TECHNICAL ARTICLE

SIMULATION 76:1, 22-39
©2001, Simulation Councils Inc.
ISSN 0037-5497/01
Printed in the United States of America

Application of the Cell-DEVS Paradigm for Cell Spaces Modelling and Simulation

Gabriel A. Wainer
Systems and Computer Engineering Dept.
Carleton University
4456 Mackenzie Bldg.
1125 Colonel By Drive
Ottawa, ON. K1S 5B6. Canada
Gabriel.Wainer@sce.carleton.ca

Norbert Giambiasi
DIAM-IUSPIM
Université d'Aix-Marseille III
Av. Escadrille Normandie Niemen
13397 Marseilles Cédex 20 - FRANCE
Norbert.Giambiasi@iuspim.u-3mrs.fr

We present the results obtained when using the Cell-DEVS paradigm for cell spaces modelling and simulation. This formalism allows one to model and simulate cell spaces, including delay functions, to specify their timing behavior. Cell spaces can be defined in an automated fashion, simplifying the construction of new models, and easing the verification of the structural models. The approach was implemented in a development tool, showing that development times can improve by several orders of magnitude. The main results of development experiences are presented, showing the usefulness of the approach.

Keywords: Discrete-event simulation, DEVS, cellular automata, modelling methodologies, simulation development tools

1. Introduction

Formal specification mechanisms are useful in improving the security and development costs of complex systems' computer simulations. A formal conceptual model can be validated, improving the error detection process and reducing testing time. Several modeling paradigms specify dynamic systems formally. Some of them were developed to analyze Discrete Event Dynamic Systems (DEDS), that is, systems with discrete state space and piecewise constant input-output trajectories. Many of the DEDS paradigms ignore the occurrence times of the events, leading to the so-called *logical DEDS models.* These models are used to solve problems in which only the order of the events must be considered.

In recent years, the number of artificial systems that can be represented as DEDS has grown dramatically. Due to their complex characteristics, the simulation models should be able to record timing information. We need methodologies to specify *timed models*, which represent the occurrence dates of the events in the system. The DEVS formalism, proposed in [1], allows this kind of specification. In this paradigm, the model's timing is described as a lifetime for each state value.

In digital circuit modeling, some constructions define delays that can be associated with functional components [2]. This approach allows the modelers to describe complex timing behavior without knowing the simulation mechanism of the delays. The modeler description is only based on an interconnection of basic functions and different delays. The delay functions are not only useful in hardware design applications, but also to model other physical phenomena. Three kinds of delays are usually employed: *transport*, *rise-fall* and *inertial*. The behavior of transport delays allows one to reflect the straightforward propagation of signals over lines of infinite bandwidth (anticipatory semantics). Rise-fall delays allow one to model different kinds of delays for rise and fall signals, considering a finite bandwidth of the real system. Inertial delays use a preemptive semantics to represent that a quantity of energy should be provided to the system to change its state.

The present work deals with the modelling and simulation of cellular models. These systems are usually described using Cellular Automata [3], that is, n-dimensional infinite lattices of discrete-time cells. Each cell uses discrete values that change according to a local function. This is computed using the present value of the cell and a finite set of neighbors. The use of such a discrete time base poses constraints in the precision and execution performance of these models; to achieve better timing precision, the processor is overloaded.

DEVS, a discrete event paradigm, avoids these problems. It also allows modular description of the model's behavior, which is encapsulated in the model definition. Every interaction is done through input/output ports related with functions that are local to the model. The quantitative complexity is attacked using a hierarchical approach. This means that a model can be subdivided into models of lower levels of abstraction. Likewise, different models can be integrated into higher-level ones, leading to productivity improvements.

Timed Cell-DEVS [4, 5] integrates these points of view. Cellular models can be easily built, improving their execution speed and precision by using a discrete event paradigm. Their timing definition is enhanced using delay functions. Different delays can be specified, and this can be done in a simple fashion, besides the quantitative and qualitative complexity of the manipulated models.

2. Background

As stated in the previous section, the use of a modeling formalism eases the development of simulations. A formal paradigm should capture the system dynamics, allowing one to model its temporal behavior. The formal specifications should be able to be translated into an executable model. In this way, the behavior of a conceptual model can be validated against the real system, and the response of the executable model can be verified against the conceptual specification. A decomposition mechanism should be provided to reflect the hierarchical nature of most physical phenomena.

DEVS (Discrete EVent systems Specification) meets these goals [1, 6, 7]. A real system modeled with this paradigm can be described as several submodels coupled into a hierarchy. Each model can be behavioral (atomic) or structural (coupled), consisting of a time base, inputs, states, outputs and functions to compute the next states and outputs. The models can be hierarchically integrated, allowing model reuse. A DEVS atomic model is described as:

$$M = < I, X, S, Y, \delta_{int}, \delta_{ext}, \lambda, D >$$

Here, $\boldsymbol{I}$ is the model's interface, $\boldsymbol{X}$ is the input events set, **S** is the state set, and $\boldsymbol{Y}$ is the output events set. There are also several functions: $\boldsymbol{\delta_{int}}$ manages internal transitions, $\boldsymbol{\delta_{ext}}$ external transitions, λ the outputs, and **D** the elapsed time. The basic idea is that each model uses input/output ports in the interface to communicate with other models. The input external events are received in input ports, activating an external transition function. Every state has an associated lifetime, after which the internal transition function is activated. These internal events also produce state changes, whose results are transmitted through the output ports.

A DEVS coupled model is defined as:

$$CM = < I, X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}, select >$$

Here, $\boldsymbol{I}$ is the model's interface, $\boldsymbol{X}$ is the set of input events, and $\boldsymbol{Y}$ is the set of output events. **D** is an index of components, and for each $i \in D$, $\mathbf{M_i}$ is a basic DEVS model (that is, an atomic or coupled model). $\mathbf{I_i}$ is the set of influencees of model i. For each $j \in I_i$, $\mathbf{Z_{ij}}$ is the i to j translation function. Finally, **select** is a tie-breaking function defining which model executes if more than one is activated simultaneously. Therefore, a coupled model defines the components and their interaction, carried out by the translation function.

In 1948, John Von Neumann and Stephan Ulam defined another modelling formalism, suited to define spatial systems. They are called Cellular Automata, and allow the description of cell-based models by using simple rules [3]. A conceptual cellular automaton is an infinite regular n-dimensional lattice of cells, each of them taking a value from a finite set. The states in the lattice are updated according to a local rule in a simultaneous and synchronous way. The cell state changes according to a local function that uses the present cell state and a finite set of nearby cells (called the cell's neighborhood). These models can be described by:

$$CA = < S, C, N, T, \tau, c.\boldsymbol{Z}_0^+ >$$

Here, **S** is the set of discrete values of the cells, **C** defines the cell space, **N** is the neighborhood function,

T is a global transition function, τ is the local computing function, and c.$\mathbf{Z}_0^+$ is the discrete time base. The cell space is composed of individual cells ($\mathbf{c_{ij}}$) activated in discrete time steps. The evolution of the automaton is defined by the execution of the global transition function, defined by the state of every cell in the space. This behavior depends on the results of the local computing functions, which executes locally in each cell's neighborhood.

The use of a discrete time base poses restrictions in the precision and efficiency of the simulated models. If complex cellular automata are considered, higher precision can only be achieved by reducing the activation period for each time step. Therefore, large amounts of computing time will be wasted to obtain the desired results. Furthermore, in several cases, most cells of the automaton do not need to be updated in each time step. These "quiescent" states allowed one to define modifications in which the automaton advances using instantaneous events that can occur at unpredictable times. One of these extensions, presented in [1, 6], uses DEVS for cellular modelling. The basic idea is to consider a cell space as a discrete event model where each cell is defined as an atomic model. These ideas were applied in real applications in later works [8, 9]. We have introduced extensions to allow explicit timing for each of the cells in the space, providing a simple mechanism to define them.

2.1 *Timed Cell-DEVS Atomic Models*

Timed Cell-DEVS defines each cell as an atomic model. Inertial or Transport delays allow one to define the timing behavior of each cell. A Timed Cell-DEVS atomic model can be described as:

$$TDC = < I, X, Y, \theta, N, delay, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D >$$

Here, $\mathbf{X}$ defines the external inputs, $\mathbf{Y}$ the external outputs, and $\mathbf{I}$ is the interface of the model. θ is the cell state definition, and **N** is a set of inputs for the cell. **Delay** defines the kind of delay used, and **d** its duration. Finally, there are several functions: τ for local computations, and $\delta_{\mathbf{int}}$, $\delta_{\mathbf{ext}}$, λ and **D**, with the same semantics as in other DEVS. Each cell has a well-defined interface composed of a fixed number of ports, and it is connected with a neighbor using these ports. If a cell is connected to other models outside the cell space, a set of other input/output ports is included. The cell's state is composed by its present value, a queue to keep track of the next event times and their associated input values, and the cell's phase (active or passive). The **N** set includes the values that are used to compute the future state using the Boolean function τ. The delay allows one to defer the transmission of the results (this behavior is defined by the $\delta_{\mathbf{int}}$, $\delta_{\mathbf{ext}}$, λ and **D** functions). Therefore, a modeler must only focus on defining the local computing function, the kind of delay and its length.

Whenever an event arrives, the external transition function is activated. In this case, it takes the set of inputs and computes the cell's future state using the τ function. If the new cell state is different from the previous one, its value should be sent to the neighbors (that is, the cell's influencees). Otherwise, the cell is quiescent and its neighbors must not be activated [1, 6]. In any case, the state changes are transmitted only after the consumption of the delay.

