A Grammatical Approach to the Modeling of an Autonomous Robot

Gabriel López-García, A. Javier Gallego-Sánchez, J. Luis Dalmau-Espert, Rafael Molina-Carmona and Patricia Compañ-Rosique

***Abstract* — Virtual Worlds Generator is a grammatical model that is proposed to define virtual worlds. It integrates the diversity of sensors and interaction devices, multimodality and a virtual simulation system. Its grammar allows the definition and abstraction in symbols strings of the scenes of the virtual world, independently of the hardware that is used to represent the world or to interact with it. A case study is presented to explain how to use the proposed model to formalize a robot navigation system with multimodal perception and a hybrid control scheme of the robot. The result is an instance of the model grammar that implements the robotic system and is independent of the sensing devices used for perception and interaction. As a conclusion the Virtual Worlds Generator adds value in the simulation of virtual worlds since the definition can be done formally and independently of the peculiarities of the supporting devices.**

***Keywords* — Autonomous robots, virtual worlds, grammatical models, multimodal perception.**

I. Introduction

Autonomous robots are physical agents that perform tasks by navigating in an environment and by manipulating objects in it. To perform these tasks, they are equipped with effectors to act on the environment (wheels, joints, grippers...) and with sensors that can perceive it (cameras, sonars, lasers, gyroscopes...). It should be notice that, in general, the environment in which a robot operates may be inaccessible (it is not always possible to obtain all the information necessary for decision-making in every moment) non-deterministic (the effect of the action taken by the robot in the environment cannot be guaranteed), non-episodic (the action to be performed by the robot depends on the current perceptions and on the previous decisions), dynamic (the robot and the other elements in the environment may be constantly changing) and continuous (the location of the robot and the moving obstacles change in a continuous range of time and space) [8].

The growing disparity of available sensors adds complexity to systems, but it also allows the control of robots to be more accurate. There are several reasons that support the use of a combination of different sensors to make a decision. For example, humans and other animals integrate multiple senses. Various biological studies have shown that when the signals reach the superior colliculus converge to the same target area [9], which also receives signals from the cerebral cortex and causes the resulting behavior. A large majority of superior colliculus neurons are multisensory. There are other reasons of mathematical nature: combining multiple observations from the same source provides statistical advantages because some redundant observations are obtained for the same estimation.

The concepts from biology can be extrapolated to the field of robotics. In fact, one of the current research fields that arouses most interest is the management of several inputs from different types, the so called multimodal data.

Combining data from different sensors is an open field of research. In this sense, there are several concepts related to this subject that deals with the concept of multimodality from different points of view. Signhal and Brown [10] consider that two main processes may be performed from several multimodal inputs: multisensor fusion and multisensor integration. Multisensor integration refers to the synergistic use of the information provided by multiple sensory devices to assist in the accomplishment of a task by a system. Multisensor fusion refers to any stage in the integration process where there is actual combination (fusion) of different sources of sensory information into one representation format. Other authors describe the evidence that humans combine information following two general strategies: The first one is to maximize information delivered from the different sensory modalities (sensory combination). The second strategy is to reduce the variance in the sensory estimate to increase its reliability (sensory integration) [3]. Another example is set in [11]. They consider that, in general, multimodal integration is done for two reasons: sensory combination and sensory integration. Sensory combination describes interactions between sensory signals that are not redundant. That means crossmodal integration leads to increased information compared to single modalities. By contrast, sensory integration describes interactions between redundant signals. This leads to enhanced robustness and reliability of the derived information.

In this paper we deal with the integration of multimodal inputs in the sense stated by Signhal and Brown [10], that is, the use of data of different nature for decision-making in high-level tasks performed by a robot. However, the proposed system can also deal with the concept of fusion, defined as the combination of low-level redundant inputs for the cooperative construction of the complete information of the environment, reducing, as a consequence, the levels of uncertainty.

Different architectures have been described for defining the behavior of a robot and the combination of sensory

Gabriel López-García, A. Javier Gallego-Sánchez (corresponding author), J. Luis Dalmau-Espert, Rafael Molina-Carmona and Patricia Compañ-Rosique are in the Group of Industrial Computing and Artificial Intelligence, University of Alicante, Ap. 99, 03080 Alicante, Spain (e-mail: [glopez, ajgallego, jldalmau, rmolina, patricia]@dccia.ua.es).

information. A robotic control architecture should have the following properties: programmability, autonomy and adaptability, reactivity, consistent behavior, robustness and extensibility [4].

