Construction of Intelligent Virtual Worlds using a Grammatical Framework

Gabriel López-García, Antonio Javier Gallego-Sánchez,

Rafael Molina-Carmona and Patricia Compañ-Rosique

Group "Informática Industrial e Inteligencia Artificial".

University of Alicante (Spain)

March 31, 2014

Abstract

The potential of integrating multi-agent systems and virtual environments has not been exploited to its whole extent. This paper proposes a model based on grammars, called Minerva, to construct complex virtual environments that integrate the features of agents. A virtual world is described as a set of dynamic and static elements. The static part is represented by a sequence of primitives and transformations and the dynamic elements by a series of agents. Agent activation and communication is achieved using events, created by the so called event generators.

The grammar defines a descriptive language with a simple syntax and

a semantics, defined by functions. The semantics functions allow the scene to be displayed in a graphics device, and the description of the activities of the agents, including Artificial Intelligence algorithms and reactions to physical phenomena.

To illustrate the use of Minerva, a practical example is presented: a simple robot simulator which considers the basic features of a typical robot. The result is a functional simple simulator.

Minerva is a reusable, integral and generic system, which can be easily scaled, adapted, and improved. The description of the virtual scene is independent from its representation and the elements which it interacts with.

1 Introduction

The growing influence of Multi-Agent Systems (MAS) in several fields of research (sociology, economics, artificial intelligence, ecology, and so on) has led to a significant evolution in its development. On the other hand, the spectacular progress of the Virtual Reality (VR) systems has contributed to present the information in a more immersive way using new forms of analysis [18]. Video games and the entertainment industry in general, have had a decisive influence on this striking progress [17, 14].

All these developments have opened up new challenges, among them the possibility of integrating MAS and Virtual Worlds (VW). Several advantages has been detected from the possibility of combining them [20]:

- Thanks to the regulation imposed by the MAS, the 3D environment becomes a normative virtual world where norms are enforced at runtime instead of by the terms of service contract.

- A 3D real-time representation of the multi-agent system facilitates a better understanding of what is happening at both agent and the entire system levels.

- Virtual world participants can be both humans and software agents facilitating human direct participation in MAS and intelligent agents participation in VW.

We also consider that the combination of MAS and VW would lead to a much more natural way of developing complex VR systems. VR systems can be considered to be composed of three main modules (rendering system, physics engine and artificial intelligence system), and the three modules are usually build using different tools that are frequently not designed to work together. At most, the integration of MAS and VR are usually confined to endow the system with Artificial Intelligence (AI), but the physics engine is frequently missed. As a conclusion, the potential of combining MAS a VR has not been exploited to its whole extent.

This paper describes an integral model based on grammars to construct complex environments that take advantage of the MAS and the VR systems features. This framework uses a descriptive language and discrete events to specify all the features necessary to define the agents and their environment. It

allows an easy interaction with the user so that the initial conditions could be changed, the experiments can be reproduced and the components of the scene are displayed in real time. It is also independent from the display and interaction devices and it allows the incorporation of physical phenomena. Finally, the agents can be easily reused.

The following section presents some related work. Section 3 provides a description of the proposed model. In section 4, a case of study is proposed with the aim of showing the use of the whole system. Finally, some conclusions and possible lines of future work are presented.

2 Related work

There are many generic and specific work environments to develop MAS, but they seldom allow the definition of complete VR systems, such as advanced visual features or physics phenomena. For instance, in Sociology [2, 6], very specific solutions are used, generally oriented to sociological studies, such as the movement of crowds [22, 16]. In some cases, they use graphics to display statistics, the density of agents within the environment or even simple animations of the movements of agents [16]. Netlog [24] is a useful tool to study highly complex systems. It has developed some graphical features, but the agents are just represented by Logo style triangles just to allow the user to observe the movement of agents and the population density. There are also some interesting works related to very specific aspects of graphical representation of characters in

a virtual environment. Such are the cases of the use of agents to get an expressive gaze animation [19] or to control the movement of an skeleton-based human character [3]. All these examples deal with only few aspects of the graphical representation.

The applications of the MAS to endow a VR system with AI are much easier to be found. One of these systems proposes an implementation of a virtual world where objects react to several gestures of the user [12]. The work of [23] uses the concepts of perception of the MAS applied to virtual reality, where the agents react interactively with the user.

In the entertainment industry, some MAS applied to games can also be found. In this case, the agents are usually called bots [9] and they are programmed as characters of the game, with objectives, strategies and actions. Programming is done with a specific script language for each game, so that characters cannot easily be transferred between games. These systems are so successful that they are being used in research for some specific cases [17].

An increasing variety of generic development environments with a similar philosophy are also emerging: they implement the most important features of agents, i.e., perception of the world, communication, reaction, and so on [6]. Some examples are Repast [13] and MASON [10].