To provide transport delays, the external transition function schedules an internal event after the time defined by the delay. If a state change is detected and the cell is passive, it is activated. Instead, if the cell is active, the values of elapsed times since the last event must be updated. When the delay has expired, the state change should be transmitted to the influencees. Therefore, the output function is activated, sending the first value stored in the queue. Then the internal transition function removes this element and it schedules a new internal event. If the queue is not empty, the time associated with the first element in the queue is used. Otherwise, the cell becomes passive. An example of this behavior is presented below. The cell shown in Figure 1 (previously presented in [10]) represents an ideal Boolean function that transmits its inputs.

Let us consider that the cell uses a transport delay of 17 time units. In this case, the cell's interface is defined by $I = <1, P^x, P^y>$, where $P^x = \{X1\}$, $P^y = \{Y1\}$. Then, $X = Y = \{0, 1\}$, d = 17, and τ(s) = s. The functions $\delta_{\mathbf{int}}$, $\delta_{\mathbf{ext}}$ and λ will behave as previously defined.

Let us suppose that this model is activated with the input trajectories depicted in Figure 2. Table 1 shows the cell's behavior for those trajectories, and they are also depicted in the previous figure. The table shows each transition, its activation time and the cell's state values. The * sign identifies the execution of internal transition functions. The remaining lines represent the execution of external transitions (the fields containing two values separated by a slash represent the variable values before and after execution). The *s* and *s′* columns represent the present state value and the new computed value. The cell *phase* can be active/passive. The variable σ represents the remaining time up to the next scheduled event, and *e*, the elapsed time since the last one. Finally, *σqueue* contains the future values to be transmitted and their scheduled times.

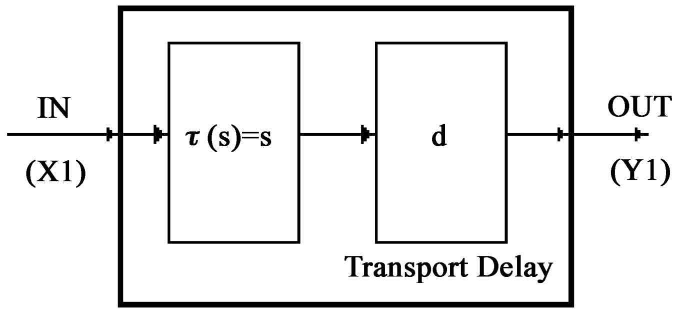


Figure 1. Boolean function with transport delay

Table 1. Execution sequence of a transport delay cellular model

	t	s	s'	phase	σ	e	σqueue
	...	0	0	passive			
	30	0/1	1	active	17	0	(1,17)
	40	1/0	0	active	7	10	(1,7),(0,17)
*	47	0	0	active	0 / 10	17 / 0	(0,10)
	55	0/1	1	active	2	8	(0,2), (1,17)
*	57	1	1	active	0 / 15	10 / 0	(1,15)
	60	1/0	1/0	active	12	3	(1,12), (0,17)
*	72	0	0	active	0 / 5	15	(0,5)
*	77	0	0	passive	∞	5	

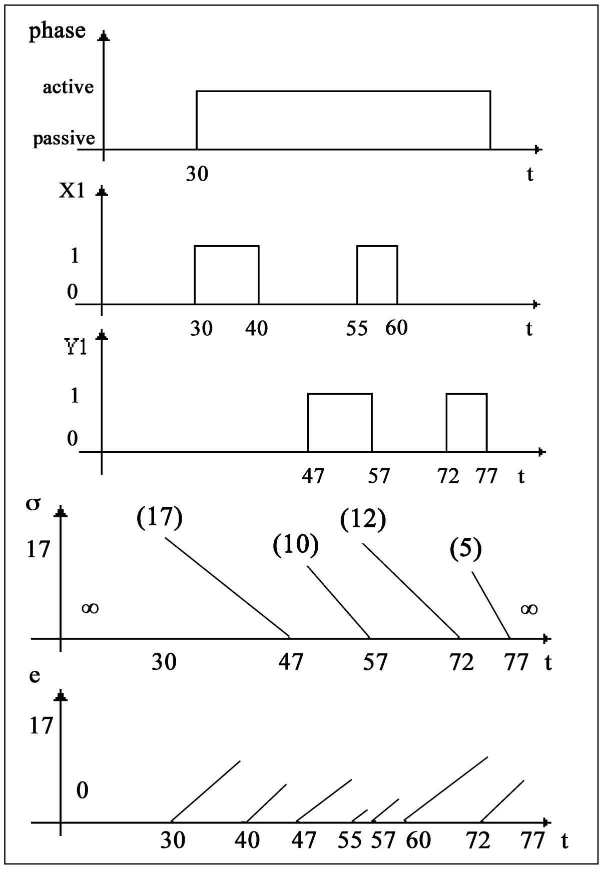


Figure 2. Input and output trajectories with transport delay

We can see that in the instant 0 the cell is in passive state. In simulated time 30, an external event arrives, producing a state change. The cell is now active, and an internal event is scheduled 17 time units from the present instant. In 40, another external event occurs, producing a new state change. In this instant, the σqueue contains two scheduled outputs: one that is 17 time units from the present instant (recording the last change), and the first one 7 time units away. When this time is consumed, the internal transition is executed. The first value in the queue is transmitted, and its entry erased. The first value in the queue is used to schedule the next internal transition, which will occur in the simulated time 57. When there are no future queued events, the cell passivates.

Inertial delays introduce preemptive semantics: an input must be discarded if its value is not kept for a certain period. Instead, if the input flow is steady during that time, the state change is transmitted. The behavior for atomic cells with inertial delays is presented in the following example. Figure 4 and Table 2

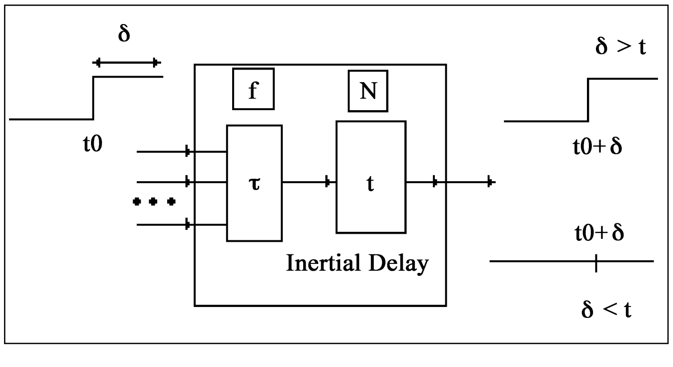


Figure 3. Behavior of an atomic cell model with inertial delay

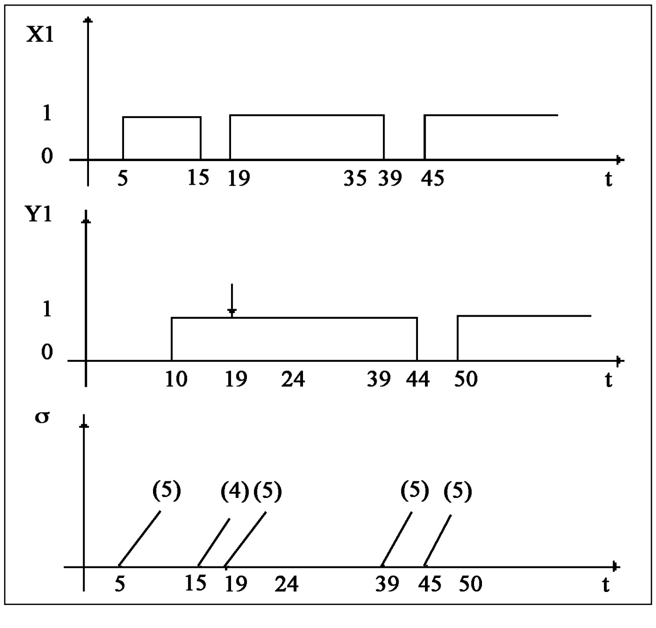


Figure 4. Input and output trajectories with inertial delays

Table 2. Execution sequence of Figure 4

	t	s	s'	phase	σ	e	x	f
	...	0	0	passive	∞			
	5	0/1	1	active	5	0	1	1
*	10	1	1	passive	0/∞	5		
	15	1	0	active	5	0	0	0
!	19	1	1	active	1/5	4/0	1	0/1
*	24	1	1	passive	0/∞			
	39	1/0	0	active	5	0	0	1/0
*	44	0	0	passive	0/∞			
	45	0	0	active	5	0	1	0/1
*	50	1	1	passive	0/∞			

present input/output trajectories based on the same atomic cell used previously. An inertial delay of 5 time units is used. The table representation is the same previously used, and the lines marked with "!" signs (arrows in the figure) represent the model behavior under preemption.

