Formal model to integrate Multiagent Systems and Interactive Graphic Systems

Gabriel López García, Rafael Molina-Carmona and Javier Gallego-Sánchez

Grupo de Informática Industrial e Inteligencia Artificial
Universidad de Alicante, Ap.99, E-03080, Alicante, Spain
{glopez, rmolina, ajgallego}@dccia.ua.es

Abstract. A formal grammar-based model to integrate the essential characteristics of a Multiagent System using the visualization given by an Interactive Graphic Systems is presented. This model adds several advantages, such as the separation between the implementation of the system activity and the hardware devices, or the easy reusability of components. To illustrate the model, a practical case is presented. It implements a simulation of the fires in forests caused by lightning.

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 Interactive Graphic Systems (IGS) has contributed to present the information in a more friendly way using new forms of analysis. Videogames and the entertainment industry in general, have had a decisive influence on this spectacular progress, that has arrived to the field of research [10]. However, with few exceptions, solutions to integrate MASs and IGSs are not proposed.

Agents are usually considered to have some generic characteristics that make them difficult to model [3]. Although there are some common strategies for implementing them, there is a lack of a unified design pattern. As a consequence, some problems arise, such as the difficult reproduction of the results provided by the MASs [1], the lack of a suitable visual representation or the limited capabilities of interaction with the user and with the environment. MASs could take advantage of the wide research carried out in this fields to develop IGSs.

There are many generic and specific work environments to develop MASs, but they seldom include advanced visual features. For instance, in Sociology [1, 3], very specific solutions are used, generally oriented to sociological studies, such as the movement of crowds [11, 9]. In some cases, they use graphics to display statistics, the density of agents within the environment or even the animation of the movements of agents [9]. Netlog [12] is a useful tool to study highly complex systems. It has developed some graphical features, but the agents are represented by Logo style triangles just to allow the user to observe the movement agents and the population density.

Some applications of the MASs to Virtual Reality systems can also be found. One of these systems proposes an implementation of a virtual world where objects react to several gestures of the user [7]. The work of [13] uses the concepts of perception of the MASs applied to virtual reality, where the agents react interactively with the user.

In the area of the entertainment industry, some MASs applied to games can also be found. In this case, the agents are usually called bots [5] 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 [10].

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 [3]. Some examples are Repast [8] and MASON [6].

As a conclusion, there are a variety of environments that implement the essential characteristics of the MASs and, in some cases, they have some basic display functions. However, a model that unifies the definition of MASs and the whole potential of IGS has not been proposed.

This paper describes an integral model based on grammars to develop complex environments that take advantage of the MASs and the IGSs features. This system uses a descriptive language and discrete events to specify all the features necessary to define an agent. 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 incorporates a physics engine. Finally, the agents can be easily reused.

The following section provides a description of the proposed model. In section 3, 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 Model for Virtual Worlds Generation

In the proposed model, a scene is a set of dynamic and static elements. They are all represented by a sequence of primitives an transformations defined in a geometric system G. A primitive is not just understood as a draw primitive (e.g. a sphere) but an action on the geometric system that can be a visual action or not. 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 internal states. 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 executes an activity. Its generation is independent from the physical device. They provide the different ways of communication in a MAS.

Those are the main elements of the proposed system. Next, a formal mathematical model is presented, to precisely define its features.

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

Table 1. Rules of the grammar

A string $w \in \Sigma^*$ is generated by the grammar M (Table 1), if it can be obtained starting with the initial symbol WORLD and using the given rules. The language $L(M)$ is the set of all the strings which can be generated by this method, so: $L(M) = \{w \in \Sigma^* \mid \text{WORLD} \xrightarrow{*} w\}$. 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.

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 denoted using a denotational method, which defines the meaning of the string through mathematical logic terms.

Semantic Function for Primitives (Rule 6) 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 carries out 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.

Semantic Function for Transformations (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:

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

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 behaviour, encapsulating the implementation details. Therefore, the strings developed to define virtual worlds may be reused in other systems.