Agents are usually considered to have some generic characteristics that make them difficult to model [6]. Some common strategies for implementing them and some languages to define their behavior can be found. For instance, Jason [8], Jack [1] or 2APL [4] provide programming languages based on the BDI

(Beliefs/Desires/Intentions) architecture. Nevertheless they do not provide capabilities to deal with the graphical representation of the system. This lack of unified system to define the graphical and the behavioral aspects of the system makes some problems arise, such as the difficult reproduction of the results provided by the MAS [2], the lack of a suitable visual representation or the limited capabilities of interaction with the user and with the environment. Some new contributions try to describe multi-agent meta-models to standardize the design, simplify the implementation and integrate the agent frameworks into a single tool [7]. MAS could take advantage of the wide research carried out in this fields to develop VR systems. In this context, some agent meta-models for virtual reality applications [15] have also been described. They cover some aspects of virtual reality, but always considering the AI modules and disregarding most aspects about the graphical representation and the physics.

There are also some few attempts to combine virtual worlds with multi-agent systems using a grammatical approach. In [20, 21] a system that can automatically generate a 3D virtual world from a MAS specification is proposed. They present the concept of Virtual World Grammar (VWG) that supports the generation of 3D virtual worlds, through the extension of the shape grammar concept, adding semantic information of the environment and the MAS. This way, the geometrical data included in the shape grammar are extended with semantic information about both MAS specification, and shape grammar elements. They also include heuristics and validations that guide the VW generation and present a framework named Virtual World Builder Toolkit (VWBT) for the definition

and execution of VWGs.

Our proposal has some points in common with [20, 21]. We propose a grammatical framework to construct VWs and we also extend its features to add semantic information. However, our approach has an important contribution: we endow the system with the capacity of evolving, so that the dynamic processes can be easily incorporated and the AI system and the physics engine can be naturally and elegantly embedded in the definition of our VW.

3 Model description

A virtual world can be described as a set of dynamic and static elements. The static part is made of a sequence of primitives and transformations defined in a representation system G, usually a geometric system. However, a primitive is not just a draw primitive (e.g. a sphere) but also any action on the representation system that may be a visual action or not (e.g. a sound). A transformation is a modification on the primitive behavior. They will affect the set of primitives inside their scope of application.

The agents are the dynamic elements and they are made of activities and an optional set of attributes that make up their internal state. Each activity is a process that is executed as a reaction to a given event. The agents can have a geometrical representation, defined using primitives and transformations, depending on their internal state.

An event causes the execution of an activity. Its generation is independent

Table 1: Production rules for grammar M

1.	**WORLD** $\rightarrow$ OBJECTS
2.	**OBJECTS** $\rightarrow$ OBJECT | OBJECT $\cdot$ OBJECTS
3.	**OBJECT** $\rightarrow$ FIGURE | TRANSFORMATION | AGENT
4.	**AGENT** $\rightarrow a^d_{st}$(OBJECTS), $a^d_{st} \in A^D_{ST}, d \subseteq D$, $st \subseteq ST$
5.	**TRANSFORMATION** $\rightarrow t$(OBJECTS), $t \in T$
6.	**FIGURE** $\rightarrow p^+$, $p \in P$

from the physical device. They provide the different ways of communication in the MAS.

Those are the main elements of the proposed system, Minerva. Minerva stands for *Modelo Integral para Entornos de Realidad Virtual y Agentes* (Integral Model for Virtual Reality Environments and Agents, in spanish). Minerva is a grammatical model, where the whole scenes are represented with a string generated by a given grammar M.

A string $w \in \Sigma^*$ is generated by the grammar M, if it can be obtained starting with the initial symbol WORLD and using the given production rules (table 1), where P is the set primitives, T is the set of transformations, A^D_{ST} is the set of agents, that have some state from a set ST and respond to some events of a set D. The symbols () indicate the scope and $\cdot$ the concatenation of symbols. The language $L(M)$ is the set of all the strings which can be generated by this method, so: $L(M) = \{w \in \Sigma^* \mid WORLD \stackrel{*}{\rightarrow} w\}$. This grammar is a context-independent grammar (or a type-2 grammar, according to the Chomsky

hierarchy). Therefore, there is a procedure which verifies if a scene is correctly described.

3.1 Semantics of the language $L(M)$

Apart from the language syntax, it is necessary to define the functionality of the strings, that is, the semantics of the language. In our case, it is defined using a denotational method, which describes the meaning of the string through mathematical functions.

3.1.1 Semantic Functions for Primitives and Transformations (Rules 6 and 5)

Rule 6 defines the syntax of a figure as a sequence of primitives. Primitive's semantics is defined as a function $\alpha : P \rightarrow G$. Each symbol in the set P represents a primitive on a given geometric system G. So, depending on the definition of the function α and on the geometric system G, the result may be different. G represents the actions which are run on a specific geometric system. An example of geometric system are graphical libraries such as OpenGL or Direct3D, but the function α has no restrictions on the geometric system that can be applied to.

In Rule 5, the scope of a transformation is limited by the symbols "()". Two functions are used to describe the semantics of a transformation: $\beta : T \rightarrow G$ (it is carried out when the symbol "(" is processed), and $\delta : T \rightarrow G$ (it is carried out when the symbol ")" is found). These two functions have the same features

that the function α, but they are applied to the set of transformations T, using the same geometric system G.