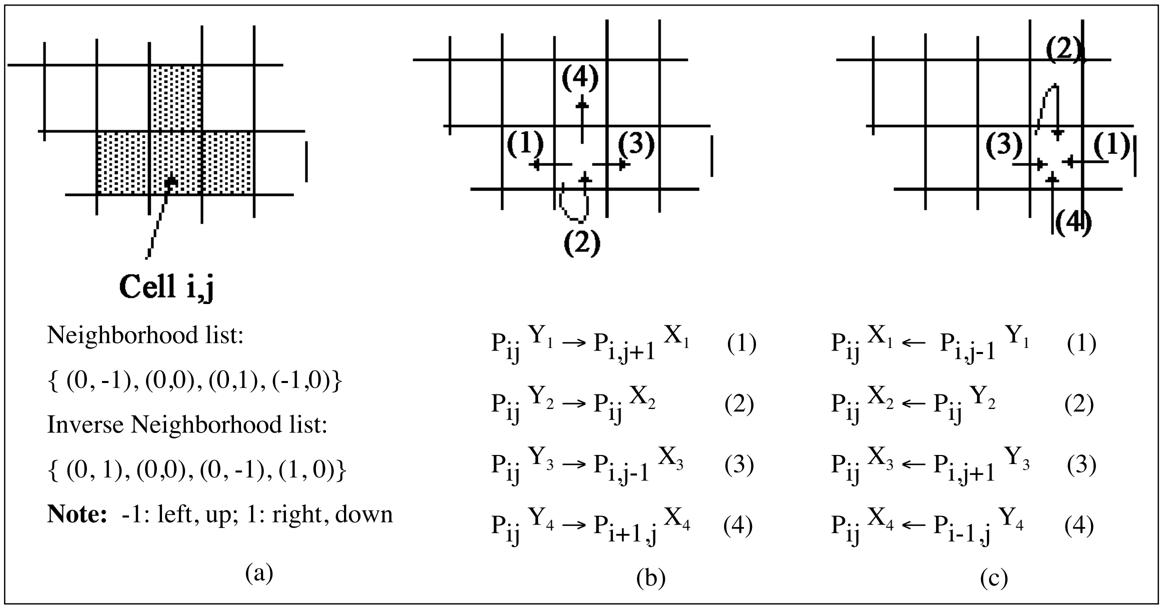
Initially, the cell is in passive state. When an external event arrives, the local function is activated. The cell executes, its state changes, and it schedules an internal event 5 time units from the present. In the simulated time 15, an external event arrives. The present state for the cell is 1, and it is changed to 0. The feasible future state for the cell (f) is now 0. The inertial delay is not accomplished, because 4 time units later another state change occurs. Then, the feasible future state is preempted and the state is kept in 1. The state change is not explicitly transmitted because the system state at that moment was 1 (the previous external event was preempted).

2.2 Timed Cell-DEVS Coupled Models

Once the definition of each cell has been completed, coupled models must be defined. These models allow us to build complex models consisting of several submodels with different behavior using different abstraction levels. A Cell-DEVS coupled model is defined by:

$$GCC = < X_{list}, Y_{list}, I, X, Y, n, \{t_1,...,t_n\}, N, C, B, Z >$$

Here, $\mathbf{X}_{list}$ and $\mathbf{Y}_{list}$ are input/output coupling lists, used by $\boldsymbol{I}$ to define the model interface. $\boldsymbol{X}$ and $\boldsymbol{Y}$ represent the input/output event sets. The **n** value defines the dimension of the cell space, $\{\mathbf{t_1,...,t_n}\}$ is the number of cells in each dimension and **N** is the neighborhood shape. C, together with B, the set of border cells, and Z the translation function define the cells in the space. For coupled models, the modeler only has to define the neighborhood shape, size and dimension of the



Neighborhood list:

{ (0,-1), (0,0), (0,1), (-1,0)}

Inverse Neighborhood list:

{ (0, 1), (0,0), (0, -1), (1, 0)}

Note: -1: left, up; 1: right, down

$P_{ij}{}^{Y_1} \rightarrow P_{i,j+1}{}^{X_1}$ (1)

$P_{ij}{}^{Y_2} \rightarrow P_{ij}{}^{X_2}$ (2)

$P_{ij}{}^{Y_3} \rightarrow P_{i,j-1}{}^{X_3}$ (3)

$P_{ij}{}^{Y_4} \rightarrow P_{i+1,j}{}^{X_4}$ (4)

$P_{ij}{}^{X_1} \leftarrow P_{i,j-1}{}^{Y_1}$ (1)

$P_{ij}{}^{X_2} \leftarrow P_{ij}{}^{Y_2}$ (2)

$P_{ij}{}^{X_3} \leftarrow P_{i,j+1}{}^{Y_3}$ (3)

$P_{ij}{}^{X_4} \leftarrow P_{i-1,j}{}^{Y_4}$ (4)

Figure 5(a). A cell, its neighborhood and the neighbor's list; **5(b)**. Connection of the output ports of cell i,j (using the neighborhood list); **5(c)**. Connection of the input ports of cell i,j (using the inverse neighborhood list) model, definition of the borders, and the coupling lists.

The B set defines the cell's space border. If the set is empty, every cell in the space has the same behavior. The space is "wrapped," meaning that cells in one border are connected with those in the opposite. Otherwise, the border cells will have different behaviors than those of the rest of the model. The interface ***I*** is built using X_{list} and Y_{list}, two lists defining the model's inputs/outputs. Finally, the Z function allows one to define the coupling of cells in the model. This function translates the outputs of m-eth output port in cell C_{ij} into values for the m-eth input port of cell C_{kl}. Each output port will correspond to one neighbor and each input port will be associated with one cell in the inverse neighborhood [1]. The ports' names are generated using the following notation: P_{ij}^{Xq} refers to the q-eth input port of cell C_{ij}, and P_{ij}^{Yq} to the q-eth output port. These ports correspond with the port names denoted as X_q or Y_q for each cell.

The definition of DEVS coupled models has been changed to include Cell-DEVS:

$$CM = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}, \text{select} >$$

where all the sets have the same meaning than the defined previously, and $\forall\ i \in D$, $\mathbf{M_i}$ is a basic DEVS model:

$$M_i = GCC_i = < I_i, X_i, Y_i, X_{list\,i}, Y_{list\,i}, n_i, \{t_1,...,t_n\}_i, N_i, C_i, B_i, Z_i >$$

if the coupled model is Cell-DEVS, and $M_i = < I_i, X_i, S_i, Y_i, \delta_{int\,i}, \delta_{ext\,i}, I_i >$ otherwise.

$\mathbf{Z_{ij}}$ is the i to j translation function, where:

$Z_{ij}: Y_i \rightarrow X_j$ if none of the models involved are Cell-DEVS, or

$Z_{ij}: Y(f, g)_i \rightarrow X(k, l)_j$, with $(f, g) \in Y_{list\,i}$, and $(k, l) \in X_{list\,j}$ if either of the models i or j is a Cell-DEVS.

3. An Application Example

The previous section presented a set of specifications for DEVS and Cell-DEVS models. These specifications allowed us to define formal verification mechanisms, and to prove a model's properties. However, they also make easy the definition of a cellular model. These models can be defined using very simple rules and a few parameters. Complex timing definition is overruled due to the use of delay functions. As a result, a very simple set of procedures is needed to define complex models, thus improving the development process. This is shown in this section, which includes a detailed example of application of the paradigm.

The model depicted in Figure 6 is devoted to simulate a section of urban population. It is composed of

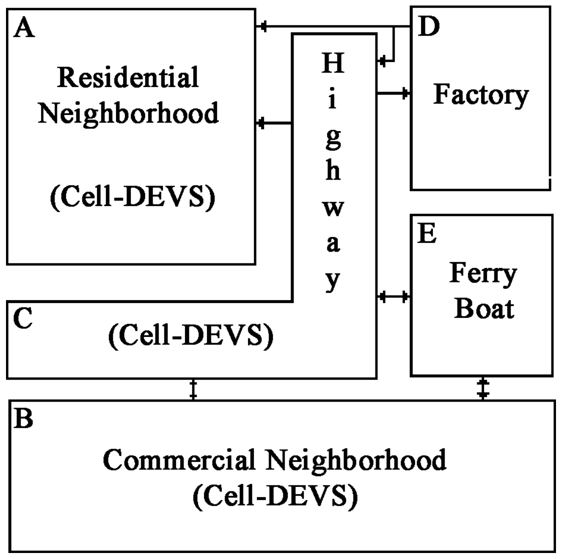


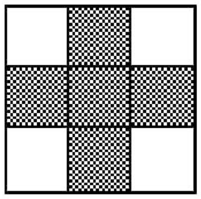
Figure 6. Coupling of Cell-DEVS and other DEVS models

different submodels built using different approaches. As the goal of the example is to show the application of the paradigm, the model's behavior has been simplified.

The submodel A represents pollution in a residential neighborhood. This model is built using a Cell-DEVS representing particles of smog. Influences of the traffic on a nearby highway and the smoke of trucks in a factory are considered to generate pollution. The local computing function will model the influence of the wind. Any vehicle is considered to generate enough smog to activate a cell in this model. Model B represents the traffic movement in a commercial neighborhood. The streets are one-way (northbound or eastbound) and no traffic lights are modeled. This Cell-DEVS model could be used to study the traffic flow to install traffic lights. The Cell-DEVS model C represents a one-way highway passing across neighbors A and B. Traffic flow on the highway will be represented using a cellular model for one-way routes. The model could serve to study the traffic density on the highway and its influence on the rest of the city. Atomic model D represents a factory, with trucks arriving from/to the highway. The simulation results could be used to schedule the input and output of trucks to the factory, improving product delivery. Finally, atomic model E represents a ferryboat that connects the city with an island. This model could be used to determine the optimal number of boats to be used depending on the traffic flow.

3.1 Cells Behavior Specification

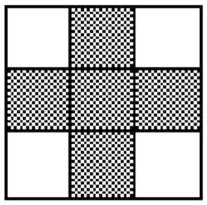
This section analyzes the definition of the cell behavior of the example. As the D and E models are traditional DEVS, their definitions are not included. In both cases, the models are constructed as queuing servers simulating the arrival and departure of vehicles (to see the details about these models, see [11]). The local computing functions are described using a specification



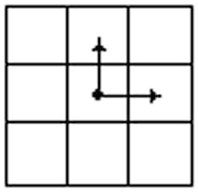
Neighborhood: { (a_{12}, a_{21}, a_{22}, a_{23}, a_{32}) };
wind = 3;
m = 9; n = 10; in_delay(wind).

Result	Input values
1	(a_{22} = 1 AND NOT (ALL = 0)) OR ((a_{22} = 0) AND (a_{12} = 1 OR a_{21} = 1 OR a_{23} = 1 OR a_{32} = 1))
0	(ALL = 0) AND (a_{22} = 1)

Figure 7. Specification for model A



Neighborhood: { (a_{12}, a_{21}, a_{22} ,a_{23}, a_{32}) };
speed = 5;
m = 50; n = 10;
tr_delay(speed).



