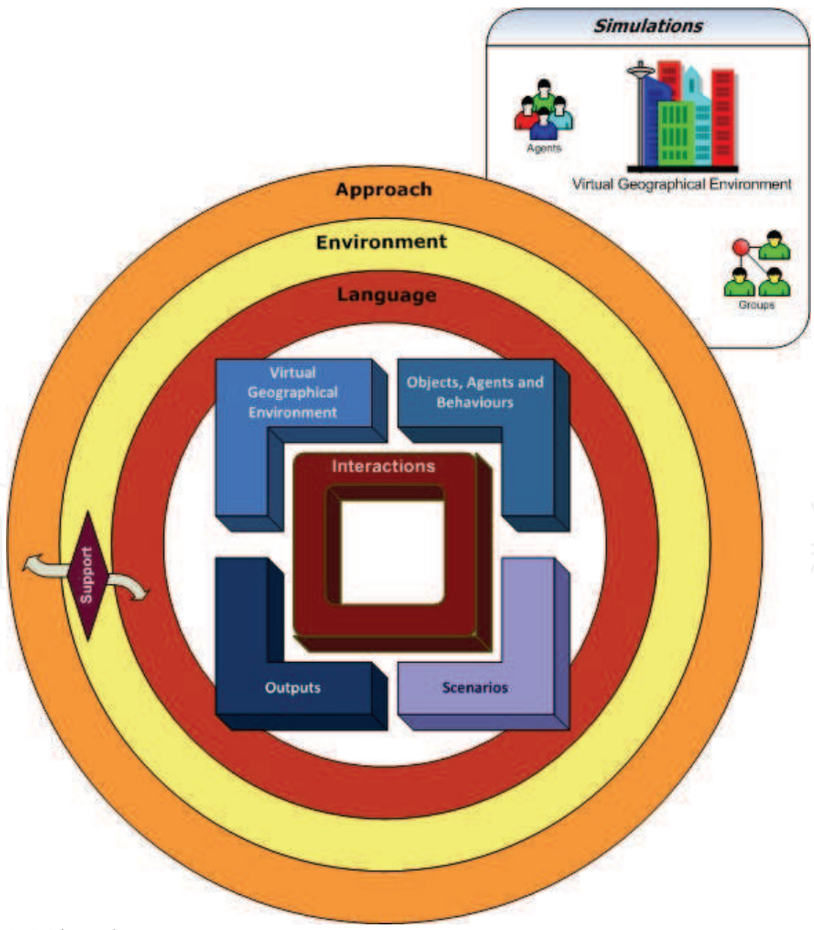


Fig. 3. PLAMAGS' architecture.

However, at a more global level, the PLAMAGS' architecture can be thought of as being composed of two distinct parts: 1) the programming language (Garneau et al. 2008) fully dedicated to the specification, definition, execution and the deployment of MAGS and all its elements; and 2) a set of tools to facilitate its use. These tools are bundled in 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.
PLAMAGS also offers a powerful behavioral model that is modular and extensible and is used to specify the agents' behaviors (Garneau et al. 2010). Similarly to tools which have "navigation or spatial driven" behaviors, PLAMAGS offers a lot of predefined navigation actions. In addition to basic navigation, PLAMAGS introduces behaviors using multi-layered directed graphs that, contrary to other models inspired by finite state machines, manage the concurrent execution and multiple concurrent states (using "on the fly" context addition and withdrawal), infinite decomposition (sub-behavior layers), expressive and powerful transitions between nodes proceeding into three phases: activation, execution and completion (using rule lists, resources and priorities). Section 4.2 presents the main characteristics of the behavioral graphs.

3.1 Materialization of the geo-spatial aspect

The model presented in Figure 3 features a close bond between the VGE and the agents (as well as their behaviors). One of the main challenges at this level is to define in a simple and legible way for users who are not expert at manipulating GIS, the structures appropriate to describe the characteristics and properties of the VGE, while offering an acceptable level of details to create a complete VGE.
The PLAMAGS Approach simplifies the specification of the VGE thanks to dedicated constructs of the language and functions of the IDE. Again such an approach provides several advantages:

- To make available the structural and spatial composition of the VGE to the agents' decisional process
- To integrate the definition of characteristics of the VGE at the scenario level
- It allows agents to interact with the VGE during the simulation development
- To extract comprehensive spatialized information ("outputs", even for beginners in GIS).

In order to be able to integrate and use the spatial and physical characteristics of the VGE in a simulation, it is necessary that the properties of the VGE be defined in an intuitive way (in formats that are legible by everyone) and that these properties be directly usable during the execution of the simulation without needing that a developer carry out any transformation, conversion or complex calculations.
Practically, a geographical environment possesses an infinite number of characteristics and properties. It is thus unthinkable to exactly model it in the VGE at the simulation level. One must rather facilitate the specification of the VGE's characteristics by simplifying and selecting the appropriate features of the environment.
The integration and management of the geographical environment in the PLAMAGS simulation model are done at different levels. First, the visual, spatial and physical characteristics of the geographic environment as well as the objects/agents must be defined (section 3.1.1). Then, it is necessary to characterize the interactions and the relations between the VGE and the objects/agents of the simulation (section 3.1.2). Finally, when considering

the agents' behaviors, the interactions and the feedback coming from the environment must be taken in account (section 3.1.3).