Given a string $w \in L(M)$, a new function φ is defined to run a sequence of primitives P and transformations T in a geometric system G:

$$\varphi(w) = \left\{ \begin{array}{ll} \alpha(w) & if\ w \in P \\ \beta(t); \varphi(v); \delta(t) & if\ w = t(v) \wedge v \in L(M) \wedge\ t \in T \\ \varphi(s); \varphi(v) & if\ w = s \cdot v \wedge s, v \in L(M) \end{array} \right\} \quad (1)$$

One of the most important features of this system is the independence from a specific graphics system. The definition of the functions α, β and δ provides the differences in behavior, encapsulating the implementation details. Therefore, the strings developed to define virtual worlds may be reused in other systems.

3.1.2 Semantic Function for Agents (Rule 4)

The semantics of agents is a function which defines its behavior or, in terms of VR systems based on frames, its evolution in time. This is why it is called *evolution function* λ and is defined as: $\lambda : L(M) \times E^D \rightarrow L(M)$, where E^D is the set of events for the device D (considering these devices as any software or hardware process that sends events). By applying the function $\lambda(w, e^f)$, $w \in L(M)$ is transformed into another string u, which allows the system to evolve. It has a different expression depending on its evolution, but the general expression is defined as:

$$\lambda(a_{st}^{d}(v), e^{f}) = \left\{ \begin{array}{ll} u \in L(M) & if \quad f = d \\ a_{st}^{d}(v) & if \quad f \neq d \end{array} \right\} \tag{2}$$

The result u of the function may contain or not the own agent, it can generate other events for the next frame or change the state st (i.e. the set of attributes) of the agent. The function λ can define a recursive algorithm, called *function of the system evolution* η. Given a set of events $e^i, e^j, e^k, \ldots, e^n$ (denoted as e^v, where $v \in D^+$) and a string w, it describes the evolution of the system at a given point in time. This algorithm also uses the operator $\prod_{\forall f \in v}$ which concatenates strings.

$$\eta(w, e^v) = \left\{ \begin{array}{ll} w & if \ w \in P^+ \\ t(\eta(v, e^v)) & if \ w = t(v) \wedge v \in L(M) \wedge \ t \in T \\ \prod\limits_{\forall f \in v} (\lambda(a_{st}^{d}(\eta(y, e^v)), e^f)) & if \ w = a_{st}^{d}(y) \wedge y \in L(M) \\ \eta(s, e^v) \cdot \eta(t, e^v) & if \ w = s \cdot t \wedge s \in L(M) \wedge \ t \in T \end{array} \right\} \tag{3}$$

For the visualization of an agent, it must be first converted into strings made up only of primitives and transformations. This conversion is carried out by a special type of function λ called *visualization function* $\theta : L(M) \times E^V \rightarrow L(E)$, where $V \subseteq D$ are events used to create different views of the system, E^V are events created in the visualization process, and $L(E)$ is a subset of the language $L(M)$, including only primitives and transformations, but no agents. This function is defined as:

$$\theta(a_{st}^{d}(v), e^{f}) = \left\{ \begin{array}{ll} w \in L(E) & \textit{if} \quad f = d \wedge d \in V \\ \epsilon & \textit{if} \quad f \neq d \end{array} \right\} \tag{4}$$

As with the function λ, an algorithm is defined for θ. It returns a string $z \in L(E)$, given a string $w \in L(M)$ and a set of events e^{v}, where $v \in V^{+}$ and $V \subseteq D$. This function is called *function of system visualization* π and it is defined as: $\pi : L(M) \times E^{V} \rightarrow L(E)$

$$\pi(w, e^{v}) = \left\{ \begin{array}{ll} w & \textit{if } w \in P^{+} \\ t(\pi(y, e^{v})) & \textit{if } w = t(y) \wedge y \in L(M) \wedge \ t \in T \\ \prod\limits_{\forall f \in v} (\theta(a_{st}^{v}(\pi(y, e^{v})), e^{f})) & \textit{if } w = a_{st}^{v}(y) \wedge y \in L(M) \\ \pi(s, e^{v}) \cdot \pi(t, e^{v}) & \textit{if } w = s \cdot t \wedge s \in L(M) \wedge \ t \in T \end{array} \right\} \tag{5}$$

3.1.3 Semantic Functions for OBJECT, OBJECTS and WORLD (Rules 1, 2 and 3)

The semantic function of WORLD is a recursive function which breaks down the string of the WORLD and converts it into substrings of OBJECTS. Then, these substrings are in turn broken down into substrings of OBJECT. And for each substring of OBJECT, depending on the type of the object, the semantic function of agent, transformation or primitive is run.

3.2 Activity and Events

In MAS some mechanisms must be established to model its activities. These activities are run by agents and are activated by events. Not all activities are run when an event is received, they can also be run when certain conditions are satisfied. The following event definition is established: e_c^d *is defined as an event of type* $d \in D$ *with data* e*, which is carried out only when the condition* c *is fulfilled. When there is no condition, the event is represented by* e^d*.* Events may include information identifying who sent the message. So, it provides a generic communication system that can implement FIPA or KMQL [5].