To achieve those requirements, most robot architectures try to combine reactive control and deliberative control. The reactive control is guided by sensors and it is suitable for low-level decisions in real time. The deliberative control belongs to a higher level, so that global solutions can be obtained from the data collected by the sensors but also from information from an a priori model. They are, therefore, hybrid architectures.

Hybrid architectures arise due to the problems and inconveniences of pure reactive approaches, such as the lack of planning, and of pure deliberative approaches, such as the slow reactions. An example of hybrid architecture is the PRS (Procedural Reasoning System). When the hybrid architectures face a problem, the deliberative mechanisms are used to design a plan to achieve an objective, while the reactive mechanisms are used to carry out the plan. The communications framework is the base that enables the necessary interaction between reactive and deliberative levels, by sending distributed sensory information to tasks at both levels and sending actions to actuators. Deliberative and reactive tasks can be structured in a natural way by means of independent software components [6].

An example of implementation is the model SWE (Sensor Web Enablement), which is applied to systems that are based on the use of sensors to obtain the information that is processed later [1]. In [7] an architecture based on models SWE and DDS (Data Distribution Service) is proposed. DDS is a general-purpose middleware standard designed specifically to satisfy the performance and Quality of Service (QoS) requirements of real-time systems.

The Virtual Worlds Generator (VWG), our proposal, is a grammatical model, which integrates the diversity of interaction and sensing devices and the modules that make up a Graphics System (Graphics, Physics and AI engines). The scene definition is separated from the hardware-dependent characteristics of the system devices. It uses a grammar definition, which integrates activities, visualization and interaction with users. The hypothesis is that it can be used as a formal framework to model a robot navigation system, including several multimodal inputs, sensor fusion and integration, and behavior strategies.

In section 2, the formal model for the VWG is presented. In section 3, the formal model is applied to construct a robotic system. Finally, some conclusions are presented in the last section.

II. MODEL FOR VIRTUAL WORLDS GENERATION

In the VWG model, a virtual world is described as an ordered sequence of primitives, transformations and actors. A primitive is the description of an object in a given representation system (typically, they are graphical primitives but they could also be sounds or any other primitive in a representation space). Transformations modify the behavior of primitives, and actors are the components that define the activities of the system in the virtual world. The actors may be finally displayed through primitives and transformations. To model the different actor's activities, the concept of an event is used. Events cause the activation of a certain activity that can be processed by one or more actors.

Each element in the scene is represented by a symbol from the *set of symbols of the scene*. The symbols make up strings that describe the scenes, in accordance with a language syntax, which is presented as a grammar [2].

A. Syntax

A grammar M is a tuple $M = <\Sigma, N, R, s>$, where Σ is the finite set of terminal symbols, N is the finite set of non-terminal symbols, R is the finite set of syntactic rules (a syntactic rule is an application $r: N \rightarrow W^*$, where $W = \Sigma \cup N$) and s ∈ N is the initial symbol of the grammar. In our case, M is defined as:

- $\Sigma = P \cup T \cup O \cup A^D_{ATTR}$, where:
  - P: set of symbols for primitives.
  - T: set of symbols for transformations.
  - $O = \{\cdot\ ()\}$: symbols for indicating the scope () and the concatenation ·.
  - A^D_{ATTR}: set of symbols for actors, where D is the set of all the types of events generated by the system and $ATTR$ is the set of all the attributes of actors, which define all the possible states. For example, the actor a^H_{attr} will carry out its activity when it receives an event e^h, where $h \in H$, $H \subseteq D$ and $attr \in ATTR$ is its current state.
- N = {WORLD, OBJECTS, OBJECT, ACTOR, TRANSFORM, FIGURE}.
- Grammar rules R are defined as:
  - Rule 1. **WORLD** →OBJECTS
  - Rule 2. **OBJECTS** → OBJECT | OBJECT · OBJECTS
  - Rule 3. **OBJECT**→ FIGURE | TRANSFORMATION | ACTOR
  - Rule 4. **ACTOR**→ a^H_{attr} , $a^H_{attr} \in \mathbf{A}^D_{ATTR}$, $H \subseteq D$
  - Rule 5. **TRANSFORMATION** → t(OBJECTS), $t \in T$
  - Rule 6. **FIGURE**→ p+, $p \in P$
- s = WORLD is the initial symbol of the grammar.

M is a context-free grammar. $L(M)$ is the language generated by the grammar M: $L(M) = \{w \in \Sigma^* \mid \text{WORLD} \rightarrow^* w\}$.

