FORMAL MODEL TO INTEGRATE MULTI-AGENT 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*

Keywords: Virtual Reality, Virtual Worlds generation, Formal model, Interaction.

Abstract: A formal grammar-based model is presented to integrate the essential characteristics of a Multi-Agent System with the visualization given by an Interactive Graphic Systems. 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.

1 INTRODUCTION

The growing influence of Multi-Agent Systems (MAS) in several fields of research 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. The entertainment industry have had a decisive influence on this progress (Rhyne, 2000).

Agents are usually considered to have some generic characteristics that make them difficult to model (Gilbert, 2008). Although there are some common strategies, there is a lack of a unified design pattern. Hence, some problems arise, such as the difficult reproduction of results (Axelrod, 1997), the lack of a suitable visual representation or the limited capabilities of interaction.

There are many work environments to develop MASs, but they seldom include advanced visual features. For instance, in Sociology (Axelrod, 1997; Gilbert, 2008), very specific solutions are used (Ulicny, 2001; Reynolds, 2000). Netlogo (Wilensky, 1999) is a useful tool with some basic graphics to study complex systems.

Some applications of the MASs to Virtual Reality (VR) can also be found. (Maes, 1997) proposes an implementation of a virtual world where objects react to gestures of the user. The work of (Wachsmuth, 1995) uses the concept of MASs perception applied to VR. In entertainment industry, agents are usually called bots (Khoo, 2002) and they are programmed as characters of the game. These systems are so successful that they are being used in research (Rhyne, 2000). An growing variety of generic development environments are emerging: they implement the most important features of agents (Gilbert, 2008). Repast (North, 2005) and MASON (Luke, 2005) are examples.

As a conclusion, there are a variety of environments, but a model that unifies the definition of MASs and IGSs has not been proposed. This paper describes our proposal for 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 agents. It will the interaction with the user and it is independent from the display and the interaction devices. It could also incorporate a physics engine and the agents could be easily reused.

2 PROPOSED MODEL

In our model, a scene is a set of dynamic and static elements. They are all represented by a sequence of primitives an transformations of a geometric system G. A primitive is not just a draw primitive but an action on the geometric system that can be visual or not. A transformation is a modification on the primitives

inside its scope of application.

Agents are the dynamic part and they are made of activities and a set of internal states. Each activity is executed as a reaction to an event. The agents can have a geometrical representation, defined using primitives and transformations. They also provide different ways of communication.

Formally, each element in the scene is represented by a symbol, which are used to build strings to describe the scene. The strings follow a language syntax, which is presented as a grammar (Martin, 1994) defined by the tuple $M = <\Sigma, N, R, s>$, where $\Sigma = P \cup T \cup O \cup A_{ST}^{D}$ is the set of terminal symbols (P: set of symbols for primitives, T: set of symbols for transformations, $O = \{\cdot\,()\}$: set of symbols of separation and operation, A_{ST}^{D}: set of symbols for agents with D the set of all possible types of events and ST the set of possible states), N = {WORLD, OBJECTS, OBJECT, AGENT, TRANSFORMATION, FIGURE } is the set of non-terminal symbols, s = WORLD is the initial symbol and R is the set of grammar rules defined in the following table.

1.**WORLD** → OBJECTS
2.**OBJECTS** → OBJECT | OBJECT · OBJECTS
3.**OBJECT** → FIGURE | TRANSF. | AGENT
4.**AGENT** → a_{st}^{d} (OBJECTS), $a_{st}^{d} \in A_{ST}^{D}, d \in D, st \in ST$
5.**TRANSFORMATION** → t (OBJECTS), $t \in T$
6.**FIGURE** → p^{+}, $p \in P$

A string $w \in \Sigma^{*}$ is generated by M, if it can be obtained from the initial symbol using the given rules. The language $L(M)$ is the set of all the strings which can be generated $L(M) = \{w \in \Sigma^{*} \mid \text{WORLD} \xrightarrow{*} w\}$. M is a context-free grammar, so there is a procedure to verify if a scene is correctly described.

Apart from the language syntax, it is necessary to define the semantics. It will be defined with a denotational method, through mathematical functions.

Rule 6 defines the syntax of a figure as a sequence of primitives. Primitive semantics is defined by function $\alpha : P \rightarrow G$. Each symbol in P runs a primitive on a geometric system G. So, depending on α and on the geometric system G, the result may be different. G represents actions on a specific geometric system (e.g. a graphical library such as OpenGL).

The scope of a transformation is limited by the symbols "()". Two functions are used to describe the semantics of a transformation: $\beta : T \rightarrow G$ (run when the symbol "(" is processed), and $\delta : T \rightarrow G$ (run when the symbol ")" is found). These two functions have the same features as α, but they are applied to transformations T, on 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:

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

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

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. By applying $\lambda(w, e^{f})$, $w \in L(M)$ 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:

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

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

The function λ defines the *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 at a frame. This algorithm uses the operator $\prod_{\forall f \in v}$ to concatenate strings.