3.3 Input Devices and Event Generators

It is necessary to establish the independence between the system and the input devices that generate events (hardware or software). So, the events needed to make the system respond to a set of input devices must be defined. A new function called *event generator* is defined as: *Let* $C^d(t)$ *be a function which creates events of type* d *at the time instant* t*, where* $d \in D$ *and* D *is the set of event types which can be generated by the system.*

It is important to note that event generators encapsulate the device-dependent code. They also can model the communication processes that exist in a MAS (agent-agent and agent-environment communication).

The process which obtains the events produced by input devices and their associated generators is defined as follows: *Let* C^* *be the set of all the event generators which are associated with input devices and* $E(C^*, t)$ *the function*

that collects all the events from all the generators, then:

$$E(C^*, t) = \left\{ \begin{array}{ll} e(z, C_i(t)) & if\ z = E(C^* - C_i, t) \\ \epsilon & if\ C^* = \emptyset \end{array} \right\} \quad (6)$$

$$\text{where } e(z, e^i) = \left\{ \begin{array}{ll} z \cdot e^i & if\ e^i \notin z \\ z & if\ e^i \in z \end{array} \right\}$$

3.4 System Algorithm

Once all the elements involved in the model to manage a MAS have been defined, the algorithm which carries out the entire system can be established, as shown in table 2

In table 2, w_o is the initial string of the system, e^* are all the events generated by the system in a frame t, $G^* = \{$All the event generators which generate events of type $D\}$, $D = \{$All the possible events in the system$\}$, $V = \{$All the visual events$\}$ where $V \subseteq D$, e^v are all the events from visual devices, e^u are all the events from non-visual devices, and g is the output device.

Steps 2, 3 and 4 manage the system events. In step 5, the evolution algorithm is called to obtain the string for the next frame. In steps 6 and 7, the visualization of the system is performed. In step 8, the next iteration is prepared. Step 9 checks if the current string satisfies the condition of completion: if the following string is empty the algorithm ends (Step 11), otherwise the algorithm continues.

Step 5 and 6-7 can be parallelized because they do not share any data, so it would lead to a faster system performance.

Table 2: System algorithm

1.	$w = w_o$; $t = 0$
2.	$e^* = E(G^*, t)$
3.	$e^v =$ events of e^* where $v \in V^+$
4.	$e^u = e^* - e^v$
5.	$w_{next} = \eta(w, e^u)$
6.	$v = \pi(w, e^v)$
7.	$g = \varphi(v)$
8.	$w = w_{next}$; $t = t + 1$
9.	If $w = \epsilon$ then go to 11
10.	Go to 2
11.	End

4 Case Study

An example is proposed to illustrate the use of Minerva. Specifically, a simple robot navigation simulator has been built. Only the basic features of robots have been taken into consideration, so the result is not a wholly functional simulator, but a starting point to develop a more complex one.

4.1 Problem Description

Let us consider a robot programmed to autonomously navigate in a known environment, and to transport objects from one place to another. It is equipped with several sensors that provide information about the environment. The information from the sensors can be used for different tasks. For instance, in our case, the inputs are the data from a range sensor, e.g. a laser, to detect obstacles and distances, and the image from a camera, to identify the objects and the places, using markers. As there are heterogeneous sensor inputs, some processes of sensor integration are needed for the robot to use the data of different nature to make decisions.

There is also a human supervisor controlling the robot tasks and giving some high level instructions, such as interrupt the current task, begin a new task or change the speed parameters.

A system like this can be modeled using a classical hybrid scheme (figure 1), based on the combination of a reactive system and a proactive system. The elements in the hybrid scheme and the elements of the Minerva model can be easily related:

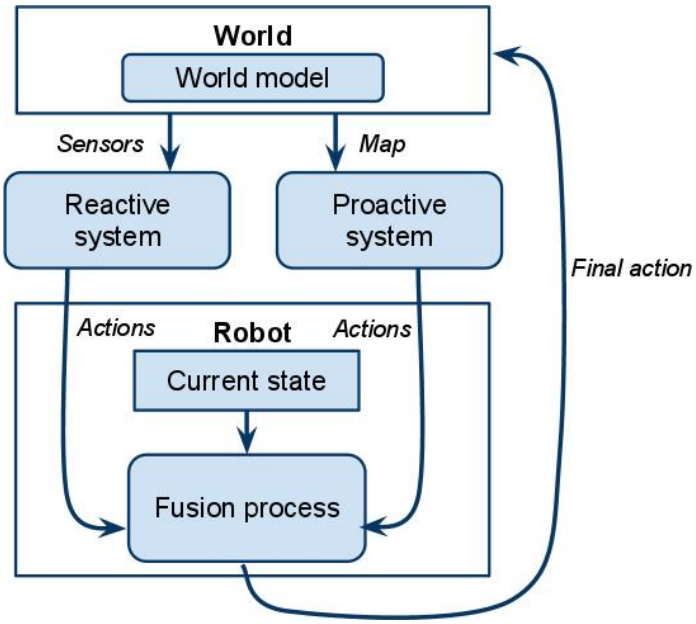

Figure 1: Hybrid scheme for a robotic system.

- The world is the real environment.
- The world model is a map, stored as an attribute of the agent robot.
- The robot is the only agent in the system.
- The reactive system is made of several generators, for the sensors and for the user's orders.
- The proactive system is the AI system of the robot, introduced in its evolution function.
- The current state is the robot state (set of attributes).
- The multi-sensorial fusion process is also introduced in the evolution function of the robot.