3.1.1 Definition of visual, spatial and physical properties

The first step of the VGE's specification consists in defining the properties and characteristics of the geographical environment and of the objects/agents that will be used by the simulation. We can distinguish three categories: the visual properties, the spatial properties and the physical properties.

Visual properties

The visual properties allow the designer to configure the visual rendering of the simulation (simulation display). Several of these properties define pointers either to files containing the 3D structures (using the 3DS format) of the various components, or to images, as well as to their textures and colors ("png", "jpg" or other formats).

Although the visual properties have a limited influence on the development of the simulation, they are essential to display the simulation results at the execution time. Visualizing the simulation results is the first feedback that a user gets from its specification and that enables him to inspect the simulation output. The visual rendering also allows a developer to carry out a series of quick validations (checking the components present in the simulation, correcting positions, checking specific actions occurring when a specific object is perceived by an agent, etc.).

Spatial properties

Moreover, the spatial properties allow the designer to define the logical basis of the VGE structure and of the simulated objects/agents. Among these properties, let us mention the measuring units, the dimensions, the volumes, and the objects'/agents' positions in the VGE. These spatial properties relate the geographical environment to the simulated objects/agents and are used by their decision making processes (at the behavioural level).

Physical properties

The physical properties (gravity, components' weight, friction, etc.) allow for keeping a certain level of coherence and of reality in the management of the spatial interactions between the objects/agents and the VGE. For example, the physical properties can be used to determine the effects of a collision between two components, the effect of a movement of the object/agent on a steep ground, etc. They are also used to simulate the propagation of gases. The management of the physics is an extremely complex mechanism which is demanding in terms of calculations. This mechanism is managed in the PLAMAGS environment with the help of a very powerful external library called PhysX (PhysX 2010) which is used in many video games.

Using these visual, spatial and physical characteristics allow the designer to create a simplified definition, yet representative, of the geo-referenced environment, of the objects and agents as well as of their relationships.

3.1.2 Definition of the agents' sensory capacities

Once the visual, spatial and physical properties have been defined for the objects, the agents and the VGE, the designer only needs to define the properties corresponding to the sensory capacities of the objects and agents in order to be able to get back the perception information (these properties correspond to a subset of the visual, spatial and physical properties described in the previous section). Figure 4 presents some of the main properties allowing the designer to define the sensory capacities of an active object or an agent.

The "perceivable" property specifies that the agents that possess it can be perceived visually by other agents. The "perceiveSelfMouvements" and "perceiveSelfAltitude" properties offer to the agent the capacity of getting upon request the information relative to its location and its "comfort zone" described in the middle section of figure 5. The five last properties define the capacities of perception of the agent (described in block 1 of figure 5). For example, "fieldOfView 200,180,10" means that the agent perceives elements located in a circle of 200 distance units (with respect to the coordinates defined in the VGE) and that its sector of perception is of 180 degrees in front of it.

```
19    map perceivable : true
20    map perceiveSelfMovements : true
21    map perceiveSelfAltitude : true
22
23    map fieldOfView : "200, 180, 10"
24    map perceiveGroups : true
25    map gasFieldOfView : "100,180"
26    map perceiveGas : true
27    map personalSpacePerceptionDistance : 0.5
```

Fig. 4. Properties allowing to define some sensory capacities.

3.1.3 Application of geo-spatial and physical concepts in the simulation

Once the sensory capacities have been defined, it is possible to manipulate them directly and to integrate them in the agents' decision making process.

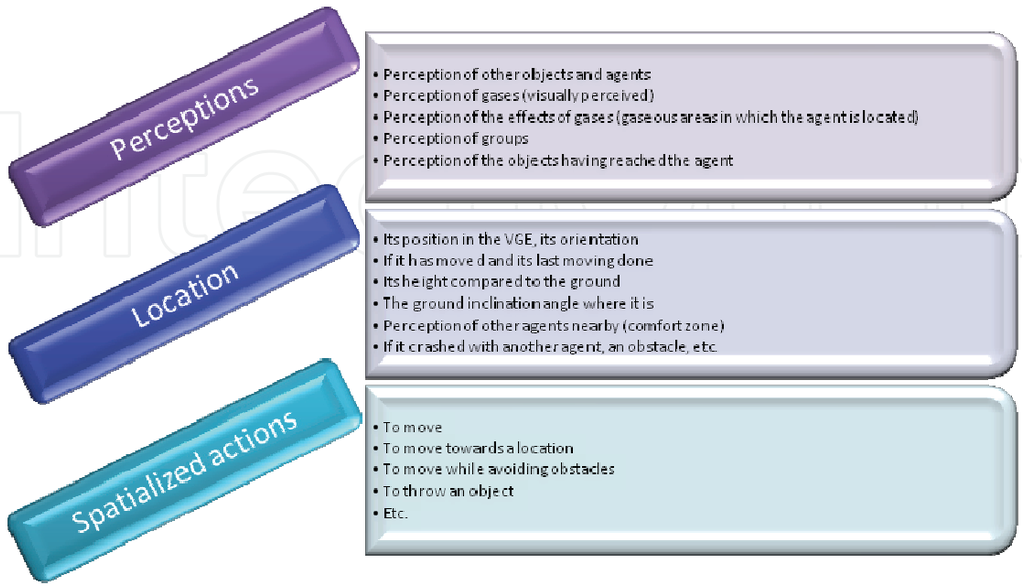


Fig. 5. Summary of the main interaction mechanisms between the agents and the VGE

To facilitate these manipulations and support properly the integration of the geo-spatial aspect into the simulations, the PLAMAGS allows the explicit definition of the spatialized interactions between the agents and the VGE. To this end, the different interaction mechanisms are directly integrated in the language and available through the use of 3 key-words*: percepts, references and perform.* Indeed, the interactions are sorted by categories, each of them grouping a collection of functionalities/capacities semantically related. Figure 5 sums up the spatialized interaction mechanisms defined in the language.