Result	Input values
1	(a_{22} = 0 AND a_{32} = 1) /* Northbound */ OR (a_{22} = 0 AND a_{21} = 1) /* Westbound */ OR (a_{22} = 1 AND a_{23} = 1) OR (a_{22} = 1 AND a_{32} = 1)
0	(a_{22} = 1 AND a_{12} = 0) /* Northbound */ OR (a_{22} = 1 AND a_{23} = 0) /* Westbound */

Figure 8. Specification for model B: **8(a)**. Cell space parameters; **8(b)**. Car movement description

language defined in [11]. Here, the a_{ij} variables define the inputs of a cell, related with a 9×9 matrix representing the cell neighborhood. These values are used as preconditions for the local computing function. The result obtained is the new state for the cell.

3.1.1 Model A (Smog in the Residential Neighborhood)

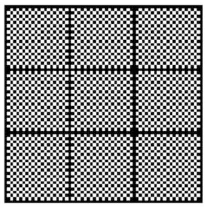
An inertial delay has been used to model the pollution diffusion. If the wind removes the smog faster than its diffusion, it will not be expanded. If there is an input with an inertial delay of 3 time units, the particles remain in the cell. If there is a particle of smog in a neighbor cell, and the particle sustains for 3 time units, the particle expands to the present cell. Otherwise, the influence of the wind is considered, and the preemption leaves the cell unchanged. Therefore, an input event will influence the cell only if nothing happens before 3 time units.

3.1.2 Model B (Traffic in the Commercial Neighborhood)

The specification of model B represents the movement of cars on one-way streets of the commercial neighborhood. The first rule of the specification represents the input of a new car into an empty cell, or a car that cannot move on that cell. The second rule represents the car's movement (departure of a car from the origin cell), as it can be seen in Figure 8(b).

The movement of a car is not explicitly registered in one rule, but in two separated phases. The state change expands to all the neighbors, but is registered in those where the movement is valid. The first movement removes the vehicle from the origin cell. The neighbors will detect this state change. When the cell corresponding to the arrival activates, it records the arrival of the new car. The remaining neighbors are not changed.

If there is a crossing, the car on the right passes because northbound flow is evaluated first. The transport delays allow one to model the acceleration delay of the cars in a very simple fashion. The car movement is delayed before the next movement to the following cell, allowing one to model different speeds. The delay can be represented as a random number to model the different speeds of each car.



Neighborhood: { ($a_{11,}$ a_{12}, ..., a_{33}) };
speedH = 2;
m = 15; n = 4;
tr_delay(speed).

Result	Input values
1	($a_{22} = 0$ AND $a_{21} = 1$) /* Normal flow */ OR ($a_{22} = 0$ AND $a_{21} = 0$ AND $a_{31} = 1$ AND $a_{32} = 1$) /* Pass through the left */ OR ($a_{22} = 0$ AND $a_{21} = 0$ AND $a_{11} = 1$ AND $a_{12} = 1$) /* Pass through the right */ OR ($a_{22} = 1$ AND Column$_3 = 1$) /* Jam */
0	($a_{22} = 1$ AND $a_{23} = 0$) /* Normal flow */ OR ($a_{22} = 1$ AND $a_{23} = 1$ AND $a_{13} = 0$ AND $a_{12} = 0$) /* Pass through the left */ OR ($a_{22} = 1$ AND $a_{23} = 1$ AND $a_{33} = 0$ AND $a_{13} = 1$ AND $a_{32} = 0$) /* Pass through the right */ OR ($a_{22} = 0$) /* Empty cell not considered in rule 1 */

Figure 9. Specification for model C.

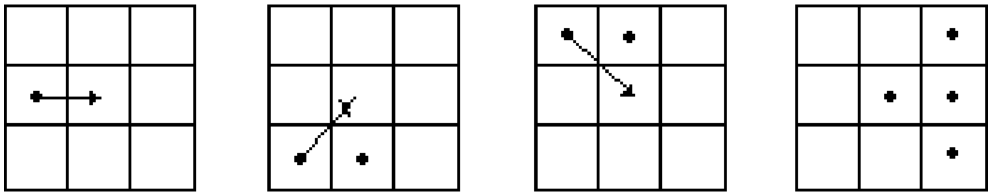


Figure 10. Valid movements for rule 1 (different expressions)

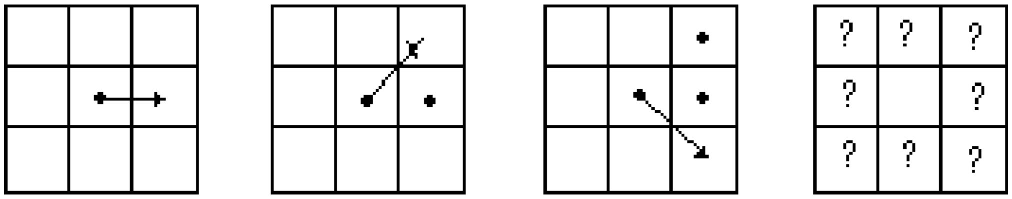


Figure 11. Valid movements for rule 2 (different expressions)

3.1.3 Model C (Highway)

This specification represents a four-lane, one-way highway. In this case, the car movement is also divided into two movements. The first rule included represents the arrival of a new car to the present cell, or a car stopped. Each of the expressions in this rule represents either of the movements shown in Figure 10.

The second rule reflects the movements shown in Figure 11.

3.2 *Formal Specification of the Models*

In this section, the Cell-DEVS models presented in the previous section will be formally specified. Let us now consider first the complete definition of the cell space A. A detailed sketch of this model can be seen in Figure 12.

A formal specification for this cell space is:

$$M_A = < I_A, X_A, Y_A, X_{listA}, Y_{listA}, \eta_A, N_A, \{m_A, n_A\}, C_A, B_A, Z_A >$$

First, we show the parameters that must be introduced by a modeler:

$\mathbf{X}_{\text{list}\mathbf{A}}$ = { (1,10); (2,10) }; $\mathbf{Y}_{\text{list}\mathbf{A}}$ = { ∅ }.
$\mathbf{N_A}$ = { (0,0), (–1,0), (1, 0), (0,1), (0,–1) };
$\mathbf{m_A}$ = 9; $\mathbf{n_A}$ = 10; $\mathbf{B_A}$ = { ∅ };

The remaining parameters are automatically built as follows:

$\eta_\mathbf{A} = 5$;
$\mathbf{I_A} = < P^x, P^y >$, where $P^x = \{ <X(1,10), \text{binary}>, < (2,10), \text{binary}> \}$; $P^y = \{\varnothing\}$;
$\boldsymbol{X}_\mathbf{A} = \boldsymbol{Y}_\mathbf{A} = \{0, 1\}$;

$P_{ij}^{Y_1} \to P_{i,j-1}^{X_1}$	$P_{i,j+1}^{Y_1} \to P_{ij}^{X_1}$
$P_{ij}^{Y_2} \to P_{i+1,j}^{X_2}$	$P_{i-1,j}^{Y_2} \to P_{ij}^{X_2}$
$P_{ij}^{Y_3} \to P_{i,j+1}^{X_3}$	$P_{i,j-1}^{Y_3} \to P_{ij}^{X_3}$
$P_{ij}^{Y_4} \to P_{i-1,j}^{X_4}$	$P_{i+1,j}^{Y_4} \to P_{ij}^{X_4}$
$P_{ij}^{Y_5} \to P_{ij}^{X_5}$	$P_{ij}^{Y_5} \to P_{ij}^{X_5}$

Figure 13. Port interconnection of ZA function

$\mathbf{C_A}$ is the cell space set, defined as $C_A = \{ C_{A\,ij} \,/\, i \in [1, 9], j \in [1, 10]\}$, where each $C_{A\,ij}$ is a Cell-DEVS component.

$\mathbf{Z_A}$ is defined as previously explained. In this case, if we consider the Neighborhood list N and the inverse neighborhood, the port interconnection for the $Z_\mathbf{A}$ function will be defined as shown in Figure 13.

Each cell of the space can be formally described as:

$$C_{A\,ij} = < I_A, X_A, S_A, Y_A, N_A, \delta_{int\mathrm{A}}, \delta_{ext\mathrm{A}}, \mathrm{Delay}_A, d_A, \tau_A, \lambda_A, D_A >$$

In this case, we also mention the values defined by the modeler:

$\mathbf{t_A}$, that was defined in the previous section using the specification language.

$\mathbf{Delay_A}$ = transport; $\mathbf{d_A}$ = wind;

The remaining components are defined automatically. The description of X_A, Y_A and N_A are inherited from the cell space. Besides:

$\mathbf{I_A} = < 5, P^x, P^y >$, where

$P^x = \{ (X_1, \text{binary}), (X_2, \text{binary}), (X_3, \text{binary}), (X_4, \text{binary}), (X_5, \text{binary}) \}$;

$P^y = \{ (Y_1, \text{binary}), (Y_2, \text{binary}), (Y_3, \text{binary}), (Y_4, \text{binary}), (Y_5, \text{binary}) \}$.

$\mathbf{S_A}$:

Descriptive variables:

$$s = \begin{cases} 1 \text{ if there is smog in the cell;} \\ 0 \text{ otherwise} \end{cases}$$

$\lambda_\mathbf{A}$, $\delta_{int\mathbf{A}}$ and $\delta_{ext\mathbf{A}}$ behaves as the functions defined earlier.

The remaining cellular models are defined in a similar fashion, detailed in [4]. Let now us consider the formal specification for the higher level coupled model. In this case:

$$M = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}, \text{select} >$$

$\boldsymbol{X} = \boldsymbol{Y} = \{\varnothing\}$;

$\mathbf{D} = \{ A, B, C, D, E \}$, and $\forall\, i \in D$, M_i is one of the basic DEVS models previously defined.