- The final action is the result of the process of sensor integration and the final action carried out by the robot.

4.2 Formalization

When describing a system with Minerva, the following aspects must be considered:

- The static displayable elements of the scene are described through the use of primitives, and the way to change their aspect are the transformations.
- The dynamic elements are the agents.
- For an agent to be displayable, there must be an associated primitive.
- Every aspect about the activities of the agents must be implemented in their evolution function, including AI algorithms, reactions to physical phenomena, and so on.
- The events are the way to activate the actions of the agents and to communicate among them. The events can have themselves some attributes.
- The agent attributes are the way to store all the information belonging to the agent.

4.2.1 Primitives and transformations

In our robotic system, only two primitives are needed, the map and the robot. The robot can be modified by two possible transformations: move and rotate

Table 3: Primitives and transformations of the robotic system

Symbol	Meaning
$PMap$	Draw the map
$PRobot$	Draw the robot
$TMove_{<dist>}$	Move a distance $dist$
$TRotate_{<angle>}$	Rotate an angle $angle$

(table 3). The primitives and the transformations will represent the operations carried out in the simulated robot, that is, the operations in the graphics system. The operations are performed by the semantic functions α for the primitives and β and γ for the transformations.

4.2.2 Events and generators

Each event is defined by its identifier and some attributes. They activate the changes on the agents, through their evolution functions. These events are produced by generators. There is a generator for each event type. In the robotic system, six generators are needed:

- gLaser: It generates an eLaser event when the laser detects an obstacle, by obtaining the laser data and processing them to find the possible obstacles.
- gCamera: It generates an eCamera event when a marker is detected in the camera image. This generator is the responsible of obtaining the images, processing them and detect the markers. Markers are used to identify the rooms in the environment.

- gDecide: It generates an eDecide event to report the system that a decision must be made. The system will use the accumulated information to activate the evolution function of the agents to make the correct decision, as it will be explained in the following sections.

- gRepresent: It generates an eRepresent event to indicate the system to represent the robot actions in the current representation space. For our simulator, the operations will take place in the graphics system. In this case, it is similar to the usual 'redraw' event of a typical graphics system.

- gObjective: It generates an eObjective event to set a new objective marker. This generator is connected to the user orders. Users can specify a new target room simply by selecting its associated marker.

- gSpeed: It generates an eSpeed event when the user changes the robot speed.

The generators in our system and their associated events are shown in table 4.

An order relation must be defined to establish an execution priority among generators. In the robotic system, the order relation is: gLaser, gCamera, gSpeed, gObjective, gDecide, gRepresent. Therefore, events related with the acquisition of data have the highest priority, compared with the events of decision and execution.

Table 4: Generators and events of the robotic system

Generator and events	Event description	Associated data
$gLaser = eLaser_{<dist,angle>}\ if\ obstacle$	The laser detects an obstacle	$dist, angle$: obstacle position
$gCamera = eCamera_{<marker>}\ if\ marker$	The camera detects a marker	$marker$: detected marker
$gDecide = eDecide\ each\ frame$	The robot makes a decision	No data
$gRepresent = eRepresent\ each\ frame$	The robot action is represented	No data
$gObjective = eObjective_{<marker>}\ if\ user\ order$	The user sets the objective marker	$marker$: objective marker
$gSpeed = eSpeed_{<speed>}\ if\ user\ order$	The user sets a new speed value	$speed$: robot speed

4.2.3 Agents

The only agent in our robotic system is the robot which is defined as:

$$ARobot^{eLaser,eCamera,eDecide,eRepresent,eObjective,eSpeed}_{<grid,row,column,angle,speed,objective,action>}$$

where *eLaser, eCamera, eDecide, eRepresent, eObjective, eSpeed* are the events which the agent is prepared to respond to, and *<grid, row, column, angle, speed, objective, action>* are the attributes that make up its state, whose meanings are:

- grid: is a matrix of *mxn* cells, representing the environment where the robot moves in. Each cell stores the registered data obtained from the sensors, that is, the detected obstacles and markers.
- row, column: position occupied by the robot in the grid.
- angle: robot orientation, defined by an angle.
- speed: robot speed, defined by a numerical value.
- objective: objective room, represented by its marker.
- action: string of primitives and transformations which indicates the next command to be executed.

To simplify, in the following equations this agent will be referred as $ARobot^{E}_{<g,r,c,ang,s,o,act>}$.