B. Semantics

Apart from the language syntax, it is necessary to define the semantics of $L(M)$. It will be defined with a denotational method, that is, through mathematical functions.

1) Semantic Function of Primitives (Rule 6)

Rule 6 defines a figure as a sequence of primitives. Primitive's semantics is defined as a function α, as follows:

$$\alpha = P \to G \tag{1}$$

Each symbol in the set P carries out a primitive on a given geometric system G. So, depending on the definition of the function α and on the geometry of G, the result of the system may be different. G represents the actions to be run on a specific visual or non-visual geometric system (e.g. the actions on OpenGL or on the system of a robot). The function α provides the abstraction needed to homogenize the different implementations of a rendering system. Therefore, only a descriptive string is needed to run the same scene on different systems.

2) Semantic Functions of Transformations (Rule 5)

In Rule 5, two functions are used to describe the semantics of a transformation, whose scope is limited by the symbols "()":

$$\begin{aligned} &\beta : T \to G \\ &\delta : T \to G \end{aligned} \tag{2}$$

β represents the beginning of the transformation. It is carried out when the symbol "(" is processed. Function δ defines the end of the transformation which has previously been activated by the function β. It is run 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.

3) Semantic Functions of Actors (Rule 4)

Rule 4 refers to actors, which are the dynamic part of the system. The semantics of the actor is a function that defines its evolution in time. For this reason, the semantic function is called *evolution function* λ and it is defined as

$$\lambda : A_{ATTR}^{D} \times E^{D} \to L(M) \tag{3}$$

where E^D is the set of events for the set of all event types D. Some deeper aspects about events will be discussed later.

The function λ has a different expression depending on its evolution. However, a general expression can be defined. Let $H = \{h_0, \ldots, h_n\} \subseteq D$ be the subset of event types which the actor a^H_{ATTR} is prepared to respond to. The general expression for λ is:

$$\lambda\left(a_{ATTR}^{H}, e^{h}\right) = \begin{cases} u_0 \in L(M) & if\ h = h_0 \\ \cdots & \\ u_n \in L(M) & if\ h = h_n \\ a_{ATTR}^{H} & if\ h \notin H \end{cases} \tag{4}$$

where $u_0, \ldots, u_n$ are strings of $L(M)$. This equation means that an actor a^H_{ATTR} can evolve, that is, it is transformed into another string u_i when it responds to an event e^h which the actor is prepared to respond to. However, the actor remains unchanged when it is not prepared to respond.

As well as dynamic elements, actors can also have a representation in the geometric space G. To be displayed, an actor must be converted to a string of primitives and transformations. This visualization function is defined as:

$$\theta : A_{ATTR}^{D} \times E^{V} \to L(M') \tag{5}$$

where $V \subseteq D$, $E^V \subseteq E^D$ are events created in the visualization process, and $L(M')$ is a subset of the language $L(M)$, made up of the strings with no actors. Let $H \cap V = \{v_0, \ldots, v_n\} \subseteq D$ be the subset of visual event types which the actor a^H_{ATTR} is prepared to respond to. The expression of Θ is defined as:

$$\theta\left(a_{ATTR}^{H}, e^{v}\right) = \begin{cases} z_0 \in L(M') & if\ v = v_0 \\ \cdots & \\ z_n \in L(M') & if\ v = v_n \\ \varepsilon & if\ v \notin H \cap V \end{cases} \tag{6}$$

4) Semantic Functions of OBJECT, OBJECTS and WORLD (Rules 1, 2 and 3)

The semantic function of Rules 1, 2, and 3 breaks down the strings and converts them into substrings, executing the so called *algorithm of the system*, which performs the complete evolution of the system and displays it in the current geometric system. It performs several actions, which are described in the following paragraphs.

To display the scene on the geometric system G, the function φ is defined, for the set of symbols that can directly be displayed: primitives and transformations. Given a string $w \in L(M)$ and using only symbols of P and T, φ is defined as:

$$\varphi(w) = \begin{cases} \alpha(w) & if\ w \in P \\ \beta(t);\varphi(v);\delta(t) & if\ w = t(v) \wedge v \in L(M) \wedge t \in T \\ \varphi(u);\varphi(v) & if\ w = u \cdot v \wedge u, v \in L(M) \end{cases} \tag{7}$$