Perceptions
An agent can get its perception information at any time during the simulation run (this partial knowledge of the environment can be used by the decision process). The perceptions are sorted in five categories that can each be treated in an independent way. Figure 6 shows the way to get back the objects/agents perceived by an agent. The system only needs to access the list of perceived objects/agents (using *references.Sight*). Each of the components is accessible afterward.

```
public void perceiveAgents()

    local objects : references = [references.Sight]
    local objectList : array<object> = [objects.toArray()]
    local obj : object

    for [i : int = 0;i < objectList.size();i = i + 1]
        set obj = [objectList.get(i)]
        ...
    end for

end method
```

Fig. 6. Retrieval and manipulation of components perceived by an agent.

Location
The language also allows to obtain an agent's geographic location. (Figure 7)

```
public void printSelfInfo()

    call println("xMovement: " + percepts.xMovement)
    call println("zRotation: " + percepts.zRotation)
    call println("elevationAngle: " + percepts.elevationAngle)
    call println("orientationAngle: " + percepts.orientationAngle)
    ...

end method
```

Fig. 7. Retrieval of information about the spatial situation of the agent.

Given its perception list and its own spatial situation, an agent can determine which agents are located around it, as well as the objects located in the environment in which it evolves. This knowledge is called "location knowledge".
The location knowledge also allows for the definition of a minimal zone (comfort zone) that is necessary for an object/agent to exist in the VGE. If another object/agent gets into this zone, then some forces are automatically applied to attempt pushing back the intruder

(using the PhysX functions). The applied forces and their effects are configured by the parameters defined in the location knowledge of the objects and agents implied in the spatial conflict. Such a capability allows for the automation of the proximity management of objects/agents (figure 8), in addition generating more realistic spatial micro-behaviors (for example, the movements of an agent making his way through a crowd).

Fig. 8. Use of the confort zone (debug view)

Spatialized actions

The spatialized actions allow agents to spatially interact with the VGE and with other agents. Figure 9 shows how an agent can perform a 5-degree rotation and do a movement toward position 134, 158 at the speed of 3.6 units per iteration.

```
public void move()

    perform relativeRotation(5)
    perform moveToward(134,158, 3.6,true)

end method
```

Fig. 9. Use of spatialized actions

Availability and usability of the interactions model

The PLAMAGS interaction model is clearly defined and completely integrated with the language syntax: it is not necessary to set specific configurations, neither to use a complex syntax, nor to make use of an external module. The direct invocation of these mechanisms through the language allows a user to easily integrate the geo-spatial information in the agents' decisional processes. As an example, when a rioter in a demonstration sees a policeman approaching, he must decide either to continue to throw objects or to run away.

3.2 Geo-spatial and physical coherence

The mechanism responsible for managing the spatial and physical coherence of the simulation consists in controlling, in a transparent and automatic way, the spatial and physical state of each object/agent. With the help of the PhysX Physical engine, the PLAMAGS execution engine ensures that the spatialized actions carried out by each of the objects/agents are available and valid. For example, if an object/agent attempts to move in a given direction, and if there is a risk of collision with an obstacle (a building wall, another object/agent, etc.), then the execution engine must calculate the present forces (friction, weight, etc.) and correct the "desired" movement of the component and transfer it to another acceptable position which respects the spatial and physical restrictions.
The execution engine is also responsible for producing the edge effects associated with the physical forces which affect the objects/agents. For example, if a collision occurs with an object/agent at a location where the ground is characterized by a certain slope, then dependently of the physical properties of the ground and of the object/agent (gravity, friction, weight, etc.), it is possible that the colliding object/agent will "start to slip". In this case, the execution engine (once again with the help of the Physical engine) is responsible for managing the unwanted movement of the object/agent and to notify it of its position change, at each iteration.

4. Composition of a simulation

This section presents an overview of the elements composing a PLAMAGS simulation (see Figure 3). Generally speaking, a PLAMAGS simulation is composed of a VGE (mainly the 3D model, the coordinates system, the structures containing the elevation maps, physical forces such as gravity), the objects and agents that evolve in the VGE, each of them having its particular capabilities, either visual, spatial or physical, that allow it to carry out its specific behaviors and to evolve in the VGE. In addition, a simulation includes a scenario which defines the initial state of the simulation as well as how it will unfold, taking into account particular conditions that may dynamically change the VGE during the simulation. PLAMAGS also natively supports agent groups' (behaviors and interactions) and the simulation of any kind of particle systems (such as tear gas).
Of all the elements that are composing a simulation, the main actors of a PLAMAGS simulation are the objects and simulation agents, which are in constant interactions with each other and with the VGE. The various PLAMAGS components are formally defined in the language. Here are the definitions of the main categories of components.