The evolution function is, probably, the most important element in the system, as it defines the way the robot behaves in the environment. Let e be an event that is received by the agent, the evolution function is defined as:

$\lambda(ARobot^{E}_{<g,r,c,ang,s,o,act>}, e) =$

$$\left\{\begin{array}{ll}
ARobot^{E}_{<g',r,c,ang,s,o,act>} & if\ e = eLaser_{<dist,angle>} \\
ARobot^{E}_{<g',r,c,ang,s,o,act>} & if\ e = eCamera_{<marker>} \\
ARobot^{E}_{<g,r',c',ang',s,o,act'>} & if\ e = eDecide \\
\alpha(ARobot^{E}_{<g,r,c,ang,s,o,act>}) & if\ e = eRepresent \\
ARobot^{E}_{<g,r,c,ang,s,o',act>} & if\ e = eObjective_{<marker>} \\
ARobot^{E}_{<g,r,c,ang,s',o,act>} & if\ e = eSpeed_{<speed>} \\
ARobot^{E}_{<g,r,c,ang,s,o,act>} & otherwise
\end{array}\right\}$$

where the symbol apostrophe (') on an attribute indicates that it is changed. The changes in the attributes are:

- If $e = eLaser_{<dist,angle>}$, the grid ($g$) must be updated to indicate that an obstacle has been detected. The cell to mark is the one in position $(r + dist\ \cos(ang + angle), c + dist\ \sin(ang + angle))$.
- if $e = eCamera_{<marker>}$ the grid (g) must be updated to indicate that a marker has been detected. The cell to mark is the one in position $(r + dist\ \cos(ang), c + dist\ \sin(ang))$.

- if $e = eDecide$, the current position and orientation of the robot row (r), column (c) and angle (ang), must be updated, as well as the actions to be executed. This function is very important, as it provides the behavior of the robot. In the section 4.2.5, the way to introduce intelligent behaviors and the Physics engine will be shown.

- if $e = eRepresent$, the robot must be executed in the representation space, through the use of the α function.

- if $e = eObjective_{<marker>}$, a new objective has been set by the user, so the objective (o) must be changed to the new one ($marker$).

- if $e = eSpeed< speed >$, a new value for the speed has been set by the user, so the speed (s) must be updated.

In any other case, the agent must remain unchanged.

4.2.4 Initial string

The initial string in our system defines its initial state. It is the string

$$PMap \cdot ARobot^{eLaser,eCamera,eDecide,eRepresent,eObjective,eSpeed}_{<grid,row,column,angle,0,\epsilon,\epsilon>}$$

where the attribute grid is initialized to a set of empty cells, the attributes row, column and angle are initialized to the initial position and orientation of the robot, the initial speed is 0, and the objective and the actions are defined as empty.

4.2.5 Evolution function

As it was stated in section 3.2.3, the evolution function is the way of introducing behaviors in an agent. The key is implementing the AI algorithms and the Physics engine into this function, given that all the needed elements are known. In this example, only a basic intelligent behavior is introduced.

In the robot simulator, the intelligent actions are triggered when an *eDecide* event is received. The current grid is known, since it has been built as the *eLaser* and *eCamera* events have been received, the current position and orientation of the agent is also known, the speed has been defined by the user, and the objective has also been fixed by the *eObjective* event. The implemented algorithm must decide the new position and orientation, and the next action to do. The specific algorithm to make this decision is up to the user, since the aim of this work is not to develop an AI algorithm to achieve the goal but to give a well-structured framework. In our case, to prove that any intelligent behavior can be introduced by just changing the evolution function, two simple decision algorithms have been chosen to decide how the robot should move in the world. The first algorithm is the simplest one: make decisions randomly to find the target position. The second algorithm is the A* algorithm [11], considering the Euclidean distance to the goal as the weights. If there is an obstacle the distance is defined as infinite.

4.3 Features of the system

One of the main features of our model is that the system definition is independent from the input devices. For instance, in our original system, a laser range sensor was used to detect obstacles. However, any other sensor may be introduced. To add a new device, just a new event generator must be defined, to create events of the same type that the ones generated by the laser generator. That is, it provides the same information: the angle and the distance to the obstacle. The new device is then introduced with no other modification in the system. The new device is then used to replace the laser device or to obtain redundant information for the detection of obstacles.

Maybe, the most important achievement in the proposed model is the fact that the description for the simulation can be used with a different representation system of even with the real robot with minor changes. In fact, the system definition, i.e. the string representing the system, remains exactly the same. To achieve this goal, only the generator for the execution of the robot commands and the visualization functions must be changed. The commands are transparently executed no matter whether the robot is real or simulated in any representation system, just using the appropriate generator. As a result, the navigation would be exactly the same for the simulated robot and for the real one, if there were not odometry errors. An example of this feature is shown in figures 2 and 3, with two different representations, in 2D with sprites and in 3D with OpenGL.