In the case of strings including both displayable elements, and actors, two functions must be defined. The first one is the so called *function of the system evolution* η, which requires a sequence of sorted events $S = e^1 \cdot e^2 \ldots e^n$, where every $e^i \in E^D$ and a string of $L(M)$ including actors, and implements a set of recursive calls to the function λ to perform the evolution of all the actors in the system at a given frame:

$$\eta(w, S) = \begin{cases} w & if\ w \in P \\ t(\eta(v, S)) & if\ w = t(v) \\ \prod_{e^i \in S} \lambda(a_{attr}^{H}, e^i) & if\ w = a_{attr}^{H} \\ \eta(u, S) \cdot \eta(v, S) & if\ w = u \cdot v \end{cases} \tag{8}$$

The operator $\Pi_{ei \in S}\, \lambda\, (a^H_{attr}\, ,\, e_i)$ concatenates the strings of the function λ.

The actors to be displayed in the system must be converted to displayable elements, that is, primitives and transformations. The second function, returns a string of the language $L(M')$ given a string $w \in L(M)$ and a sequence of ordered visualization events $S' = e^1 \cdot e^2 \ldots e^n$, where every $e^i \in E^V$ and $S' \subseteq S$. This function is called *function of system visualization* π and it is defined as:

$$\pi(w,S) = \begin{cases} w & if\ w \in P \\ t(\eta(v,S')) & if\ w = t(v) \\ \prod_{e^i \in S} \theta(a^H_{ATTR}, e^i) & if\ w = a^H_{ATTR} \\ \pi(u,S') \cdot \pi(v,S') & if\ w = u \cdot v \end{cases} \quad (9)$$

C. Events and Generators

The events are the mechanism to model the activity in the system. The actors' activity is carried out when a certain type of event is produced. The following event definition is established: e^d_c *is defined as an event of type* $d \in D$ *with data* c.

A new function called *event generator* is defined as: Let $C^d(t)$ be a function which creates a sequence of ordered 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. This function is:

$$C^d : Time \rightarrow (E^D)^* \quad (10)$$

In the previous definition, it should be noticed that events are generated in the time instant t. It is due to synchronization purpose. The event generator can generate several or no events at a given moment.

Different event generators can create the same type of events. So, a priority order among event generators must be established to avoid ambiguities. Given two generators C_i and C_j which create the same event, if $i < j$, then the events generated by C_i will have a higher priority.

D. System Algorithm

Once all the elements involved in the model have been defined, the *System Algorithm* can be established. It defines the system evolution and its visualization at every time instant t or frame:

1) $w = w_0$; $t = 0$
2) while $w \neq \varepsilon$ do
   - S = collect events from generators C^* in order of priority.
   - Z = extract visual events from S.
   - $w_{next} = \eta(w, S)$
   - $v = \pi(w, Z)$; $g = \varphi(v)$
   - $w = w_{next}$; $t = t + 1$
3) end while

where w_0 is the initial string, C^* = {All the event generators which generate events of type D}, D = {Set of all the types of possible events in the system}, g is the output device, S is a sequence of all the events generated by the system at instant t, Z is a subsequence of S, and it includes all the events from visual devices. These events are the input of the visual algorithm π.

A diagram of the virtual world generation algorithm is shown in Fig. 1.

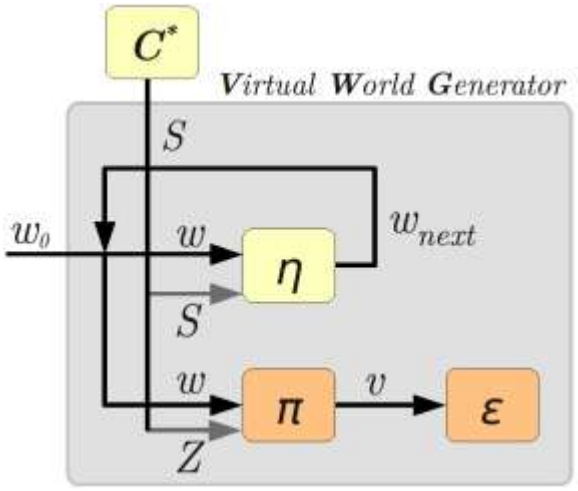

Fig. 1. Virtual world generator algorithm.

This formalization of the system has two main consequences. First, the scene definition is separated from the hardware-dependent characteristics of components. The functions α, β and δ provide the independence from the visualization system, and the event generators provide the independence from the hardware input devices. Secondly, due to the fact that there is a specific scheme to define the features of a system, the different system elements can be reused easily in other areas of application.