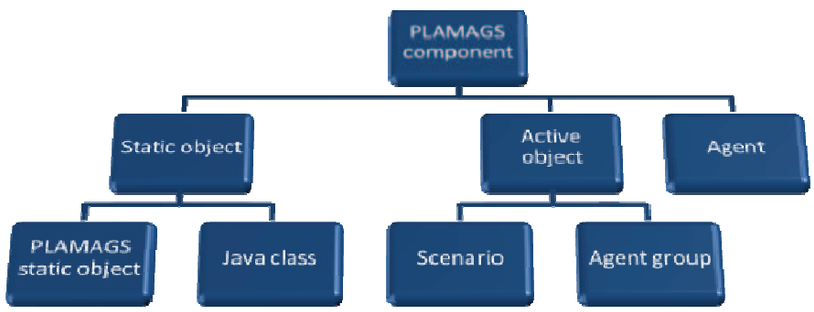


Fig. 10. Main categories of PLAMAGS components.

Components categories
PLAMAGS defines the three main categories of components: ***static object***, ***active agent*** and ***agent*** which are distinguished by their different behavioural capabilities. Moreover, two sub-categories of the active objects are defined, whether the ***agent groups*** and the ***scenarios***. Also, two sub-categories of the static objects are defined: PLAMAGS static object and java class. The following sub-sections characterize each of the categories presented in figure 10.

4.1 Objects and agents
Whatever its nature (static object, active object, agent), the definition of a component type is minimally composed of a set of visual, spatial and physical characteristics, as described in section 3.1.1 and of a set of internal properties that can be either constant or dynamic. In addition, the designer can associate a set of actions to a component type.
The main difference between static objects, active objects and agents corresponds to the definition of the associated behaviors (Garneau et al. 2010). Static objects do not possess any behavior and therefore, are totally passive (they are used to represent objects such as trees, fences and street lamps). Active objects can be considered as reactive agents, their behaviors being defined using lists of powerful rules (Levesque et al. 2008). Moreover, agents possess the most expressive of PLAMAGS' behavioural structures as discussed in the next section.

4.2 Agent behaviors
Behavioral graphs 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.

Fig. 11. Composition of an objective

A PLAMAGS behavioral graph 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. Each agent owns a set of objectives corresponding to its needs (Moulin et al. 2003). An objective is associated with rules containing constraints

characterizing the activation, execution or completion of the objective: we call them *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. An objective is also related to resources that it either needs to acquire or already owns, these resources being required for the objective's execution. Figure 11 presents the different elements constituting an objective in a schematic way.
The selection of the current agent's objectives relies on the graph structure, on previously executed objectives and is influenced by required resources, the objectives' priorities, as well as by the associated activation/execution/completion rules (Garneau et al. 2010). The execution of an objective is always conditional to its execution rule being triggered and to the availability of the required resources. An objective's priority is primarily a discriminating function or expression which is used to choose the active objective among a set of potential 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 at a given iteration is subject to the objectives' priorities.

Control of each objective execution in many phases
Compared to the majority of graph-based models where the execution of a node (corresponding whether to a state, an objective or a goal.) is monolithic, the execution of a PLAMAGS objective (be it simple, composed or aggregated) is a flexible process associated with several specification steps that can be easily controlled (figure 12).

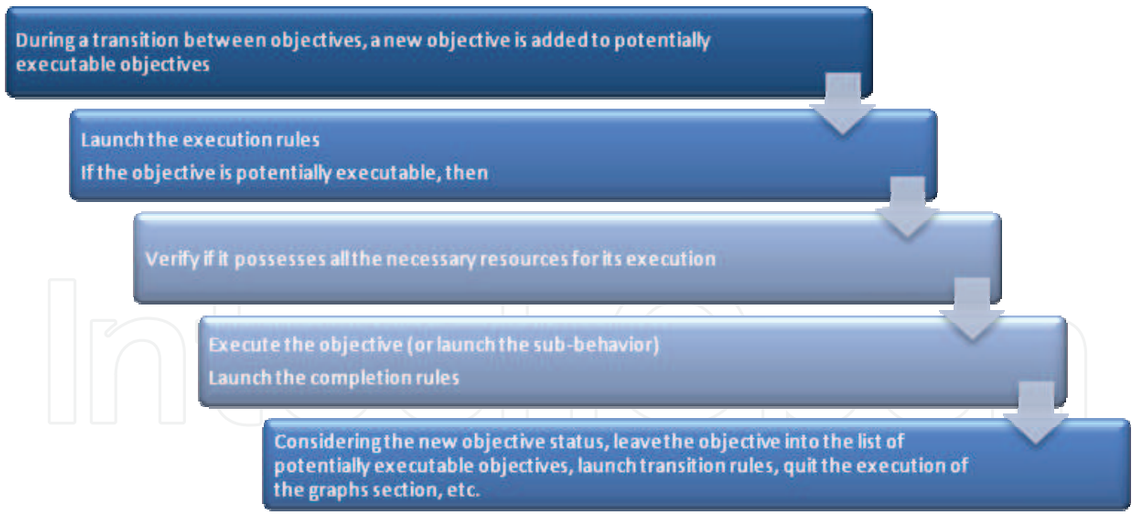


Fig. 12. Summary of an objective's execution dynamics.

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). *Aggregate objectives* are also decomposable structures: they are composed of a set of objectives which may not be related.