The proposed model is, by definition, easily extensible, too. The updating

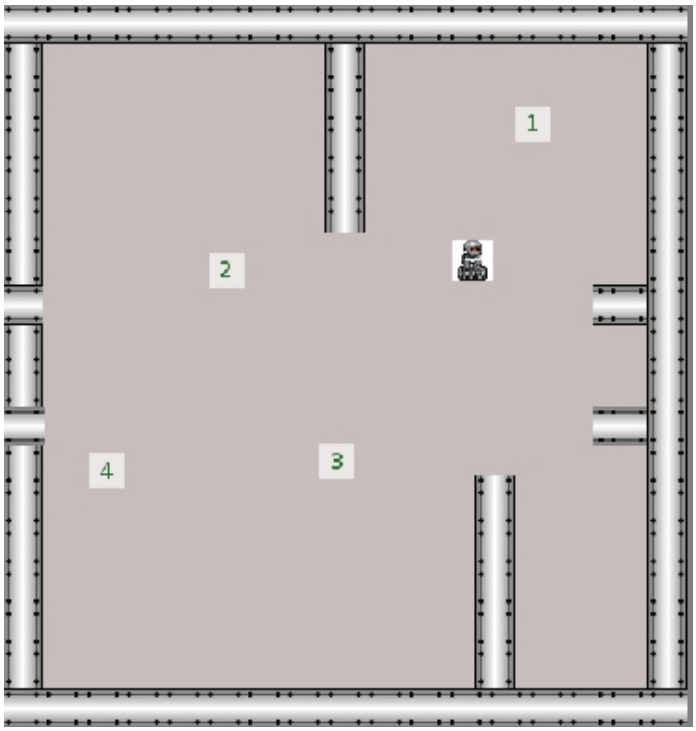

Figure 2: Sprite view in 2D.

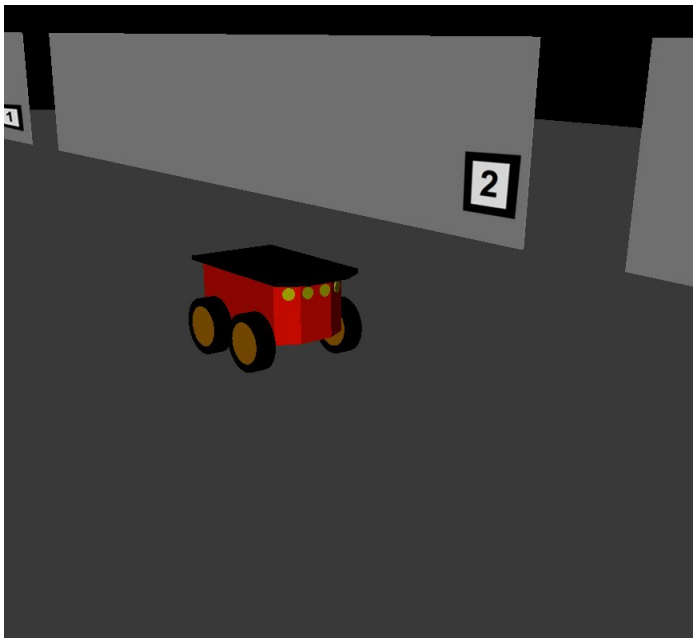

Figure 3: OpenGL view in 3D.

of the definition string supposes the extension of the model and the addition of new features. Moreover, most elements can be reused in new definition strings to obtain new behaviors with little effort. In our case, new instances of the agent symbols (representing robots) have been added to the definition string to extend the system in an almost immediate way.

A good way to improve the simulation is introducing some odometry errors in the motors and in the sensor signals, accordingly with the features of the real robot.

5 Conclusions and Future Work

A new agent-based framework to model virtual scenes, Minerva, has been presented. The major goal of this system is to take advantage of the MASs features to model a whole VR system, including the AI modules, the Physics engine and the graphics system. The proposed model uses a context-free grammar.

The key of the proposed model is the evolution function: it allows the definition of the behaviour of the agents, e.g. the evolution of the elements in time. This way, the virtual world can be modeled as a set of static and dynamic elements, that have a visual representation, evolve during the execution and behave in accordance to AI algorithms or reacting to Physics phenomena.

The presented model separates the hardware-dependent implementation of the interaction devices from the description of the scene. This separation is made by event generators, that create a layer between the hardware and the representation of the system. The generators are mathematical models used to transform device actions, both visual and input devices, into more general actions. The system must be able to identify these general actions regardless of the origin of the action. It is achieved using abstraction. As a consequence of these features, the different elements of the system can be easily reused.

The AI engine can be implemented in the evolution function of the agents. Agents use these functions to make decisions according to their current status. Moreover, events trigger activities which can change the status and the behaviour of agents.

The framework has proven its usefulness, since a VR system has been built

using it. The resulting model has shown to be expressive enough to model the basic features of a robot simulator, such as the control of a mobile robot, the navigation in a given environment, and the process of a set of several inputs from different types of sensors. Moreover, the system can be directly implemented on a real robot, just making minor changes. In fact, the only thing to do is changing the representation space, using the real robot instead of the OpenGL environment used in the example. It would be achieved by redefining functions α, β and δ to make actions on the real robot.

The model presented in this work is currently under development. It is pretended to continue developing several issues. We consider that the Physics phenomena can be easily defined. It could be achieved by setting up different types of event generators. Depending on the physical features of the device, it would activate the activity of the needed agent to react to that physical process. For example, if an agent has to react to collisions, the event generator of this type would calculate the collisions between elements by extracting the scene geometry from the graphics engine (it would use the implementation of the functions α, β and δ to calculate the bounding box of the elements). Then, it would generate the events needed to react to such collisions. This event generator could be implemented with hardware if the system allowed it. The integration between the Physics engine and the AI engine is also guaranteed because both engines are connected by events.