III. CASE STUDY

A. Description

Let us consider a robot with several sensors that provide information about the environment. It is programmed to autonomously navigate in a known environment, and to transport objects from one place to another. The input data are: the data from a range sensor (e.g. a laser to detect obstacles and distances), the image from a camera to identify objects and places using markers, an internal representation of the environment (a map) and a human supervisor who is controlling the robot (he can give some high level instructions, such as interrupt the current task or begin a new task). The information is combined using a multimodal algorithm based on priorities, so that the robot can attend to the users' request, select the best way to follow to the destination and use the sensors to detect and avoid obstacles, as well as to identify the objects and the places.

A system like this can be modeled using a classical hybrid scheme (Fig. 2), based on the combination of a reactive system

and a proactive system. This hybrid scheme can be adapted using the VWG introduced in the previous section.

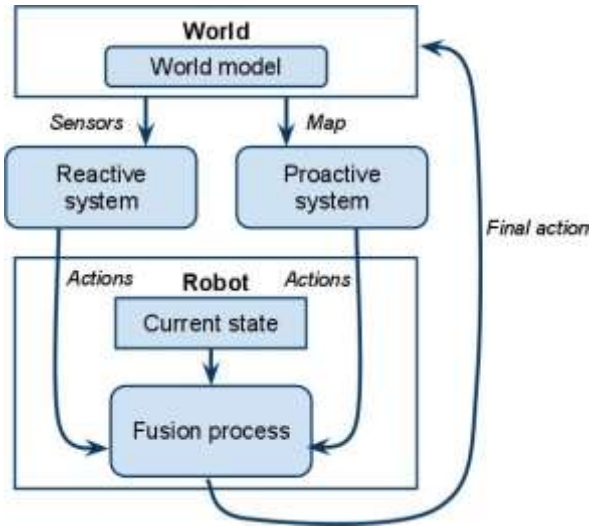

Fig. 2. Hybrid scheme for a robotic system.

In this picture the world is the real environment. The world model is a map containing the static elements of the environment. The reactive system is made of several generators, for the sensors and for the user's orders. The proactive system is the AI of the robot. The robot is the only actor in the system. The current state is the set of robot attributes. The multisensorial integration process is 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.

B. Primitives and Transformations

As it was stated in section 2, primitives are the description of objects in the space of representation, and transformations are used to modify primitives. In our robotic system, only one primitive is needed, the robot, and it is modified by two possible transformations: move and rotate (table I). When the system is executed in a real environment, the robot primitive represents the real robot and the transformations correspond to the actual operations performed by the robot. If it is executed in a simulator, the primitive and the transformations will represent the operations carried out in the simulated robot, that is, the operations in the graphics system (GS). The operations are performed by the semantic functions α for the primitives and β and δ for the transformations.

TABLE I
PRIMITIVES AND TRANSFORMATIONS OF THE ROBOTIC SYSTEM

	Real Environment	Simulator
PRobot	No action	Draw the robot in the GS
TMove$_{<dist>}$	Move a distance *dist*	Move a distance *dist* in the GS
TRotate$_{<angle>}$	Rotate an angle *angle*	Rotate an angle *angle* in the GS

C. Events and Generators

Events are used to define the activity in the system. Each event is defined by its identifier and some attributes. They produce changes on the actors through their evolution functions. These events are produced by generators. There is a generator for each event type. In the robotic system, five 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. Markers are used to identify the rooms in the environment.
- *gDecide*: It generates an *eDecide* event each frame to indicate to the robot to make a decision.
- *gExecute*: It generates an *eExecute* event to indicate the system to execute the robot actions in the current representation space. If the representation space is the real environment, the real operations will take place (move the robot, rotate the robot...). If the current space is the simulator, the operations will take place in the graphics system.
- *gObjective*: It generates an *eObjective* event to set a new objective marker. This generator is connected to the users' orders. Users can specify a new target room simply by selecting its associated marker.