These objectives allow the designer to represent agents' objectives in which neither a hierarchical structure nor a predefined sequence of objectives is needed. Compound and aggregate objectives are well suited to regroup an agent's objectives as set of goals. 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 elementary actions which correspond to simple objectives: this corresponds to the "execution level" of the behavior.
Since agents often need to simultaneously achieve more than one objective, we provide an execution mode allowing to concurrently activate several objectives. The "mode" of declaration is specified for each objective because concurrent activation is not desirable everywhere in a behavior graph. This allows the designer to locally control the activation of parts of a behavior graph.
This graph structure and the PLAMAGS language associated offer several advantages.

- Intuitive modeling approach
- Executable specifications
- Abstraction and iterative refinement (based on the specification of sub-behaviors)
- Elimination of the translation step between a model definition and its implementation
- Concurrent execution of multi-layered objectives
- Automatic management of objectives' concurrency based on resources and priorities

4.2 Scenarios

Defining all the elements of a simulation is an error-prone and complex process. Since one of our aims in creating PLAMAGS is to guide the user when creating a simulation, we introduce a specific structure called "scenario" to specify in a structured way the various elements composing a simulation. Figure 13 presents the steps of the specification of a scenario.

Definition of VGE and execution properties
The scenario allows for the specification of the parameters of the VGE. Figure 13 presents the skeleton of a scenario that we discuss in the paragraph. In order to offer guidelines to the user, each of the VGE properties is defined by a specific key-word and associated value. Several other parameters need to be specified to configure the execution, the performance, the personalization, the interaction with external tools, the debugging as well as the optimization of the simulation. All these parameters are directly available in the scenario (within a specific block) and each of them possesses default values, minimally necessary to run the simulation. In this way, the user can concentrate on defining the properties he wishes to specify and can accept the default values for the properties that he is not interested in (see for example lines 7 and 8 in Figure 14).

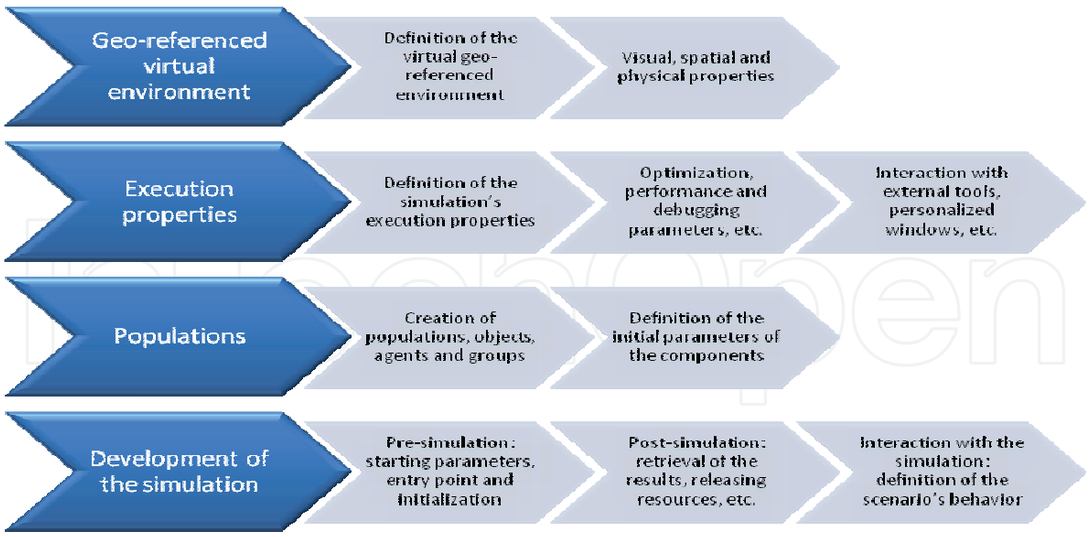


Fig. 13. Defined elements in a PLAMAGS scenario.

```
5   public scenario Scenario1
6
7       map coordinates : "0,500,0,900,0,100"
8       map mapModel ...
9       ...
10      component :
11          ...
12
13      rules Scenario1RB trigger every [50]
14
15      initialize call main()
16      initialize call println("Starting simulation...")
17
18      exit when [shouldStop()]
19
20      terminate call println("Ending simulation...")
21
22      logger debug = {
23              behaviour.algorithm,
24              behaviour.warning
25      }
26
27      method :
28
29          private void main(array<string> args)
30              for [i : int; i < args.size(); i = i + 1]
31                  ...
32              end for
33          end method
34
35          private boolean shouldStop()
36              ...
```

Fig. 14. Basic structure of a scenario.

Creation of populations and initial conditions

A block within the scenario (lines 10 and 11 in Figure 14) allow to instantiate (in a similar way as using calls of constructors in an object-oriented language) the objects and agents

populations that will participate in a simulation (static object, active object, gas, group, agent).

Preparation of the simulation (Pre-simulation)
The scenario offers the possibility to define the actions that will be executed only once before the execution of the simulation's main loop (main event loop), lines 15, 16 and 29 to 33 of figure 14, which is often necessary to carry out different initializations of a simulation.

Main execution loop
Once the pre-simulation is completed, the execution of the main loop is automatic and launches in an iterative way the behaviors of each active component of the simulation.

Interactions with the simulation
Since the scenario is itself a component, it can interact with the simulation. Its behavior is defined by a list of rules (line 13 of figure 14).

Ending of simulation
A scenario also allows for the definition of the actions that will be executed only when the simulation is over (line 20 in Figure 14). In this way, the designer can explicitly define and schedule the actions to be executed once the execution of the main loop simulation is over.