Semantic Function for Agents (Rule 4) The semantics of agents is a function which defines its evolution in time. 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:

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

The result u of the function may contain or not the own agent, it can generate other event for the next frame or change the state 'st' 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.

$$\eta(w, e^v) = \left\{ \begin{array}{ll} w & if \quad w \in P \\ t(\eta(v, e^v)) & if \quad w = t(v) \\ \prod\limits_{\forall f \in v} (\lambda(a^d_{st}(\eta(y, e^v)), e^f)) & if \quad w = a^d_{st}(y) \\ \eta(s, e^v) \cdot \eta(t, e^v) & if \quad w = s \cdot t \end{array} \right\} \tag{3}$$

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 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 the language $L(M)$ without agents. This function is defined as:

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

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: $\pi : L(M) \times E^V \rightarrow L(E)$

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

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. Primitives generated by agents without the visualization function represent the static part of the system.

2.1 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 [2].

2.2 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:*

$$E(C^*,t) = \left\{ \begin{array}{ll} e(z, C_i(t)) & \textit{if } z = E(C^* - C_i, t) \\ \epsilon & \textit{if } C^* = \emptyset \end{array} \right\}, \text{ where } \; e(z, e^i) = \left\{ \begin{array}{ll} z \cdot e^i & \textit{if } e^i \notin z \\ z & \textit{if } e^i \in z \end{array} \right\}$$

2.3 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:

1. $w = w_o$; $t = 0$	w_o is the intial string of the system.
2. $e^* = E(G^*, t)$	e^* are all the events generated by the system in a frame t.
3. e^v = events of e^* where $v \in V^+$	$G^* = \{$All the event generators which generate events of type $D\}$
4. $e^u = e^* - e^v$	$D = \{$All the possible events in the system$\}$
5. $w_{next} = \eta(w, e^u)$	$V = \{$All the visual events$\}$ where $V \subseteq D$
6. $v = \pi(w, e^v)$	e^v are all the events from visual devices.
7. $g = \varphi(v)$	e^u are all the events from non-visual devices.
8. $w = w_{next}$; $t = t + 1$	g is the output device.
9. If $w = \epsilon$ then go to 11	
10. Go to 2	
11. End	

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, it leads to a faster system performance.

3 Case of Study

This example is an application that simulates fires in forests caused by lightning [4]. The system basically consists of an agent who defines the forest. This agent can create other agents that implement trees (with a given probability g) and lightning (with a probability f). If a lightning is created in the same position as a tree, it will burn as well as the trees around it. To model this example, four elements are defined: events, event generators, agents and graphic primitives.

Events are used to produce the necessary activity of system. Their aim is to run certain activities of the agents that compose the scene. The events defined for this example are:

- t: Event generated to increase the time since the previous event.
- c: Creates a tree at the position (i, j) of the forest.
- f: Creates a bolt of lightning at position (i, j).
- e: Eliminates the tree of the position (i, j).
- b: Burns the tree at position (i, j).
- v: Draws using a graphics library (e.g. OpenGL).

The next step is to define the event generators:

- *Time event generator* (C_{time}): It makes the animation of the system. Every time instant t, it generates an event e^t to change the appearance of an agent.
- *Forest event generator* (C_{forest}): It produces events to create trees (e^c) and lightning (e^f).
- *Visualization event generator* ($C_{visualization}$): It captures the drawing orders and produces an event to send the elements to draw on the graphics system.

Table 2 shows the primitives and transformations that make up the scene. The functions α, β and δ define them.

Table 3 shows the evolution function λ and the graphical representation π of the agents defined for this example. The agent defined for trees (TR) has three internal states $st = \{s1, s2, s3\}$: $s1$ is the state of growth, $s2$ corresponds to a grown tree and $s3$ to a burning tree. This is an example of how an agent can have different representation depending on its internal state. The function Nbo sends burn events e^b to all the neighbours trees of an agent, creating a chain reaction (agent-agent communication). An example of agent-environment communication is made between the forest and the generator C_{forest}.

Primitive	*Description*
TR	Draw a tree
TR_b	Draw a burning tree
FA	Draw a bolt of lightning
BO	Draw a grid of NxN

Transformations	*Description*
$D_{(i,j)}$	Translate (i,j)
$S_{(s)}$	Scale (n) units