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

TABLE II
GENERATORS AND EVENTS OF THE ROBOTIC SYSTEM

Generator and Events	Description	Associated data
gLaser = *eLaser*$_{<dist,angle>}$ if obstacle	Event produced when the laser detects an obstacle	*dist*: disntace to the obstacle *angle*: angle to the obstacle
gCamera = *eCamera*$_{<marker>}$ if marker	Event produced when the camera detects a marker	*marker*: detected marker
gDecide = *eDecide* each frame	Event generated each frame to indicate to the robot to make a decision	No data
gExecute = *eExecute* each frame	It runs the robot action in the real environment or in the simulator	No data
gObjective = *eObjective*$_{<marker>}$ if user order	Event produced by the user to set the objective marker	*marker*: objective marker

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

D.Actors

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

$$ARobot^{eLaser,eCamera,eDecide,eExecute,eObjective}_{<grid,row,column,angle,objective,action>} \quad (11)$$

where the superscript are the events which it is prepared to respond to, and the subscript are the attributes, whose meanings are: the *grid* represents the environment where the robot moves in. Each cell stores the registered data obtained from the sensors (the detected obstacles and markers). *Row* and *column* are the position occupied by the robot in the grid. *Angle* is the robot orientation. *Objective* is the objective room, represented by its marker. And *action* is the string of primitives and transformations that indicates the next command to be executed by the robot. To simplify, in the following equations this actor will be referred as $ARobot^{E}_{<g,r,c,an,o,ac>}$.

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

$$\lambda(ARobot^{E}_{<g,r,c,an,o,ac>}, e) = \\ = \begin{cases} ARobot^{E}_{<g',r,c,an,o,ac>} & if\ e = eLaser_{<dist,angle>} \\ ARobot^{E}_{<g',r,c,an,o,ac>} & if\ e = eCamera_{<marker>} \\ ARobot^{E}_{<g,r',c',an',o,ac'>} & if\ e = eDecide \\ \alpha(ARobot^{E}_{<g,r,c,an,o,ac>}) & if\ e = eExecute \\ ARobot^{E}_{<g,r,c,an,o',ac>} & if\ e = eObjective_{<marker>} \\ ARobot^{E}_{<g,r,c,an,o,ac>} & otherwise \end{cases} \quad (12)$$

where the symbol apostrophe (') on an attribute indicates that it has changed as a consequence of the received event. The way the attributes change is the following:

- 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 ($r + dist \cos(ang)$, $c + dist \sin(ang)$).
- 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 following section, the way to introduce intelligent behaviors will be shown.
- If *e* = *eExecute*, the actions of the robot must be executed in the representation space, through the use of the α function.
- If $e = eOb\ jective_{<marker>}$, a new objective has been set by the user, so the objective (*o*) must be changed to the new one (*marker*).
- In any other case, the actor must remain unchanged.

E. Initial string

The initial string in our systems defined as:

$$ARobot^{eLaser,eCamera,eDecide,eExecute,eObjective}_{<grid,row,column,angle,\varepsilon,\varepsilon>} \quad (13)$$

where the attribute *grid* is initialized to a set of empty cells, the attributes *row*, *column* and *angle* are the initial position and orientation, and the *objective* and the *action* are empty.

F. Analysis

A set of tests has been designed to prove the features of our model. Specifically, five tests have been carried out.

1) Test of the evolution function

As it was stated before, the evolution function is the way of introducing intelligent behaviors in an actor. Therefore, the aim of this test is to prove the suitability of the evolution function to introduce new AI algorithms. This test is not to obtain the best AI algorithm to achieve the goal, but to prove that a new intelligent behavior can be introduced by just changing the evolution function. An important question is guaranteeing the same conditions for all the experiments, so the AI algorithms are introduced with no other modification in other parts of the system.

Two simple decision algorithms have been used to decide how the robot should move in the world. The first algorithm makes decisions randomly to find the target position. The second one is the A* algorithm [5], considering the Euclidean distance to the goal as the weights. If there is an obstacle the distance is defined as infinite.

2) Test of device independence

One of the main features of our model is that the system definition is independent from the input devices. The aim of this test is to prove that the input devices can be replaced without changing the definition of the string representing the system.

In our original system, a laser range sensor was used to detect obstacles. In this test, a Kinect device is introduced. To add this new device, we have just designed a new event generator (*gKinect*) that creates 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 Kinect is then used to replace the laser device or to obtain redundant information for the detection of obstacles.

3) Test to validate the simulation

The most important achievement in the proposed model is the fact that the description for the simulation and for the real robot is exactly the same. That is, the command execution for the simulated robot can be directly used for the real robot with no change in the string that represents the system.

To achieve this goal, two generators for the execution of the robot commands have been implemented: one for the real robot and one for the robot simulation. This way, the commands are transparently executed no matter whether the robot is real or simulated, 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. 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.

4) Test of the system extensibility

The proposed model is, by definition, easily extensible. The updating 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 actor symbols (representing robots) have been added to the definition string to extend the system in an almost immediate way and to create a multi-robot system.

