PLAMAGS: A LANGUAGE AND ENVIRONMENT TO SPECIFY INTELLIGENT AGENTS IN VIRTUAL GEO-REFERENCED WORLDS

Tony Garneau, Bernard Moulin
Département d’informatique et de génie logiciel, Université Laval, Québec, Canada, G1K 7P4
tony.garneau.1@ulaval.ca, bernard.moulin@ift.ulaval.ca

Sylvain Delisle
Département de mathématiques et d’informatique, Université du Québec à Trois-Rivières, Québec, Canada, G9A 5H7
sylvain.delisle@uqtr.ca

ABSTRACT
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. Several platforms were developed for the specification and the implementation of this type of simulations, but they do not yet offer a complete language for the specification and validation of agents’ behaviors which have apprehension capacities of virtual space (perception, reasoning on their objectives, etc). In this article we present the PLAMAGS project in which we propose 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.

KEY WORDS
multiagent geo-simulations, agent behavior, Agent-oriented language, virtual geo-referenced environment.

1. Introduction
During the past decade, the technology based on Intelligent Virtual Agents (IVAs) 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 IVA 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 has been used for a long time by all governmental and private organizations whose activities deal with the geographic space in a way or another. But most of these applications are complex since they deal with spatial dynamic phenomena and usually involve large populations of individuals (persons, animals, insects, plants, etc.) and their interactions [1].

There are numerous situations that decision makers from various sectors (governmental, military, industrial, medical, social) need to monitor in order to insure human security (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 [2] 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 [3], especially in the domain of urban dynamics and landcover planning.

1.1 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, which resulted in the creation of the new field of Multi-Agent- Geo-Simulation [4, 5]. 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 the virtual geo-referenced space [6]. 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 [7, 8]. 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.

However, there is a large variety of phenomena in which individuals 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 intelligent virtual agents’ models, in order to carry out autonomous behaviors within geographic virtual environments. Such a model was proposed by Moulin and colleagues [5] 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 behavior (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 [9], AI.Implant [10] and PathEngine [11] deal with the spatial aspects of agent behaviours 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 [12] and SPIR.OPS [13] offer sophisticated specification means for objects/agents behaviors based on models inspired by finite state machines [14]. But, the use of finite state machines leads to complex graphs to represent relatively simple reactive behaviours. Behaviors developed using these tools lead to reactive agents or “navigation driven” agents [15]. 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 mechanism, agents cannot apprehend the virtual environment (act in the environment and interact with the object contained in it).

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

- Autonomous and individual perception mechanism
- Decision-making in relation with a geo-referenced virtual environment
- Proactive and autonomous behaviours 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 SIVAs. 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 multiagent geo-simulations. The PLAMAGS toolkit was motivated by the need to provide a complete and real programming language completely dedicated to the specification and the execution of multiagent geo-simulations.

Section 2 introduces spatialized IVAs, the main concepts on which the PLAMAGS language is based. Section 3 presents the main characteristics of the language and Section 4 concludes the paper with a discussion and some future work.

2. Spatialized IVAs

Independently of their specific characteristics (be they passive objects, reactive agents or proactive/cognitive agents), SIVAs appropriate for geo-simulations have common important characteristics: they are situated in a virtual geo-referenced space (they actually move in it), they must comprehend 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 example, in the simulation of a manifestation (protest march, demonstration), rioter agents have to distinguish police agents from other rioter agents and demonstrator agents participating to the manifestation. On their side, police agents need to perceive the demonstrator agents’ movements and displacements of fences, in order to adopt an appropriate strategy. As another example, pedestrians participating in a peace walk must be able to perceive and follow the walking group along a planned route, adjust their own pace to the crowd’s pace, avoid cars and other obstacles, etc. We distinguish three categories of SIVAs, depending on the sophistication of their behaviours, reasoning and interactions with the virtual space: passive SIVAs, reactive SIVAs, and cognitive SIVAs.

2.1 Passive SIVA

In a simulation, numerous components are still objects having a physical presence 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 represent passive SIVAs thanks to a specific language structure that we call a static object type which is equivalent to an object-oriented class in an object-oriented language plus a visual representation and a physical definition (if necessary). As for OO classes, a static object type has properties (which can be constant or modifiable) and methods. Methods provide us with a way to attach processes to static objects that may change their location, their visual representation and change their properties as the result of actions applied on the objects. In addition to standard OO classes’ capacities, static objects can own a visual representation and a physical definition and description (bounding volume, mass, etc.). In the manifestation simulation, fences are defined as static objects since they only have some properties and a physical presence, but no behavior.