$\mathbf{I_i}$ is the set of influencees of model i. In this case,

$I_A = \{\varnothing\}$;	$I_B = \{ C, E \}$;
$I_C = \{ A, B, D, E \}$;	$I_D = \{ A, C \}$; and
$I_E = \{ B, C \}$.	

$\forall\, j \in I_i$, $\mathbf{Z_{ij}}: Y_i \to X_j$ if i and j are DEVS models, or $\mathbf{Z_{ij}}: Y(f,g)_i \to X(k,l)_j$, with $(f,g) \in Y_{list\,i}$, and $(k,l) \in X_{list\,j}$ if either model is Cell-DEVS. In this case, the $\mathbf{Z_{ij}}$ function is defined as:

$Z_{BC}: Y(1,3)_B \to X(4,5)_C$	$Z_{BE}: Y(1,9)_B \to IN_f$
$Z_{CA}: Y(1,10)_C \to X(2,10)_A$	$Z_{CB}: Y(4,4)_C \to X(1,2)_B$
$Z_{CD}: Y(4,15)_C \to IN_f$	$Z_{CE}: Y(4,1)_C \to IN_b$
$Z_{DA}: OUT_f \to X(1,10)_A$	$Z_{DC}: OUT_f \to X(4,14)_C$
$Z_{EB}: OUT_b \to X(1,7)_B$	$Z_{EC}: OUT_b \to X(3,4)_C$

Finally, **select** = { C, A, B, D, E}.

4. Implementation Models for DEVS-Cells

Cell-DEVS modelling is independent of the simulation technique used. This assertion was confirmed by implementing two different simulators (and defining a parallel version of both, which is still not implemented). In one of them, simulation advances as in other DEVS-based environments. Special simulation processors (coordinators and simulators) drive the process [6]. Coordinators and simulators are associated with coupled and atomic models respectively. These processors drive the simulation by sharing information through message passing. There are four different messages: **-messages, X-messages, Y-messages* and *done-messages*.

The submodel with the smaller scheduled internal event in the hierarchy is called the *imminent*. When this model must execute, a *-message is sent to its simulator. The message passes through the middle-level coordinators, provided with a list of imminent children with this purpose. When an external message arrives, an X-message is consumed and the external transition function executed. The simulators return done-messages and Y-messages that are converted to new *-messages and X-messages, respectively. Y-messages carry the results to be transmitted to other models. Done-messages are used to inform that each model has finished with the task given by its higher level coordinator.

When a coupled Cell-DEVS is defined, the set of processors associated with the cell space is automatically created. As the coupled model consists of several atomic ones (the cells), one simulator is associated with each cell. The simulators are created using the

```
Data structures:

Cells = { binary state; float delay; } [m x n];
Neighbors = { (int i, int j), i = –1..1; j = –1..1};       /* Relative cell's positions */
Next-Event List = { Cell_position (int i, int j), float time};
binary New-State;
float tn, tl, e; /* time of next event, last event and elapsed time */
Init() {
  Load the initial state and delays for each cell in the array;
  Detect non-quiescent states;
  For each non-quiescent cell do
          Add {Cell_position(i,j), delay} to the Next-Event list;
  endfor
}
Reaction_to_X-message() {
          tl = tn;
          Take the first item in the Next-Event list;
          tn = time of the first item;
          e = 0;

          New-State = τ(N[i,j]);     /* Compute the local function for the cell
                                                         Receiving the input message */
          If (New-State != State) then
               Add {Cell_position(i,j), Cells[Cell_position].state} in the Next-Event list;
                                  /* Avoiding repeated elements */
          endif

           If (INERTIAL-DELAY) then
                    If (there is an element for this cell whose Sigma is greater than the new one) then
                              Delete the element;        /* Preemption */
                    endif
           endif
           Send a done message to the upper level coordinator (including the cell and time);
}

Reaction_to_*-message () {
          tl = tn;
          Take the first item in the Next-Event list;
          tn = time of the item;
          e = 0;

          /* Take the imminent cells. Execute their internal transition functions */
          For each cell (i,j) whose message in the Next-Event list has time = tn do
                Update the cell state (if necessary);

               If (the cell is in Ylist) then              /* External coupling */
                    Create an Y-message (including the next-event time) and send it to the father coordinator;
              endif

            /* Common procedures for inertial or transport delays */
            For each pair (k, l) in the Neighbors list do
                    Next_event_time = tn + Cells(i+k, j+l).delay;
                    /* Internal coupling */
                    Add { (i+k, j+l), Next_event_time} in the Next-Event list;
             endfor

            e = tn – tl;
          endfor
}
```

Figure 15. Flat simulation for Cell-DEVS spaces parameters defined in the formal specification of the previous sections. Then, the methods explained earlier are used to define the internal coupling of the cell space. A coordinator is then associated with each cell coupled model.

The coordinators now behave differently when detecting the message destinations, since they are composed by a cell and a model name. The model name is used by the high-level coordinators and the cell position is employed by the cell space coordinator. For Cell-DEVS spaces, the imminent model will be a cell in the space. The simulator linked with that cell activates the model's output and internal transition functions. When an external message arrives, an X-message is consumed and the external transition function executed. The local computing function is activated, and its output is delayed. The simulators return done-messages and Y-messages that are converted to new *-messages and X-messages, respectively. These messages are translated using the coupling mechanisms previously defined.

To allow the integration between Cell-DEVS models and other DEVS models, the Coordinator class should include the model coupling mechanism defined by the formalism. The output ports in the interface of the X_{list} are connected with the input ports of those models in the Y_{list}.

A second simulation mechanism was defined for Cell-DEVS models. We are using a hierarchical module definition, and the simulation mechanism also has hierarchical interaction. This intermodule communication produces a high degree of overhead, which can reduce the execution performance of simulations consisting of a large number of cells. A flat coordinator was introduced with this goal. It must be initialized by detecting quiescent states for the initial state of the cell space. Non-quiescent cells (i.e., those whose state can change) will be added in a Next-Events list. New arriving X-messages are also inserted in this list. When the coordinator receives a *-message, it removes the first element of the Next-Events list, and the local computing function for the cell is activated. If the cell state has changed, its value is returned and the next event time for the cell is computed. The coordinator will create a Y-message containing the cell state value, and will transmit it to the upper level coordinator. Finally, a new message is created and added in the Next-Events list for each cell in the neighborhood. If one simulation cycle finishes, a done-message is created. The detailed behavior of this simulator is represented as shown in Figure 15.

Below, we will use the example defined in Section 3 to analyze the behavior of the cell spaces simulators. First, the hierarchical simulation mechanisms are exemplified. Then, the example attacks the flat simulation procedures.

4.1 Hierarchical Simulation of the Example

Figure 16 depicts a hierarchy of simulation processors considered for the present example. Let us now suppose that in simulated time 10 the model's contents are those of Figure 17. There is only one active cell in the model: cell (1,9) of model C.

At present there is only one message waiting to be processed in the Event-List of the root coordinator. Therefore, the contents of the data structures of the simulators and coordinators are those of Figure 18.

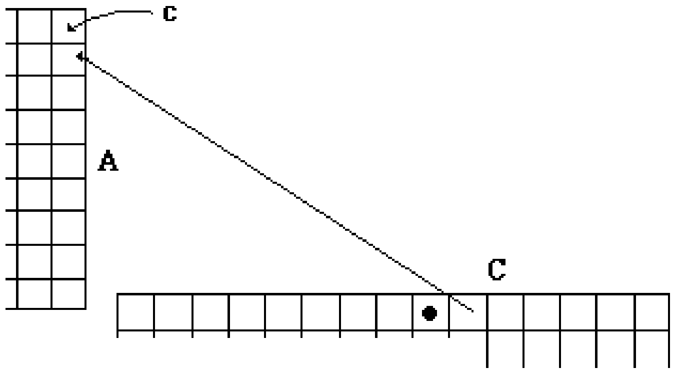


Figure 17. Model M's initial state

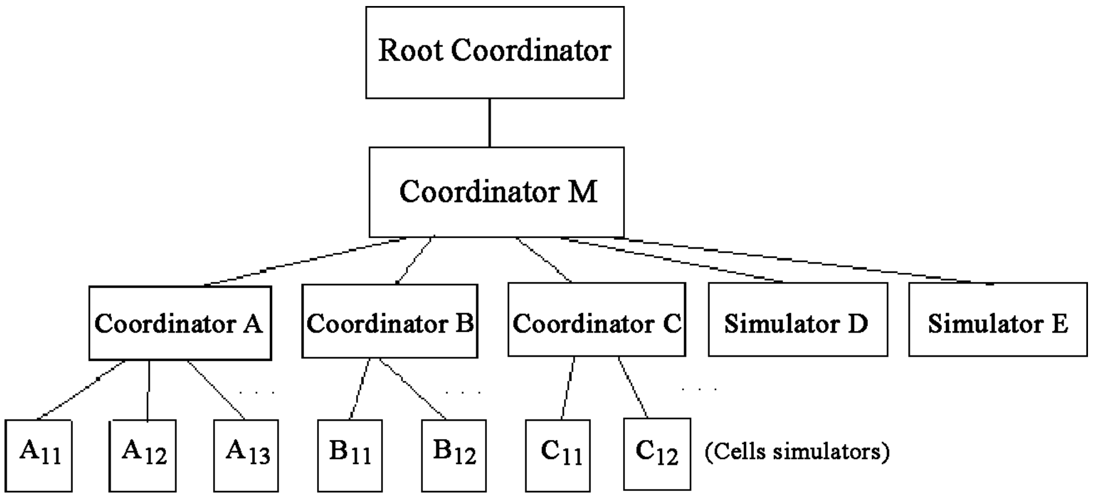


Figure 16. Hierarchy of simulation processors for model M

Root Coordinator:
Clock = 10;
Associated coordinator: Coordinator M.
	Coordinator M
	Father: Root Coordinator
	Children: { A, B, C, D, E }
	Associated coupled model: {M}
	Waiting list: { }
	Imminent children: { C }
	tn: 10; **tl:** 0.