4.4 Gas

The PLAMAGS model allows to integrate components representing gases in a simulation using a type of active object directly supplied in PLAMAGS. Their behaviours are managed with the help of the PhysX Library (PhysX 2010) in the form of particle systems. The specification of a gas is quite flexible and parameterizable.

```
local smoke : Gas = constructor()

call smoke.setPosition(244,59,0)
call smoke.setParticuleImage("res/textures/smoke.jpg")
call smoke.setMaximumParticulesNumber(500)
call smoke.setParticulesSpeed(0,0,0.025)
call smoke.setEmissionAngleMargin(130.0)
call smoke.setEmissionSpeedMargin(0.0225)
call smoke.setEmissionPositionMargin(0.0025)
call smoke.setInitialParticulesRadius(0.03)
call smoke.setParticulesDissipationSpeed(0.006125)
call smoke.setMaximumParticulesRadius(0.35)
call smoke.setParticulesAcceleration(0,0,0.000635)
call smoke.setTransparency(50)
call smoke.setIsRelativeToMap(true)

call smoke.emit()
```

Fig. 15. Creation and parametration of a gas.

Figure 15 presents the code allowing for the creation of a gas representing a smoke cloud emanating from a small fire. The result is presented in Figure 16 as a series of fires.
The code presented in Figure 15 is all that is needed to create and parameterize a gas in PLAMAGS. Let us emphasize that it is usually not necessary to initialize all the parameters

of a gas, taking advantage of the default values offered by the PLAMAGS language. Different parameter sets can be used to create different kinds of gas: smoke clouds, toxic gases, stormy cloud cells, etc.

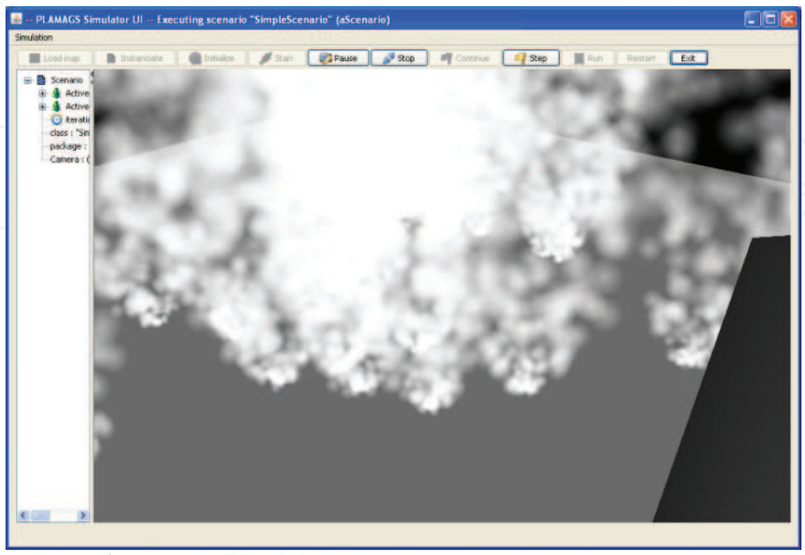


Fig. 16. Propagation of gases in PLAMAGS.

4.4 Groups

Agent groups possess a multitude of unique characteristics whose description is out of scope of this chapter. However, let us emphasize that the use of a group in PLAMAGS offers a set of functionalities, among which an important one is the possibility for a component's capability to perceive groups. The agents can also be perceived as belonging to groups (in addition to being perceived as individual agents). The groups inherit from the active agents' properties, and can be associated with a "group" behavior and all the associated rules. This does not prevent the agents belonging to a group to carry out their own behaviors.

4.5 Java classes

The extensibility of the PLAMAGS language allows for the use of any Java class in a simple way. Once a PLAMAGS type is defined from a Java class, this type is automatically defined as being a **static object** and is directly usable in a PLAMAGS specification. As a simple example, Figure 17 shows the use of the java.ioPrintWriter class in PLAMAGS.

```
private declarator JavaIO
      create type plamags.File : java.io.PrintWriter
end declarator

local file : plamags.File = constructor("data.txt")
call file.println("some text")
....
call file.close()
```

Fig. 17. Use of Java classes in PLAMAGS.

5. Language and IDE

This section is a quick overview of the general characteristics of the language (supported by the IDE), which is a complete and expressive agent-oriented language providing standard procedural and object-oriented (OO) features, as well as a bidirectional communication between PLAMAGS and Java (more details in (Garneau et al. 2010)). Although the language offers the majority of constructions available in procedural and OO languages, its power comes from its support of several structures dedicated to the specification of evolved agent behaviors. Furthermore, PLAMAGS allows a simple and transparent communication between the VGE and the agents, providing a set of tools that can be called directly from the language. This allows the user to directly integrate geographic and spatial information in the agent's decisional process.

5.1 Specification, definition, implementation and execution language

PLAMAGS is the first complete programming language for MAGS, totally dedicated to the implementation, but above all, usable in any phase of the MAGS development process (see Section 2).

Modeling, specification, definition and design

Starting with the first phases of the development process of a MAGS, PLAMAGS can be used to support the specification of the models thanks to its syntax which is both declarative and procedural. To this end, the language offers declarative syntactic blocks as well as a set of well chosen key-words.