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

For the visualization of an agent, it must be first converted into a string of primitives and transformations. This conversion is done by the *visualization function* $\theta : L(M) \times E^{V} \rightarrow L(E)$, where $V \subseteq D$ are events used to create different views, E^{V} are events created in the visualization process, and $L(E)$ is the language $L(M)$ without agents. It is defined as:

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

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

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

The activities are run by agents and activated by events under certain conditions. An event is defined as: *e^d_c is an event of type $d \in D$ with data e, which is carried out when the condition c is fulfilled.* The condition is omitted if $c = true$. Events may include information identifying who sent the message. So, it provides a generic communication system that can implement FIPA or KMQL (Genesereth, 1995).

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 the input devices must be defined. A new function called *event generator* is defined: *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.*

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).

The process which obtains the events produced by the generators is defined as: *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:*

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

Once all the elements involved in the model have been defined, the algorithm which carries out the entire system can be established:

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 $w = \varepsilon$ then go to 11
10. Go to 2
11. End

where w_o is the intial string, e^* are the events generated at a frame t, $G^* = \{$All the event generators$\}$, $D = \{$All the possible events$\}$, $V = \{$All the visual events$\}$; $V \subseteq D$, e^v all the visual events, e^u all the non-visual events and g 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. The algorithm finishes if the following string is empty.

3 CASE OF STUDY

This example is an application to simulate fires in forests caused by lightning (Miller, 2007). The system consists of an agent to define the forest that can create other agents: trees (with a given probability g) and lightning (with a probability f). If lightning are created in the same position as a tree, it will burn as well as the trees around it (Figure 1).

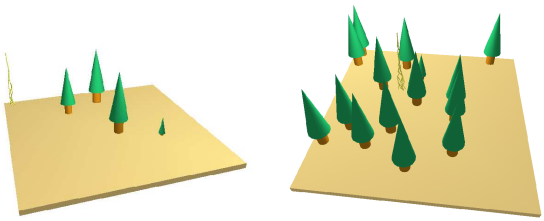

Figure 1: Examples of different simulation states

To model this example, four elements are defined: events, event generators, agents and primitives.

Events are used to produce the necessary activity of system. The events defined for this example are:

t	Event generated to increase the time.
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:

(C_{time})	Generate an event e^t at instant t
(C_{forest})	Create tree (e^c) or lightning (e^f)
(C_{draw})	Generate draw event

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

The evolution function λ and the graphical representation π are presented next. The agent defined for trees (TR) has three internal states $st = \{s1,s2,s3\}$:

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

$s1$ growth, $s2$ adult tree and $s3$ burning tree. This is an example of different representation for an agent depending on its internal state. The function Nbo sends burn events e^b to all the neighbour trees, creating a chain reaction (agent-agent communication). An example of agent-environment communication is made between the forest and the generator C_{forest}.

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

4 CONCLUSIONS

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. Although further work needs to be done, the use of a descriptive language seems to have several advantages. Firstly, the definition of the scene is reusable and independent from the platform. The use of event generators also makes the interaction with the user independent from the hardware. Event generators are also used to implement the communication, both agent-agent and agent-environment.

As future work the model will be applied to other problems to validate its features. New possibilities such as probabilistic learning strategies or genetic algorithms will be considered.

REFERENCES

R. Axelrod (1997). Advancing the Art of Simulation in the Social Sciences. *Simulating Social Phenomena*, Springer.

Michael R. Genesereth, Steven P. Ketchpel (1995). Software Agents. *Communications of the ACM*.

N. Gilbert (2008). Agent-Based Models. *SAGE Publications*.

John H. Miller (2007). Complex Adaptative Systems. *Princeton University Press*.

Aaron Khoo, Robert Zubek (2002). Applying Inexpensive AI Techniques to Computer Games. *IEEE Intelligent Systems* July-August.

Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, and Keith Sullivan (2005). MASON: A New Multi-Agent Simulation Toolkit. *Vol. 81, SAGE Journals*.

P. Maes, T. Darrell, B. Blumberg, A. Pentland (1997). The ALIVE System:Wireless, Full-body Interaction with Autonomous Agents. *ACM Multimedia Systems, Vol.5, No.2, pp.105-112*.

M.J. North, T.R. Howe, N.T. Collier, J.R. Vos (2005). The Repast Simphony Runtime System. *Generative Social Processes, Models, and Mechanisms*.

Craig Reynolds (2000). Interaction with Groups of Autonomous Characters. *Game Developers Conference*.

Theresa-Marie Rhyne (2000). Computer GamesInfluence on Scientific and Information Visualization. *Entertainment Computing*.

B. Ulicny, D. Thalmann (2001). Crowd simulation for interactive virtual environments and VR training systems. *Computer Animation and Simulation, Springer*.

Wilensky, U. (1999). NetLogo. User Manual.

Ipke Wachsmuth, Yong Cao (1995). Interactive Graphics Design with Situated Agents. *Graphics and Robotics*.

D.Martin, R.Sigal, E.J.Weyuker (1994). *Computability, Complexity, and Languages, Fundamentals of Theoretical Computer Science*, 2nd ed, Elsevier Science.