Coordinator A
Father: Coordinator M
Children: { A_{11}, A_{12}, ... }
Associated coupled model: {A}
Waiting list: { }
Imminent children: { }
tn: 0; **tl:** 0.
	Coordinator B
	Father: Coordinator M
	Children: { B_{11}, B_{12}, ... }
	Associated coupled model: {B}
	Waiting list: { }
	Imminent children: { }
	tn: 0; **tl:** 0.

Coordinator C
Father: Coordinator M
Children: { C_{11}, C_{12}, ... }
Associated coupled model: {C}
Waiting list: { }
Imminent children: {(1,9) }
tn: 0; **tl:** 0.
	Coordinators D, E
	Father: Coordinator M
	Children: { }
	Associated coupled model: {D} ({E })
	Waiting list: { }
	Imminent children: {}
	tn: 0; **tl:** 0.

Figure 18. Initial contents of simulators/coordinators' data structures

The message contents are the following: < Type, Source | Destination, Date, [Value] >. The simulation begins when the root coordinator creates the message: < *, Root, 10 >.

This message is sent to the M coordinator, which queries its list of imminent children. There, the coordinator selects model C (the first of its imminent list), and sends the message < *, M, 10 > to its coordinator. Coordinator C receives this message, and selects its imminent child (in this case, cell [1,9]) by consulting its imminent children list. It also verifies that the time of next-event is equal to the simulated time included in the message.

The message < *, C, 10 > is sent, and arrives at the simulator C_{19}, which will execute the internal transition function. As a first step, the time of next event **tn** is verified. As it is the same as that of the *-message, the arrived message is correct and the output function is executed, activating the local computation function. The second rule of the second expression in the function's specification is true. Therefore, the result of the local computation is s′ = 0. As s = 1, the cell's state has changed and its present value should be output.

A Y-message is generated with the following values: < Y, C (1,9), 10, 0 > and it is transmitted to the C coordinator. After, the event times corresponding to the coordinator are updated: tl = tn = 10; tn = tl + D(s) = 10 + ∞ = ∞. Finally, the simulator updates the local values of the neighborhood set by making $a_{22} = 0$ in the neighbor's values of cell C_{19}.

The Y-message is received by the coordinator C, which queries its coupling scheme. As this message is local to the C model, it is translated into an X-message, and sent to the following cells: (1,10), (1,8), (2,8), (2,9), (2,10), (4,8), (4,9), (4,10). The *-message < *, C (1,9), 10 > is sent to each of the cell's simulators and the cells numbers are added into the coordinator's waiting list. In each simulator, the message time is checked to see if it is between the last event and next event times. As this is correct, the elapsed time and the event times are updated. Following, the input functions are executed. The input values and its corresponding σ are queued in the σqueue, as shown previously. In this case, D(s) includes the value of the transport or inertial delays.

Then, each simulator creates a message < done, C (i, j), 12 > because the transport delay is of 2 time units. When these messages are received, the coordinator eliminates the children from the wait-list and adds the message in its imminent list. The imminent children are: { (1,10), (1,8), (2,8), (2,9), (2,10), (4,8), (4,9), (4,10) }.

When the waiting list is empty, all the influencees have finished with the execution of the external transition function. Therefore, the smallest imminent child is ready for execution. The first element of the imminent children's list is chosen, and the message < done, C (1,10), 12 > is sent to the M coordinator. In this way, if more than one imminent model exists in the hierarchy, one of them can be chosen to be simulated. To do so, M adds the model C in its queue of imminent children. As this is the only message in the queue, it will be the imminent child. Therefore, the next and last event times can be updated. A done message is sent to the root coordinator because the coordinator is not waiting other events (the waiting list is empty).

The root coordinator updates the clock to 12, and creates a new message: < *, Root, 12 >. When the M coordinator receives this message, it verifies the time, takes the imminent child C from its imminent child queue, and sends the message < *, M, 12 > to its coordinator. The coordinator selects the imminent children (1,10) and activates its simulator.

These procedures are repeated for each of the imminent children of C coordinator. When the simulators are activated in turn, the local computing functions are executed. Every cell of those activated by the event in (1,9) remains in a quiescent state, except cell (1,10). Recalling the specification of Figure 9, the only valid rule for these cells is the fourth of the second sentence. Therefore, the state value will remain in 0 and the new external events are not generated. Then, a message $<$ done, C (i, j), ∞ $>$ will be generated for each of these cells and transmitted to the C coordinator.

Instead, for the cell (1,10) the first rule of the first sentence is valid. The cell's state changes, λ is executed, and a message < Y, C (1,10), 12, 1 > is generated and sent to the C coordinator. This message is expanded to the neighbor cells by repeating the procedures presented recently. This will cause actualization in the local copies of the neighborhoods for each simulator. In this case, the Y-message also influences the links that are external to the C model. Therefore, it should be transmitted to the parent coordinator M to execute the $Z_{i\,j}$ function, allowing it to send the change to the other models. When the message is received by the M coordinator, it queries the external coupling function, and it determines that the coupling $Y(1,10)_C \rightarrow X(2,10)_A$ should be used.

Therefore, it generates the message < X, A (2,10), 12, 1 >, which is sent to the A coordinator, which expands it up to its arrival to the cell A(2,10), where the external transition function is executed and the message is queued, generating a message <done, A(2,10), 15>.

4.2 Flat Simulation of the Example

Now, let us consider the execution of the same examples using the flat coordinator. In this case, message interaction is completely avoided within the cell spaces, allowing the improvement of execution times. We can see that the Next-Events lists of all the cellular models are empty, except for the C model. For this coordinator, Next-Events = {(1,9), 10}.

The coordinator executes, taking the data for the first event of the list. The simulation times are updated: tl = 0, tn = 10 and e = 0. In this case, when the local computing function is executed, the result obtained is New-state = 0. As Cells[1,9].state = 1, the new state should be stored in the New-states list. Therefore, New-states = { (1,9), 0}. After, as the cell is not in Y_{list}, nor are inertial delays used, the following sentences are executed, making Next-event time = 10 + 2; Next-events = { <(1,10), 12>, <(1,10), 12>, <(1,8), 12>, <(2,8), 12>, <(2,9), 12>, <(2,10), 12>, <(4,8), 12>, <(4,9), 12>, <(4,10), 12> }.

After, as the Next-Events list does not have other events with time = 10, one simulation step has finished. Therefore, the cell space is updated by doing: c [1,9] = 0; New-states = {}. When the next events have been updated, a message < done, 10, C > is sent to the upper-level coordinator. The coordinator detects that C is the imminent child. Then, the Root Coordinator updates the global clock and sends a *-message to the C coordinator.

Now, tn = 12, tl = 10, and e = 0. Cell (1,10) is chosen from the Next-Events list. In this case, New-state = 1; and Cells[1,9].state = 0, therefore New-states = {<(1,10), 1>}. The cell is in the Y_{list}, therefore a Y-message is created and it is sent to the M coordinator. The coordinator will react in the same way explained in the previous section. When the message < Y, C (1,10), 1, 12 > arrives at the M coordinator, it is translated into the message < X, A(2,10), 12, 1 >, which will be transmitted to the flat coordinator A. This coordinator will insert it into the Next-Events queue, to be treated in the future.

5. Development Experiences

The N-CD++ [12] was built using the formal specifications of Cell-DEVS models. The cell's behavior is described using a specification language based in the one presented in Section 3. As previously stated, only the size for the cell's space, the delays, the borders, and the cell's neighborhood parameters must be defined. Also, the definition of the model's rules and the construction of the input/output lists must be defined. With these basic parameters, the tool builds the cellular models and simulates them.

A wide variety of cellular models were developed using this approach. The tool and the simulation results for these models can be found at *http://www.dc.uba.ar/people/proyinv/celldevs*. Some of these models include the following:

- Physical systems: heat diffusion models (2/4 dimensions), combination with air pressure models, crystal growth, excitable media, particle collision models, and surface tension.
- Cryptography: based on Wolfram's CA combined with timing delays.
- Ecological models: ant foraging models, Wa-Tor (sharks and fishes in the ocean), basic fire spread models.
- Urban traffic models.
- Dynamic heat seekers.
- General applications: the "life" game, parity checkers, sorting algorithms.

```
[ top]                                              // Top level coupled model
components : life                                   // One component (Mi)
in : in                       out : outG1 outG2     // One input and two output ports (I)
link : out1@life outG1                              // Definition of Ii
link : out2@life outG2                              // and Zij function
link : in in@life


[ life]
type : cell                                         // Cell-DEVS model
width : 20            height : 15                   // Dimension and size (n, t1..tn)
delay : transport     defaultDelayTime  : 100       // Delay function     (Delay, d)
border : wrapped                                    // Border definition  (B)
neighbors : life(-1,-1) life(-1,0) life(-1,1)       // Neighborhood shape (N)
neighbors : life(0,-1) life(0,0) life(0,1) life(1,-1) life(1,0) life(1,1)

in : in               out : out1 out2               // Xlist and Ylist
link : in in@life(1,1)
link : output1@life(1,1) out1        output2@life(1,1) out2

localtransition : life-rule                         // Local computing function (τ)
portInTransition : in@life(1,1)specialRule

[ life-rule]
rule : 1 100 {  (0,0) = 1 and truecount = 4 }
rule : 1 100 {  (0,0) = 0 and truecount = 3 }
rule : 0 100 {  t }

[ specialRule]
rule : {  portValue(thisPort) }  1 {  t }
```

Figure 19. Life game specification using N-CD++

- Plant models: autonomous robots together with classification/conveyor belt system.