2.2 Reactive SIVA

Reactive SIVAs have relatively primitive behaviors that can be efficiently represented using a reactive agent approach. We represent reactive SIVAs thanks to a specific language structure that we call active object. Active objects are equivalent to reactive agents in classical agent models. They have capabilities similar to those of static objects’ properties, methods and visual/physical definition), plus a list of rules used to specify their behaviors. Reactive SIVAs have no decision-making capabilities: they only react to their inputs (most of the time obtained from their perception mechanism) thanks to a list of rules which is automatically triggered by the simulation engine. Using these rules, the reactive object has the possibility to interact with its environment and other SIVAs during the simulation.

2.3 “Cognitive” SIVA

We consider cognitive SIVAs 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 behaviours. We thus defined cognitive SIVAs thanks to a specific language structure that we call agent.

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 behaviour which is represented by a multi-layered directed graph composed of a set of objectives that the agent tries to reach (see Figure 1).

The objectives are organized in hierarchies such that elementary objectives (called simple objectives) are associated with actions that the agent can perform (i.e. objectives “PresenceAct” and “CheckAround” in Figure 1). Each agent owns a set of objectives corresponding to its needs, usually represented as dynamic states [5]. An objective is associated with rules containing constraints on the activation and completion of the objective, called activation rules and completion rules [5]. Constraints are dependent on time, on the agent’s states, and on the environment’s states. The selection of the current agent’s objectives relies on the graph structure, on previously executed objectives and on priorities and activation/completion rules related to the agent’s objectives. 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 on the objective.

The structure of the multi-layered directed graph allows us to define a behavior at different levels of abstraction and to divide behaviours into sub-behaviours. An abstraction level can be added by inserting a compound or an aggregate objective in the behaviour. A compound objective can be thought of as a decomposable structure representing a sub-behaviour (its structure is similar to a behaviour structure), see Figure 2. 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 behaviour.

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 behaviour graph. This allows to locally control the activation of parts of the behaviour graph. For example, in Figure 1 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).
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. 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 behaviours. Among these structures, the main ones are: static objects (passive SIVA), active objects (active SIVA), agents (cognitive SIVA), rule lists (used for active objects, activation and completion rules), behaviors and objectives (simple, compound and aggregate).

It is not possible to present all the language elements in this paper. We will mainly focus on the structures implementing the SIVAs’ characteristics presented in the previous section. But let us start by quickly introducing some elements of the language facilitating the management of space dynamics.

3.1 Automatic management of spatial elements

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 in the PLAMAGS language via the use of “mapped items” and “managed actions”.

Mapped items are simple language declarations specifying space-related characteristics of agents and objects. Once these characteristics are specified, they are automatically managed by the simulation engine. Figure 3 shows how to specify some mapped items.

```
map boundingVolume : "automatic"
map fieldOfView : "28, 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. 3. Two “mapped items” and a method recovering perceived elements.

In the manifestation simulation example, demonstrator agents have their own perception mechanism. To assign perception capabilities to a component (reactive or cognitive SIVA), all we have to do is 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 (everywhere in the code). Figure 3 shows the code to recover perceived elements.

The language offers more than 100 different features corresponding to most of the basic capabilities needed to easily and automatically handle the agents’ physical interaction with the spatial environment. Using these features, the user can pay attention to the specification of the cognitive aspects of the SIVA’s behaviors without being concerned with managing their coherence at the spatial level.

In the PLAMAGS language, Managed actions are used to specify a lot of predefined spatialized actions. They are
provided to automatically handle space coherence in 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.

3.2 The main structure of the PLAMAGS language

3.2.1 Static and active objects)

Since passive SIVAs are inanimate components of the simulation having various properties or structures representing properties of other components, OO classes are perfectly suitable for the definition of those SIVA. PLAMAGS’ static object structure is used to represent the equivalent of an object-oriented class plus a visual and a physical representation when needed. Their implementation is similar to active objects except that they don’t have rules specification.

Reactive behaviours allow a component to respond to stimuli (applying functions to these stimuli, to modify internal states, to carry out actions, etc). Our rule list model is directly inspired by traditional reactive approaches in which a rule is made up of a set of conditions which must satisfy certain constraints; and when these constraints are satisfied, a series of actions is triggered. In PLAMAGS’ active objects (reactive SIVAs), the rule’s conditions are relational operators applied to both object properties and function calls. Most of the time, these function-calls return computed data that is obtained from the perception. Thereafter, if the rule conditions are satisfied, actions (method calls and spatialized actions) and properties modification are triggered.

In PLAMAGS, rule lists are used in different situations. As we saw above, they are used as behaviours for active objects, but they are also used to verify conditions when activating and completing objectives. However, independently of their use, they always have the same structure and the same execution logic. A PLAMAGS rule list contains one or more rules. Each rule is composed of a list of preconditions known as LHS (Left Hand Side) and a list of successors 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.

3.2.2 Cognitive SIVAs (i.e. agents)

As mentioned in Section 2.3, a cognitive SIVA 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 subsection introduces the main structures of the language implementing behaviours in PLAMAGS. The code presented in this sub-section represents a part of the behaviour and the objectives of Figure 1 and Figure 2. First, let us look at the demonstrator agent (top of figure 4). 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.