Implementation

All the steps of the development process are supported by the language. Let us emphasize that the language, offers all the required structures to define and implement the dynamics of behavioral graphs and rules lists.

Execution and validation

The language is associated with: 1) an interpreter/compiler of the PLAMAGS syntax which translates the specification into an interpretable/compilable code and, 2) an execution engine which is responsible for managing the execution of the objects' and agents' behaviors. The execution engine is also in charge of ensuring the coherence of the executed behaviors (concurrent execution of the objectives, priorities, resources, objectives states, etc.) by doing a systematic validation of the executed objectives, of their status, etc.

Debugging

The language also supplies a set of flexible and configurable tools for debugging, notably a tracking system enabling the user to follow step by step the different decisions made by the behavioral execution engine, to get the execution result of each objective, etc.

Geo-spatial management and spatialized interactions

For an easy development of MAGSs, the language also integrates all the primitives necessary to specify the geo-spatial aspects of the simulation as well as mechanisms (syntax elements) to configure the interactions between agents, as well with the VGE.

Extensibility and versatility
Finally, to ensure that it is never limited by the syntax of the functionalities that it offers, the language allows the direct use of any external Java class or library.

5.2 IDE to support the method and the language use

The PLAMAGS language is supported by an IDE which is a complete development environment that allows for the quick development and execution of multi-agent geo-simulations. Figure 18 presents an overview of the IDE support of the PLAMAGS method.
The IDE 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 to run simulations in an interpreted mode); 6) a Java code generator and a compiler (to run simulations in a compiled mode); 7) a 3D engine to visualize the simulations; 8) a visual programming tool (to graphically develop behaviors); 9) an interface allowing to quickly create each kind of components in the shape of "stub" files; 10) an assistant allowing to create, modify, edit a component in a visual way (during development); 11) a visualization tool of the used Java classes; 12) an integrated documentation.
Moreover, to handle physics constraints (collisions, gas dispersion, repelling, friction, ground elevations, etc.) and to allow agents and objects to take into account these constraints in their decision making process, PLAMAGS uses one of the most powerful physics engine, PhysX.
All these tools are integrated in a completely transparent way into the IDE where they are correctly configured by default (compiler, interpreter, etc.). The user only has to create a project with the help of an automated assistant and to start the creation of the simulation. Then, he can, without any additional effort (no configuration is necessary), compile and execute and visualize any PLAMAGS program. Figure 19 presents a snapshot of the PLAMAGS IDE's interface.

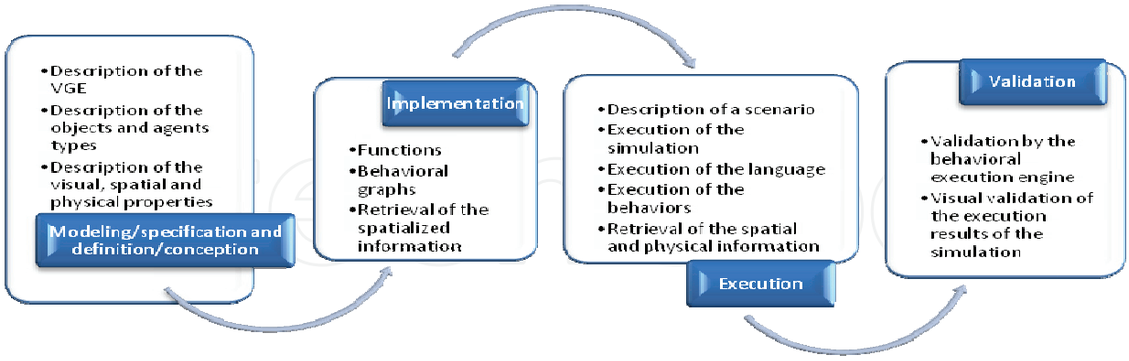


Fig. 18. PLAMAGS in all the steps of the development process of a MAGS.

6. Results and Conclusion

The PLAMAGS method distinguishes itself from the other MABS methods/tools by its framework that supports every step of the development cycle of a MAGS, from the modeling step to the validation step. This is made possible thanks to the language that

supports the modeling, the implementation and the execution of MAGS, while offering all the necessary mechanisms for the integration of the geo-spatial interactions between agents and with the VGE. In this way, PLAMAGS eliminates the transition and translation steps between the models and their implementations that are required by other MAGS specification approaches, which greatly reduces the development effort.

In order to show the large scope of PLAMAGS, we have meticulously replicated the experimentation presented in the « *Agent-based Simulation Platforms: Review and Development Recommendations* » article (Railsback et al. 2006) in the form of a 3D simulation. The results we obtained can be advantageously compared with those of the multi-agent based simulation tools used in Railsback's study: (MASON (Luke et al. 2004), SWARM (SWARM 2010), Java Swarm, Repast (North et al. 2007) and NetLogo (NetLogo 2010)). The results will be presented in a forthcoming paper, however we can already conclude that PLAMAGS could either be used for development of MABS (agents based simulations) or for the MAGS.

We used PLAMAGS in a variety of MAGS simulation projects and we have identified three main shortcomings for which we intend to find solutions in our 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 user's task. Second, the JPCT library (JPCT 2010) used for 3D rendering seems to reach its performance limits when dealing 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 in our simulation examples. We may try to find a replacement library. Third, our 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 several elements of its structure, and then resume the simulation in the state saved before the changes.