One of the simplest implemented examples includes a modification of the Life game [13]. This modification includes inputs, allowing one to analyze the transient behavior of the system. Figure 19 shows the definition of each of the parameters for these Cell-DEVS model, defined with the specification language.

The examples were simulated by using the hierarchical and flat approaches. Figure 20 shows the results of executing the Life game using both approaches. In every case more than 75% of cells were active initially. They have increased the number of cells in the space. The second test used a fixed size for the space (2,500 cells), and the length of the simulation was changed.

The results in the remaining models were similar. For instance, Figure 21 shows the results of a one-way traffic model and a heat diffusion model. A fixed-size cell space was considered (60% of active cells), and the length of the simulation was changed.

The results obtained for all the remaining examples are similar to these ones. It can be seen that, in every case, the execution of the simulation involves a minimum execution time, independent of the length of the simulation. Therefore, the tests were repeated to study the overhead used in initialization messages.

The initialization overhead is much bigger in hierarchical cell spaces, where the number of cells has a big influence. As can be seen, the time spent in initialization becomes meaningless for longer simulations.

Finally, the performance of the environment was tested against the execution of the same models implemented by hand. We have compared the overhead

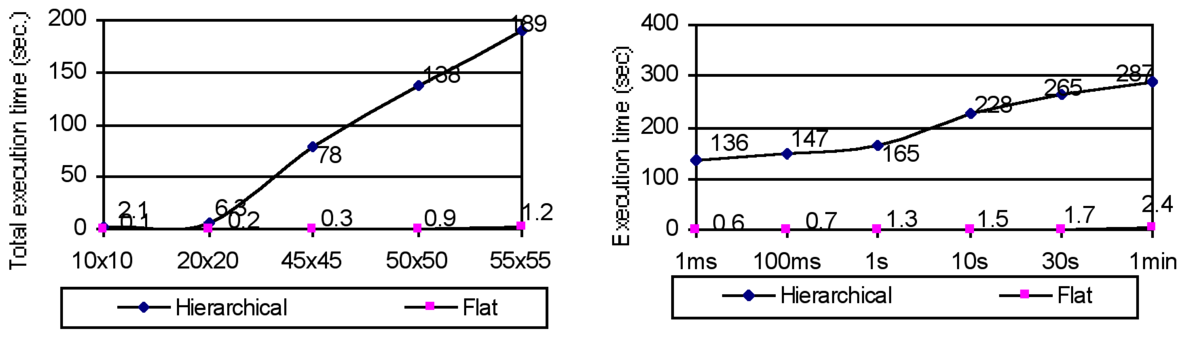
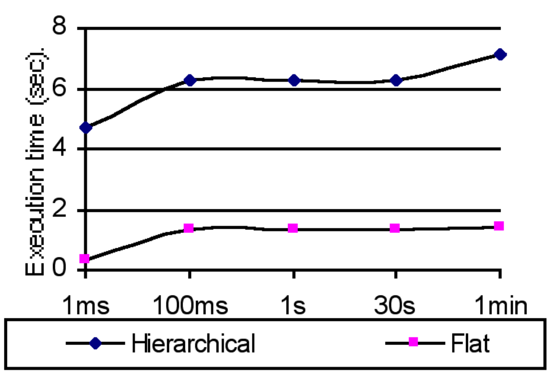


Figure 20. Execution of the Life game; **20(a)**. Increasing the size of the space; **20(b)**. Simulation duration

introduced by the tool (using flat models) against the execution time of synchronous and asynchronous cellular automata (that is, using a discrete or continuous time base). The results in the Figure 24 show that the tool introduces at most a 20% of overhead in average against a standard CA.

A main goal when Cell-DEVS was defined was to reduce the development times for the simulations.



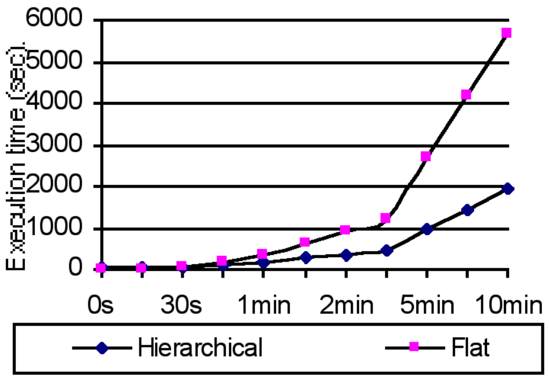
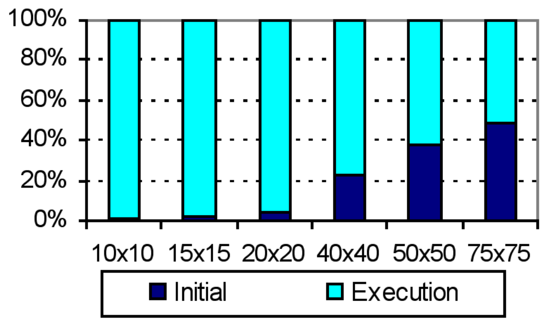


Figure 21. Hierarchical and flat models: **21(a)**. Traffic simulation; **21(b)**. Heat diffusion



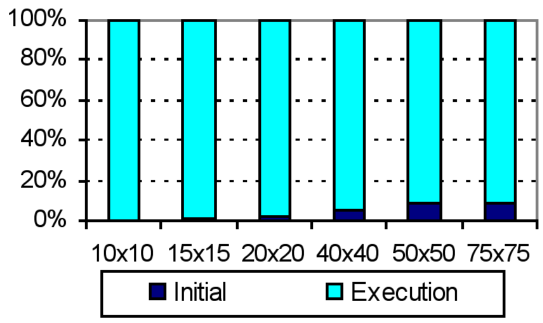
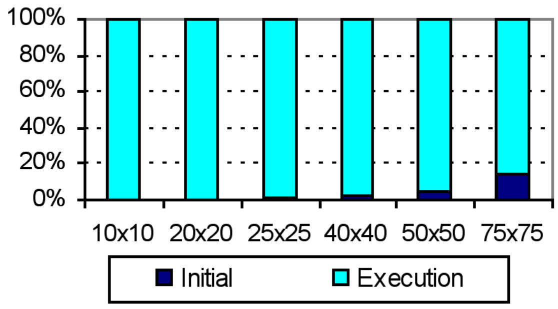


Figure 22. Initialization time (five minutes simulated time: **22(a)**. Hierarchical; **22(b)**. Flat



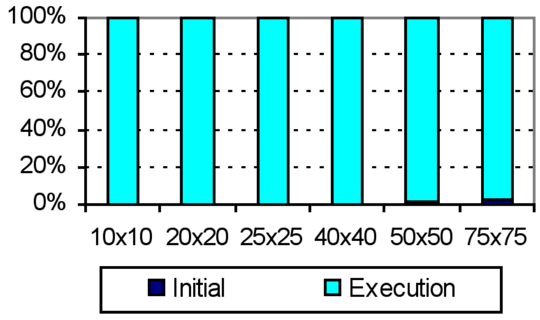
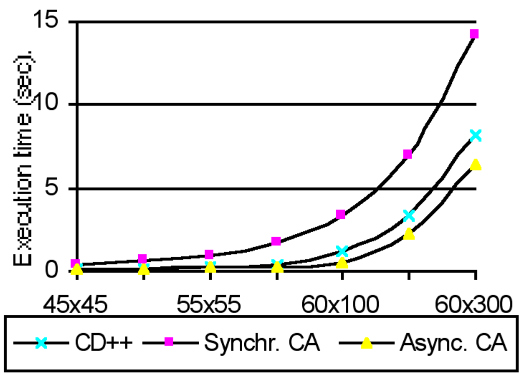


Figure 23. Initialization time (one hour simulated time): **23(a)**. Hierarchical; **23(b)**. Flat



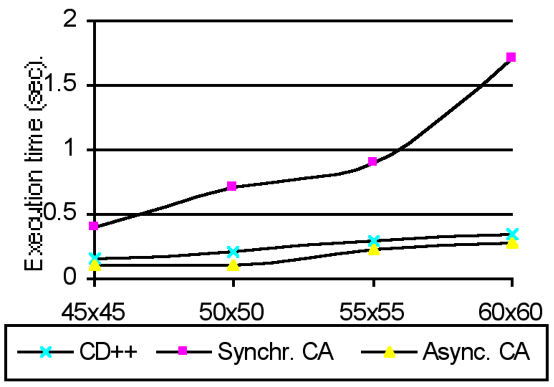


Figure 24(a). Execution results for the Life game; **24(b)**. Detail

The results obtained were promising, though several of the experiences here presented are prototypes to check the tool. Several data were recorded with the goal of analyzing the improvements in developments. The development times for the different solutions were considered, classifying different developers according to their development activities. The same problem specifications were delivered to different groups of developers. All of them had previous experience in software development activities.

Some of them have been assigned the task of developing the problems using any programming language. Others were asked to develop the same models using N–CD++. A third group was composed of expert users of the tool. The activities of each group were recorded and averaged, measuring man-hours of development times. When the software was delivered, some of the specifications were changed, allowing one to measure maintenance tasks. The results can be seen in Figures 25 through 27.