```plaintext
public agent Demonstrator
    behaviour DemonstratorBehaviour
        // optional: trigger every [...] or when [...] 
    method: ...
end agent

public active object Squad
    attribute: ...
    rules SquadRules trigger when [hasDestination0]
        // or trigger every [...] 
    method: ...
end object

private behaviour DemonstratorBehaviour
    entry execute concurrent 
    successor CheckAround activation rules ACTIVE 
    successor PresenceAct activation rules ACTIVE 
end entry 
exit ... end exit
end behaviour
```

Fig. 4. The “behaviour DemonstratorBehaviour” declaration (line 3) specifies that the demonstrator type will use a behaviour called “DemonstratorBehaviour” which is defined at the bottom of the figure.

The behavior
The “behavior” is the global structure representing the whole behavior graph. It allows identifying the “entry point” (see Figure 1 and bottom of Figure 4) 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 objectives 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), and a completion rule list (mandatory). This section quickly describes these elements.
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 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.

A successor links an objective to a potential objective (its successor) that may be activated if the execution of the behavior associated with 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. Bottom of Figure 4 declare two successors named “CheckAround” and “PresenceAct” (these objectives must be defined elsewhere).

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 4), 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 successors is 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”. The next figure 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”.

**Activation rules**

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

Fig. 5. Rule list called “RUN_AWAY.GO” used in the “PresenceAct” and “PeacefullyDemonstrate” objectives.

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 done: re-execute the objective, choose a successor, stop the branch’s execution (this graph’s section), etc. Note that the completion rules are only used to determine the action to perform on the currently executed objective. If the action is to choose 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.

**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 behaviour (Figure 1), “PresenceAct” and “CheckAround” are two simple objectives. The next figure shows the code of the “CheckAround” objective.

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

Fig. 6. The “CheckAround” simple objective implementation.

A compound objective is a decomposable structure representing a sub-behavior. Figure 2 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.

**4. Discussion and future work**

This paper emphasized the PLAMAGS language, but PLAMAGS is also a complete development environment allowing to quickly develop and execute multiagent geosimulations. The 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) 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). 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 on 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 leads to specifying 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 are not sufficient to develop advanced multiagent geosimulations in which agents possess cognitive capacities. To address this issue, we introduced behaviors using multi-layered directed graphs. Contrary to other models inspired by finite state machine, PLAMAGS behavior’s graphs manage the concurrent execution and multiple concurrent states (using “on the fly” context addition and withdrawal), infinite decomposition (sub-behavior layers), extremely expressive and powerful transitions between nodes proceeding into two phases: activation and completion (using rule lists). PLAMAGS offers the user with a simple and transparent implementation allowing to directly transfer the visual models to PLAMAGS code, since the language has a built-in structure representing all the visual model elements (objectives, rules, etc.).

Recently, we added some important new features. We added a Java code generator and a compiler allowing to execute simulations in a fully compiled way. It considerably increases the execution performance of the simulations. Moreover, to handle physics constraints (collisions, gas dispersion, repelling, friction, ground elevations, etc.) and allow agents and objects to take into account these constraints in their decision making process, PLAMAGS uses one of the most powerful physics engine named physX developed by Ageia (http://www.ageia.com/). This enabled us to integrate in PLAMAGS simple clauses to quickly and easily specify any kind of gas. We are currently developing the visual specification interface for the 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 able to automatically generate the corresponding PLAMAGS code. The user will also be able to do the opposite operation and pass from the textual programming mode to the visual one. We are also investigating the possibility to integrate some behavior validation mechanisms using graph theory.

5. References