Another point to investigate is the optimization of the algorithm and its parallelization. The definition of the system through strings facilitates the pos-

sibility of parallel algorithms. From another point of view, strings represent the states of the system and its evolution. This evolution may change through mutations, so different evolutive solutions may be conceived to design new systems. We also consider the possibility of a new type of events which are activated with a certain probability. For example, if an agent is defined as $a^{d,h}$ and it receives the event e^d, then the function associated with this event will be carried out only with a certain probability.

In conclusion, the main aim has been to design a reusable, integral and generic system, which can be easily scaled, adapted, and improved. It is also important that the core of the system (the evolution in time) is independent from its representation and the elements which it interacts with.

References

[1] Jack. http://aosgrp.com/products/jack/ (Accessed March 2014)

[2] Axelrod, R.: Advancing the art of simulation in the social sciences (1997)

[3] Chiu, C.C., Marsella, S.: A style controller for generating virtual human behaviors. In: Proceedings of 10th. Int. Conf. on Autonomous Agents and Multi-Agent Systems, pp. 1023–1030 (2011)

[4] Dastani, M.: 2apl: a practical agent programming language. Autonomous Agents and Multi-Agent Systems **16**(3), 214–248 (2008). DOI 10.1007/s10458-008-9036-y. URL http://dx.doi.org/10.1007/s10458-008-9036-y

[5] Genesereth, M.R., Ketchpel, S.P.: Software agents. Communications of the ACM **37**, 48–53 (1994)

[6] Gilbert, N.: Agent-Based Models. SAGE Publications (2008)

[7] Hahn, C., Madrigal-Mora, C., Fischer, K.: A platform-independent metamodel for multiagent systems. Autonomous Agents and Multi-Agent Systems **18**, 239–266 (2009). DOI 10.1007/s10458-008-9042-0

[8] Hbner, R.H.B.J.F., Wooldridge, M.: Programming Multi-agent Systems in AgentSpeak Using Jason. Wiley-Blackwell (2007)

[9] Khoo, A., Zubek, R.: Applying inexpensive ai techniques to computer games. IEEE Intelligent Systems pp. 48–53 (2002)

[10] Luke, S., Cioffi-revilla, C., Panait, L., Sullivan, K.: Mason: A new multi-agent simulation toolkit. In: University of Michigan (2004)

[11] Luo, R., Lin, Y.C., Kao, C.C.: Autonomous mobile robot navigation and localization based on floor plan map information and sensory fusion approach. In: Multisensor Fusion and Integration for Intelligent Systems (MFI), 2010 IEEE Conference on, pp. 121 –126 (2010). DOI 10.1109/MFI.2010.5604478

[12] Maes, P., Darrell, T., Blumberg, B., Pentland, A.: The alive system: wireless, full-body interaction with autonomous agents. Multimedia Syst. **5**, 105–112 (1997). DOI 10.1007/s005300050046

[13] North, M.J., Howe, T.R., Collier, N.T., Vos, J.R.: The repast simphony runtime system. Proceedings of the Agent 2005 Conference on Generative Social Processes Models and Mechanisms (1), 151–158 (2005)

[14] Novak, J.: Game Development Essentials: An Introduction. Second edition. Delmar Cengage Learning (2007)

[15] Querrec, R., Buche, C., Lecorre, F., Harrouet, F.: Agent metamodel for virtual reality applications. In: D. Ryzko, H. Rybinski, P. Gawrysiak, M. Kryszkiewicz (eds.) Emerging Intelligent Technologies in Industry, *Studies in Computational Intelligence*, vol. 369, pp. 81–90. Springer Berlin / Heidelberg (2011)

[16] Reynolds, C.: Interaction with groups of autonomous characters. Group **21**(4), 449–460 (2000)

[17] Rhyne, T.M.: Entertainment computing: Computer games' influence on scientific and information visualization. Computer **33**, 154–156 (2000). DOI http://doi.ieeecomputersociety.org/10.1109/2.889099

[18] Sherman, W.R., Craig, A.: Understanding virtual reality: interface, application, and design. Morgan Kaufmann. (2003)

[19] Thiebaux, M., Lance, B., Marsella, S.: Real-Time Expressive Gaze Animation for Virtual Humans. In: 8th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS). Budapest, Hungary (2009)

[20] Trescak, T., Esteva, M., Rodriguez, I.: A virtual world grammar for automatic generation of virtualworlds. The Visual Computer **26**, 521–531 (2010). DOI 10.1007/s00371-010-0473-7

[21] Trescak, T., Rodriguez, I., Sanchez, M.L., Almajano, P.: Execution infrastructure for normative virtual environments. Engineering Applications of Artificial Intelligence **26**(1), 51 – 62 (2013). DOI 10.1016/j.engappai.2012.09.016

[22] Ulicny, B., Thalmann, D.: Crowd simulation for interactive virtual environments and vr training systems. In: Proceedings of the Eurographic workshop on Computer animation and simulation, pp. 163–170. Springer-Verlag New York, Inc., New York, NY, USA (2001)

[23] Wachsmuth, I., Tao, Y.: Interactive graphics design with situated agents. In: Graphics and Robotics, pp. 73–86. Springer-Verlag, London, UK (1995)

[24] Wilensky, U.: Netlogo. center for connected learning and computer-based modeling, northwestern university, evanston, il. http://ccl.northwestern.edu/netlogo (1999)