Table 2. (left) Definition of primitives, (right) Definition of transformations

Agent	*Description*	*Function λ and π*
BO	Forest	$\lambda(BO^{cfe}, e^i) = \begin{cases} TR_{s1}^{t=1} \cdot BO^{cfv} & i = c \\ FA^{t=1} \cdot BO^{cfv} & i = f \\ BO^{cfe} & i = e \\ BO^{cfe} & i \neq c, f, e \end{cases}$ $\pi(BO^v, e^i) = \begin{cases} BO & i = v \\ \epsilon & i \neq v \end{cases}$
TR	Tree	$\lambda(TR_{st}^x, e^i) = \begin{cases} TR_{s1}^{t+1} & i = t \wedge t+1 \leqslant N \wedge s = s1 \\ TR_{s2}^{b} & i = t \wedge t+1 > N \wedge s = s1 \\ TR_{s3}^{t=1} \wedge \Delta Nbo^b & i = b > N \wedge s = s2 \\ TR_{s3}^{t+1} & i = t \wedge s = s3 \\ \Delta e^e & i = t \wedge t+1 > N \wedge s = s3 \\ TR_{st}^{t} & i \neq t \wedge i \neq b \end{cases}$ $\pi(TR_{st}^v, e^i) = \begin{cases} D_{(i,j)}(S_{(n)}(TR)) & i = v \wedge st = s1 \\ D_{(i,j)}(TR) & i = v \wedge st = s2 \\ D_{(i,j)}(S_{(-n)}(TR)) & i = v \wedge st = s3 \\ \epsilon & i \neq v \end{cases}$
FA	Lightning	$\lambda(FA^t, e^i) = \begin{cases} FA^{t+1} & i = t \wedge t+1 \leqslant N \\ \Delta e^b & i = t \wedge t+1 > N \\ FA^t & i \neq t \end{cases}$ $\pi(FA^v, e^i) = \begin{cases} D_{(i,j)}(FA) & i = v \\ \epsilon & i \neq v \end{cases}$

Table 3. Agents defined for this example

4 Conclusions and Future Work

In this paper a proposal to unify the most relevant features of MASs and IGSs has been presented. The proposed model uses a context-free language to define the elements of the system. The definition of the scene, the interaction and the

graphical representation are independent from the hardware thanks to the use of event generators. They are also used for the communication (agent-agent and agent-environment). In this case, the environment is represented by the physics engine and the associated functions (e.g. collisions, gravity...).

The use of a descriptive language to define the scene and the independence from the graphics system make this model reusable just redefining the basic elements, that is, the semantic functions.

This system can be applied to a wide variety of problems so that the functions of the IGS can help to analyse the information generated by the evolution of the agents, to display the evolution in a more helpful and attractive way, to interact with the system, to reproduce the experiments or to independize the system from the physical platform.

The proposed model opens new prospects in the future such as the development of agents that are made up of other agents, the execution of actions associated to events with a given probability. So, new applications can be developed, for instance, probabilistic learning strategies or the development of genetic algorithms.

References

1. Robert Axelrod Advancing the Art of Simulation in the Social Sciences *Simulating Social Phenomena*, Springer, 1997
2. Michael R. Genesereth, Steven P. Ketchpel Software Agents *Communications of the ACM* 1995
3. Nigel Gilbert Agent-Based Models *SAGE Publications* 2008
4. John H.Miller Complex Adaptative Systems *Princeton University Press* 2007
5. Aaron Khoo, Robert Zubek Applying Inexpensive AI Techniques to Computer Games *IEEE Intelligent Systems* July-August 2002
6. Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, and Keith Sullivan MASON: A New Multi-Agent Simulation Toolkit *Vol. 81, SAGE Journals* 2005
7. Pattie Maes, Trevor Darrell, Bruce Blumberg, Alex Pentland The ALIVE System:Wireless, Full-body Interaction with Autonomous Agents, *ACM Multimedia Systems, Vol.5, No.2, pp.105-112* 1997
8. M.J. North, T.R. Howe, N.T. Collier, J.R. Vos The Repast Simphony Runtime System *Conference on Generative Social Processes, Models, and Mechanisms, Proceedings of the Agent* 2005
9. Craig Reynolds Interaction with Groups of Autonomous Characters *Game Developers Conference* 2000
10. Theresa-Marie Rhyne Computer GamesInfluence on Scientific and Information Visualization *Entertainment Computing* 2000
11. B. Ulicny, D. Thalmann Crowd simulation for interactive virtual environments and VR training systems *Computer Animation and Simulation, Springer* 2001
12. Wilensky, U., NetLogo User Manual, 1999
13. Ipke Wachsmuth, Yong Cao Interactive Graphics Design with Situated Agents *Graphics and Robotics* 1995
14. D.Martin, R.Sigal, E.J.Weyuker: *Computability, Complexity, and Languages, Fundamentals of Theoretical Computer Science*, 2nd ed, Elsevier Science 1994