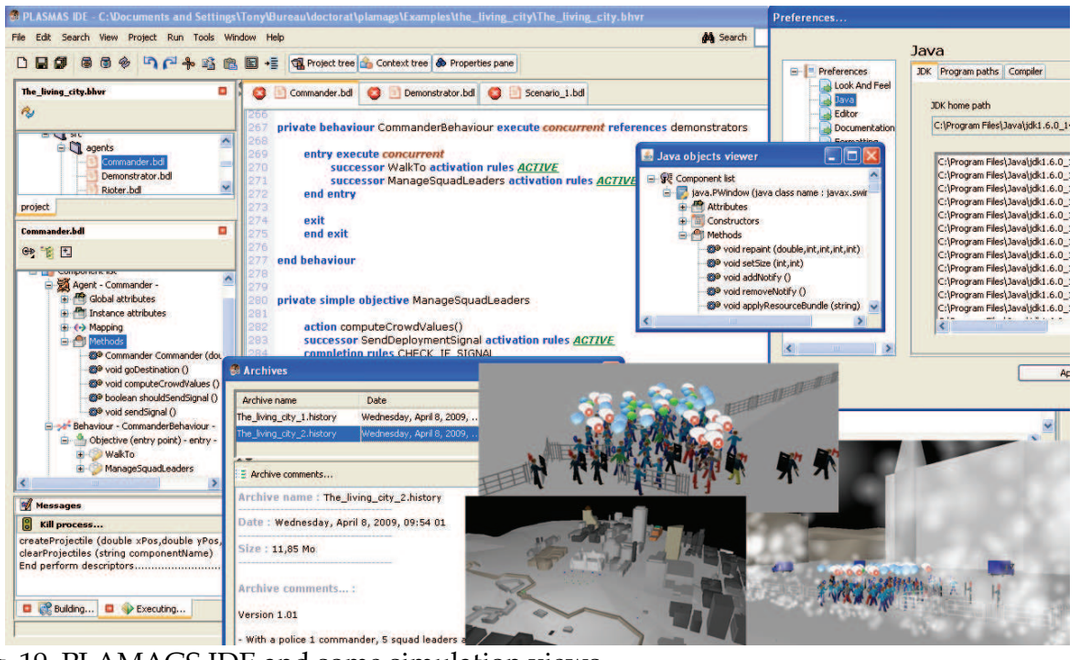


Fig. 19. PLAMAGS IDE and some simulation views.

Acknowledgments: 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. (2009). from http://www.presagis.com/.

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

Benenson, I. and P. M. Torrens (2004). "Geosimulation: object-based modeling of urban phenomena." Computers, Environment and Urban Systems **28**(1): 1-8.

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.

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.

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.

Garneau, T., B. Moulin and S. Delisle (2010). Effective agent-based geosimulation development using PLAMAGS. Modelling, Simulation and Optimization. Intech**:** 684-708.

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. (2010). 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**:** 739-744.

Luke, S., C. Cioffi-Revilla, L. Panait and K. Sullivan (2004). MASON: A New Multi-Agent Simulation Toolkit. Eighth Annual Swarm Users/Researchers Conference, SwarmFest 2004, University of Michigan, Ann Arbor, Michigan USA.

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.

NetLogo. (2010). "NetLogo." from http://ccl.northwestern.edu/netlogo/.

North, M. J., E. Tatara, N. T. Collier and J. Ozik (2007). Visual Agent-based Model Development with Repast Simphony. Agent 2007 Conference on Complex Interaction and Social Emergence, Argonne National Laboratory, Argonne, IL USA.

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/.

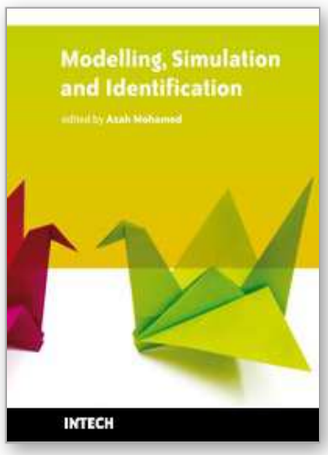
PhysX. (2010). 2008, from http://www.nvidia.com/object/physx_9.09.0408_whql.html.

Railsback, S. F., S. L. Lytinen and S. K. Jackson (2006). "Agent-based Simulation Platforms: Review and Development Recommendations." SIMULATION **62**(9): 609-623.

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

SWARM. (2010). "SWARM." from http://www.swarm.org/.

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



Modelling, Simulation and Identification

Edited by Azah Mohamed

ISBN 978-953-307-136-7
Hard cover, 354 pages
Publisher Sciyo
Published online 18, August, 2010
Published in print edition August, 2010

Modeling, simulation and identification has been actively researched in solving practical engineering problems. This book presents the wide applications of modeling, simulation and identification in the fields of electrical engineering, mechanical engineering, civil engineering, computer science and information technology. The book consists of 17 chapters arranged in an order reflecting multidimensionality of applications related to power system, wireless communication, image and video processing, control systems, robotics, soil mechanics, road engineering, mechanical structures and workforce capacity planning. New techniques in signal processing, adaptive control, non-linear system identification, multi-agent simulation, eigenvalue analysis, risk assessment, modeling of dynamic systems, finite difference time domain modeling and visual feedback are also presented. We hope that readers will find the book useful and inspiring by examining the recent developments in the applications of modeling, simulation and identification.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

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