5) Test of changes in the environment

A desired capability in a robot navigation system is, obviously, to be flexible enough to work under very different conditions. To prove this feature, the system has been tested with different maps (Fig. 3, 4 and 5), in the case of the simulated robot, and in different real environments, in the case of the real robot.

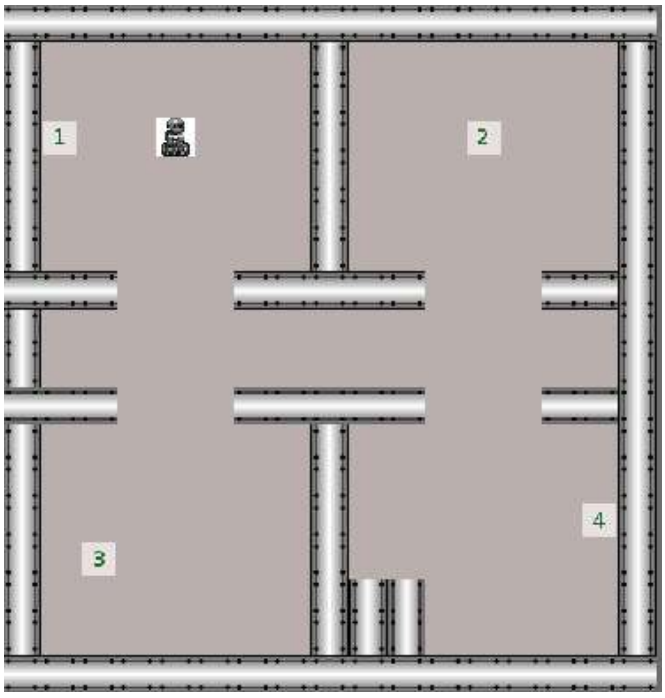

Fig. 3. Example map in 2D.

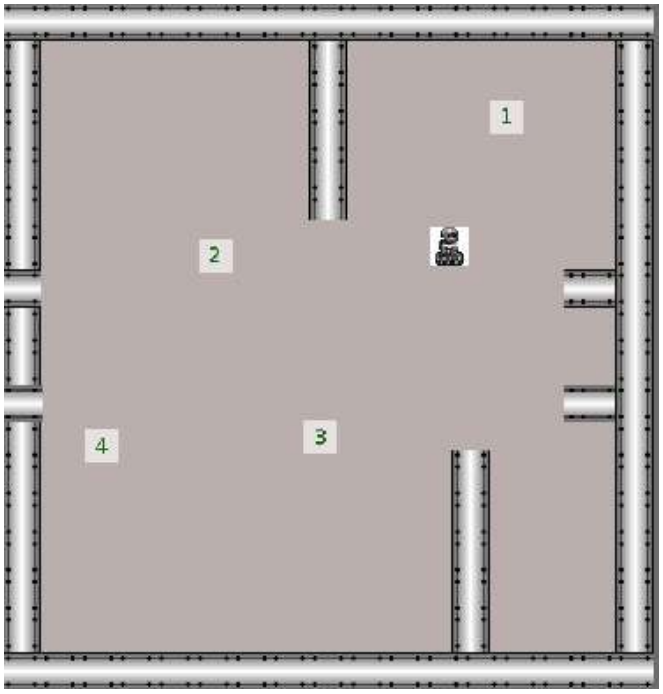

Fig. 4. Example map in 2D.

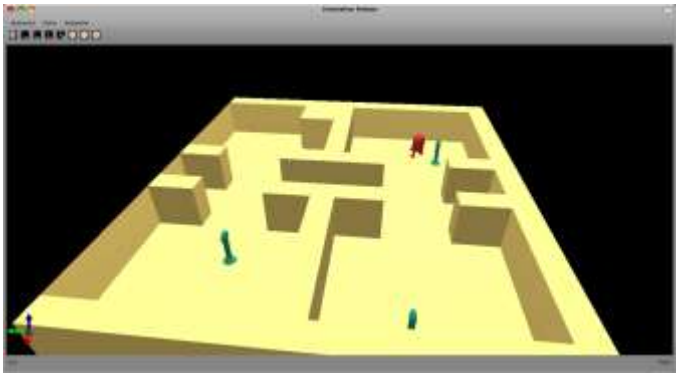

Fig. 5. Example map in 3D

IV. CONCLUSIONS

A new model to formally define virtual worlds, independently from the underlying physical layer, has been presented. Is has been used to model the control of a mobile robot, navigating in a given environment, and using a set of multimodal inputs from different types of sensors.