It can be seen that important improvements in the development times can be obtained, even more when expert users of the tool are compared with experienced programmers. The values obtained for novice users are related with the training to use the tool. In every case, a 16-hour training period was enough to start developing the models. The differences are bigger when models with multiple components of different behavior were considered. The reductions in the development times are due to the fact that specification of the cellular models is translated into executable models. Also, a simple set of verification aids was included,

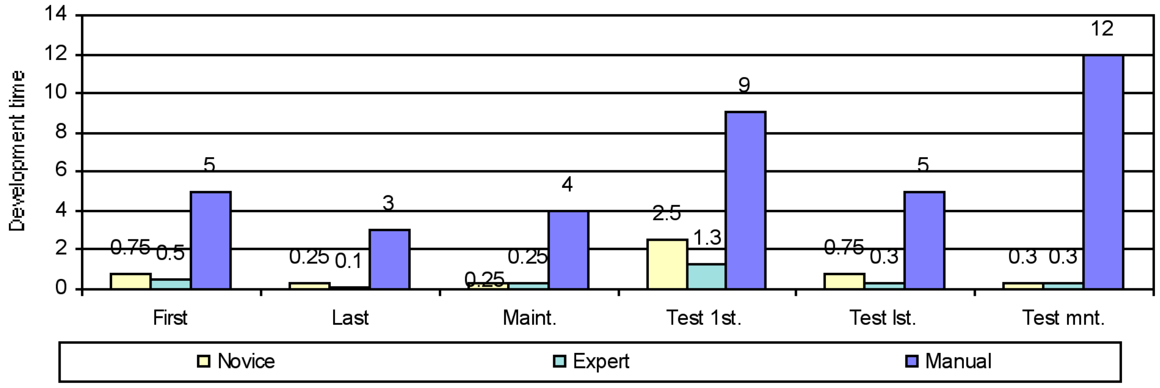


Figure 25. Comparison of development times for the heat seekers

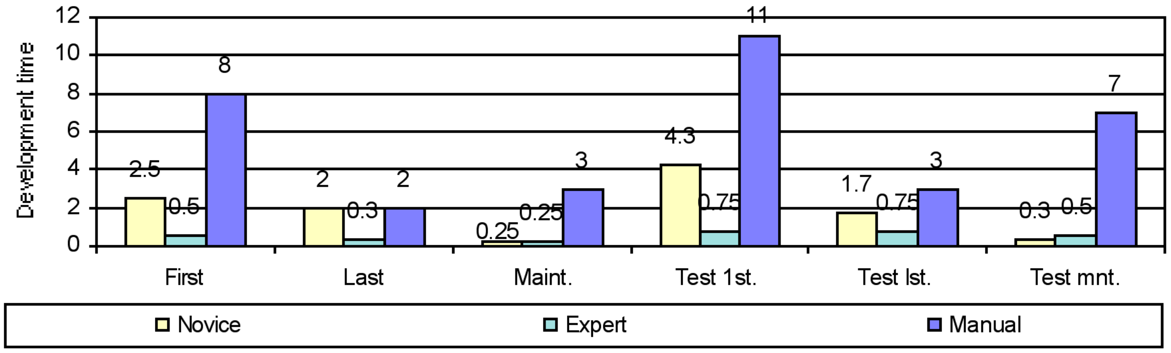


Figure 26. Comparison of development times for the traffic simulation

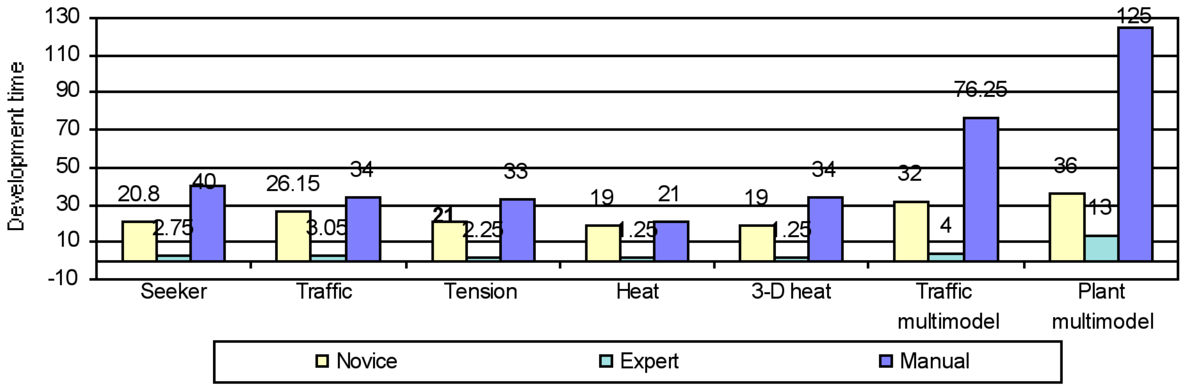


Figure 27. Comparison between total development times

allowing one to find several errors of the specification at runtime. These aids allow one to detect some inconsistencies in the model definition:

- Ambiguous models: a cell with the same precondition (state and neighbors) can produce different results;
- Incomplete models: no result exists for a certain precondition;
- Non-deterministic models: different preconditions are valid simultaneously. If they produce the same result, the simulation can continue, but the modeler must be notified. Instead, if different results are found, the simulation should stop because the future state of the cell cannot be determined.

6. Conclusion

We have presented the implementation results of a paradigm for modelling and simulation of cellular models. The Cell-DEVS paradigm allows the automatic definition of the cell spaces based on the DEVS formalism. Several modelling paradigms can be integrated in an efficient fashion, allowing multiple points of view for each model. The transport or inertial delay concepts specify complex behavior in a simple fashion, independently of the quantitative complexity of the models.

Two simulation mechanisms were included. The first one considered hierarchical simulation. Afterwards, a method to flatten the hierarchical description of the cell spaces was given. The performance improvements for the flat models provided speedups from two to seven times in the execution for the cellular models.

The formalism allows one to improve the security and cost in the development of the simulations. Experimental results of application showed improvements for expert developers. The main gains have been reported in the testing and maintenance phases, the more expensive for these systems.

The use of a modelling and simulation tool introduced overhead, mostly in the initialization phase. This happened because the tested solutions only executed for short times, and the initialization activity can be better amortized in longer simulations. Nevertheless, if a parallel coordinator is introduced, an execution speedup of several orders of magnitude can be achieved without changing the specifications. The use of parallel implementation for each hand-coded problem can produce ten-fold delays in the development times.

7. Acknowledgements

This work was developed while G. Wainer was at the Departamento de Computación Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Argentina. It was partially supported by ANPCYT Project 11-04460 and UBACYT Projects JW10 and TX-004. Further information about this work can be found at the author's Webpage, *http://www.sce.carleton.ca/faculty/wainer.html*.

8. References

[1] Zeigler, B. *Theory of Modeling and Simulation*, Wiley, 1976.

[2] Giambiasi, N.; Miara, A. "SILOG: A Practical Tool for Digital Logic Circuit Simulation." *Proceedings of the 16th. D.A.C.*, San Diego, 1976.

[3] Wolfram, S. "Theory and Applications of Cellular Automata." *Vol. 1, Advances Series on Complex Systems*. World Scientific, Singapore, 1986.

[4] Wainer, G.; Giambiasi, N. "Specification, Modeling and Simulation of Timed Cell-DEVS Spaces." Technical Report No. 98-007. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. 1998.

[5] Wainer, G.; Giambiasi, N. "Timed Cell-DEVS: Modelling and Simulation of Cell Spaces." Invited paper in *Discrete Event Modeling & Simulation: Enabling Future Technologies*, in print by Springer-Verlag, 2000.

[6] Zeigler, B. *Multifaceted Modeling and Discrete Event Simulation*. Academic Press, 1984.

[7] Zeigler, B.; Kim, T.-G.; Praehofer, H. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, Academic Press, 2000.

[8] Moon, Y.; Zeigler, B.; Ball, G.; Guertin, D.P. "DEVS Representation of Spatially Distributed Systems: Validity, Complexity Reduction." *IEEE Transactions on Systems, Man and Cybernetics*, pp 288-296, 1996.

[9] Zeigler, B.P., et al. "The DEVS/HLA Distributed Simulation Environment and its Support for Predictive Filtering." 1998, DARPA Contract N6133997K-0007: ECE Dept., University of Arizona, Tucson, AZ.

[10] Wainer,G.; Frydman, C.; Giambiasi, N. "An Environment to Simulate Cellular DEVS Models." *Proceedings of the SCS European Multiconference on Simulation*, Istanbul, Turkey, 1997.

[11] Wainer, G. "Discrete-Events Cellular Models with Explicit Delays." PhD Thesis, Université d'Aix-Marseille III, 1998.

[12] Rodriguez, D.; Wainer, G. "New Extensions to the CD++ Tool." In *Proceedings of SCS Summer Multiconference on Computer Simulation*, 1999.

Gabriel A. Wainer received a Licentiate degree (MSc, 1993) and a PhD degree (1998, with honours) from the Universidad de Buenos Aires, Argentina, and DIAM/IUSPIM, Université d'Aix-Marseille III, France. He is Assistant Professor at the SCE Dept., Carleton University, Ottawa, Canada. He was Assistant Professor at the Computer Sciences Dept. of the Universidad de Buenos Aires, Argentina, and had been a Teaching and Research Assistant in that department since 1988. He has published several articles in the field of operating systems, real-time systems and discrete-event simulation. He has been the PI of several research projects and was a referee for many international conferences, journals and research projects. He is author of a book on real-time systems and co-author of a book on operating systems. He is a member of the SCS Board of Directors. His current research interest is related to modelling methodologies and tools, modelling and simulation of cellular models, parallel execution of models and real-time simulators.

Norbert Giambiasi has been a full Professor at the University of Aix-Marseille since 1981. In October 1987, he created a new engineering school and a research laboratory, LERI, in Nîmes, in southern France. He was the Director of Research and Development of this engineering school. In 1994, he returned to the University of Marseilles where he created a new research team in simulation. He has written a book on CAD and is the author of more than 150 articles published internationally. He has continued to be the Scientific Manager of more than 50 research contracts with E.S Dassault, Thomson, Bull, Siemens, CNET, Esprit, Eureka, Usinor, and others, and has supervised more than 40 PhD students. He has been a referee for both national and European research projects. He also referees many articles for SCS publications. His current research interests converge on specification formalisms of hybrid models, discrete event simulation of hybrid systems, CAD systems, and design automation.