The model is based on a grammar which consists, on the one hand, of symbols to abstract and represent the elements of the system (primitives, actors, and so on) and, on the other hand, of a set of evolution functions so that all these elements can be combined in different ways leading to an infinite set of possible strings belonging to the grammar. By definition, each string has the ability to represent the interaction between the elements (symbols) of the system and their state at any given instant. By extension, these strings can also synthesize and formally define the system state.

As in other systems for modeling virtual worlds, the event and, in particular, the occurrence thereof, can bring about a

change in the state of a particular element and, in general, a change in the state of the system. Within the model, the event generators are responsible for managing all the possible events associated with the elements of the system.

The result of the events, namely the transition between states, involves an evolution of the original string of the system to another evolved string, which is obtained from the application of certain rules on the first string. These rules are defined within the actors, which contain the logic of how to act and deal with an event if it is activated. The main restriction to design the rules is that they should be able to translate the consequence of the events into grammar rules. The grammar rules must be applicable to the symbols of the state string and the outcome of the rules application must return a consistent string, syntactically and semantically possible.

The evolution function of the actors can be as complex as needed. In fact, this function is the vehicle to introduce intelligent behaviors in the system. This way, artificial intelligence algorithms can be introduced into the evolution function of the actor to provide it with the needed behavior.

Taking into account the diversity of virtual worlds systems available nowadays and the wide variety of devices, this model seems to be able to provide interesting features. Firstly, it is a formal model based on a grammar that allows abstracting and representing the states of the system in a general way by avoiding the specific features of other existing systems. The use of strings facilitates the parallelization and optimization of the system processes. It is also a device-independent model, therefore, is not linked to the implementation of the system with a given set of devices. It also allows the replacement of physical devices by simulated ones, and the easy addition of new ones. For instance, in the case of our robotic system, the definition string of the system is exactly the same for the simulator and for the real robot. Finally, it is a flexible model since it contemplates the possibility of reinterpreting the outputs of the actions.

In conclusion, it has been achieved the main objective of defining a new formal and generic model that is able to model general virtual worlds systems by avoiding the specific peculiarities of other models existing today.

REFERENCES

[1] Botts, M.; Percivall, G.; Reed, C. and Davidson, J.: OGC Sensor Web Enablement: Overview And High Level Architecture. OGC White Paper. Open Geospatial Consortium Inc., 2006.

[2] Davis, Martin; Sigal, Ron and Weyuker, Elaine J.: Computability, Complexity, and Languages, Fundamentals of Theoretical Computer Science, 2nd ed. San Diego: Elsevier Science, 1994.

[3] Ernst, Marc O. and Bülthoff, Heinrich H.: Merging the senses into a robust percept. TRENDS in Cognitive Sciences, vol.8, no.4, 2004.

[4] Ingrand, F.; Chatila, R. and Alami, R.: An Architecture for Dependable Autonomous Robots. IARP-IEEE RAS Workshop on Dependable Robotics, 2001.

[5] Luo, Ren; Lin, Yu-Chih; Kao, Ching-Chung: Autonomous mobile robot navigation and localization based on floor paln map information and sensory fusion approach. IEEE MFI, 2010.

[6] Posadas, J.L.; Poza, J.L., Simó, J.E.; Benet, G.; Blanes, F.: Agent-based distributed architecture for mobile robot control. Engineering Applications of Artificial Intelligence, pp. 805-823, 2008.

[7] Poza, L.; Posadas, J.; Simó, J.; Benet, G.: Arquitecturas de control jeraárquico inteligente con soporte a la calidad de servicio. XXIX Jornadas de Automática, 2008.

[8] Russell, Stuart Jonathan and Norvig, Peter: Artificial intelligence: a modern approach. Prentice Hall. ISBN: 0136042597, 2010.

[9] Sharma, R.; Pavlovic, V. I.; Huang, T. S.: Toward Multimodal Humar-Computer Interface. Proceedings of the IEEE, vol. 86(5), pp. 853-869, 1998.

[10] Singhal, A.; Brown, C.: Dynamic bayes net approach to multimodal sensor fusion. SPIE, 1997.

[11] Weser, Martin; Jockel, Sascha and Zhang, Jianwei: Fuzzy Multisensor Fusion for Autonomous Proactive Robot Perception IEEE International Conference on Fuzzy Systems (FUZZ), 2263-2267, 2008.