Applying Cell-DEVS in Models of Complex Systems

Javier Ameghino

Departamento de Computación
FCEN – Universidad de Buenos Aires
Planta Baja, Pabellón I
Ciudad Universitaria, 1428
Buenos Aires, Argentina
jameghin@dc.uba.ar

Gabriel Wainer
Ezequiel Glinsky

Dept. of Systems and Computing Engineering
Carleton University
1125 Colonel By Drive
K1S 5BE. Ottawa, ON. Canada
{gwainer,eglinsky}@sce.carleton.ca

Keywords : Modeling methodology; DEVS models, cellular automata, Cell-DEVS models; simulation tools.

Abstract

Cell-DEVS is an extension to the DEVS formalism that allows the definition of cellular models. CD++ is a modeling and simulation tool that implements DEVS and Cell-DEVS formalisms. We show the application of the paradigm through different examples developed in CD++. Complex applications can be implemented in a simple fashion, and they can be executed effectively.

1. INTRODUCTION

Simulation is a tool that enables a better understanding of complex physical and natural systems. Difference equations have been used to develop simulation models of these systems (see, for instance, [1, 2]). In the last years, several simulation models of real systems have been represented as cell spaces [3, 4]. Cellular automata [5] are dynamical systems in which time and space are discrete. A cellular automaton is defined as an infinite n-dimensional lattice of cells whose values are updated according to a local rule. Updates are done simultaneously and synchronously using the state values of the current cell and its neighborhood, which is a finite set of nearby cells.

The use of a discrete time base may constrain the precision of the model. In addition, the execution of cellular automata usually requires a large amount of computation time, primarily due to its synchronous nature. Timed Cell-DEVS solves these problems using the DEVS (Discrete EVents systems Specification) formalism [6] to define a cell space where each cell is defined as a DEVS model [7]. Thus, it is possible to build discrete-event cell spaces, improving their definition by making the timing specification more expressive.

The DEVS formalism was proposed to model discrete events systems. Basic DEVS models (called **atomic**) are specified as black boxes, and several DEVS models can be integrated together forming a structural model (called **coupled**).

A DEVS atomic model can be defined as:

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

where **X** represents a set of input events, **S** a set of states, and **Y** is the set of output events. Four functions manage the model behavior: d_{int} the internal transitions, d_{ext} the external transitions, l the outputs, and **D** ?the duration of a state. Each model uses its input and output ports to communicate with other models. External events are received via the input ports, and the model defines its behavior upon the reception of such inputs. The internal events produce state changes, and results are communicated to its influencees using the output ports.

A DEVS coupled model can be defined as:

$$CM = < X, Y, D, \{Mi\}, \{I_i\}, \{Z_{ij}\} >.$$

where **X** is the set of input events, **Y** is the set of output events, **D** is an index for the components of the coupled model, and for every i in D, $\mathbf{M_i}$ is a basic DEVS component (*i.e.*, an atomic or coupled model). The list of influencees $\mathbf{I_i}$ of a given model is used to determine the models to which outputs must be sent. The function $\mathbf{Z_{ij}}$ translates outputs of a model into inputs for the other models. An index of influencees is created for each model ($\mathbf{I_i}$). For every j in the index, outputs of model $\mathbf{M_i}$ are connected to inputs in model $\mathbf{M_j}$.

In the Cell-DEVS formalism, a cell has a set of N inputs to compute its future state. Each input (generally received from the neighboring cells) is received through the model's interface, and is used to activate the local function. It is possible to associate either a transport or an inertial delay with each cell, allowing deferring the transmission of the execution results. A **transport** delay allows us to model a variable commuting time for each cell with anticipatory semantics. Thus, every scheduled event is actually executed. Using **inertial** delays, the semantics are preemptive: some scheduled events may not be executed due to a small interval between two input events. Therefore, the outputs of a cell are not transmitted instantaneously, but after the consumption of the delay.

A Cell-DEVS model advances through the activation of the internal, external, output and state's duration functions, as in other DEVS models.

A Cell-DEVS atomic model can be defined as:

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

where each cell uses the **N** inputs to compute its future state **S** using the function t. The new state of the cell is transmitted to its neighbors after the consumption of the delay function. **Delay** defines the kind of delay for the cell, and **d** its duration. This behavior is defined by the d_{int}, d_{ext}, l and **D** functions.

After each cell has been defined, they can be combined to form a coupled model, which is composed of an array of atomic cells.

A Cell-DEVS coupled model can be defined as:

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

where the cell space **C** is defined by this specification as a coupled model composed by an array of atomic cells with size $\{\mathbf{t_1} \mathbf{\; x \; ... \; x \;} \mathbf{t_n}\}$. Each cell in the space is connected to the cells defined by the neighborhood **N**. If the cell space is "wrapped", the cells in a border are connected with those in the opposite end. Otherwise, the borders **B** must have a different behavior from the remaining cells. The **Z** function allows the definition of internal and external coupling of cells in the model. This function translates the outputs of output port **m** in cell $\mathbf{C_{ij}}$ into values for the **m** input port of cell $\mathbf{C_{kl}}$. The input/output coupling lists can be used to interchange data with other models.

The **CD++** tool [8] was built following the definitions of DEVS and Cell-DEVS formalisms, and is used in this paper to model and simulate a number of real systems. Previous experiences on simulation of complex physical systems using Cell-DEVS and CD++ were described in [9,10].

In CD++, Cell-DEVS models are described using a built-in specification language, which provides a set of primitives to define the size of the cell space, type of borders, cell's interface with other DEVS models and the cell's behavior. The behavior of a cell (the τ function of the formal specification) is defined using a set of rules of the form:

VALUE *DELAY* **CONDITION**

Rules are evaluated to compute the new value of the cell upon the reception of an external event. Starting with the first listed rule, the *CONDITION* is evaluated. If the condition is satisfied, the new cell state is obtained by evaluating the *VALUE* expression. The cell will change to this new state after the specified *DELAY* time, and when it changes, it also sends output messages to all its neighbors. If the condition does not hold, the next rule is evaluated repeating this process until a rule is satisfied. If no *CONDITION* statement is satisfied, the simulation is aborted.

The CD++ specification language offers a vast collection of functions and operators, including: boolean (*AND*, *OR*, *NOT*, *XOR*, *IMP* and *EQV*), comparison (=, !=, <, >, <= and >=), and arithmetic (+, -, * and /). Different types of functions are also available: trigonometric, roots, power, rounding and truncation, module, logarithm, absolute value, minimum, maximum, G.C.D. and L.C.M. Other available functions allow checking if a number is integer, even, odd or prime. In addition, some common constants are defined.

In this paper, we show how to apply the Cell-DEVS formalism to simulate real phenomena. We describe different models that are implemented and executed using the CD++ tool, focusing on particular characteristics of each phenomenon. Previous experiences in the simulation of complex physical systems using Cell-DEVS and CD++ were presented in [9, 10], serving as a starting point for our study. We provide the results of our experiments in a graphical fashion enabling further analysis about the execution of such models.

2. APPLICATION EXAMPLES

Our first example shows the reproduction of Vibrio Parahaemolyticus bacterium, which is a marine germ. It lives in coast and estuary, sediment, plankton and its reproduction takes place at 15ºC either in the scams or the intestine of a fish. To survive, the bacteria need a minimal percent of salt and a pH between 7.5 and 8.5. They grow in an environment with temperatures ranging from 15ºC to 43ºC, being 37ºC the optimal value.

They need between 20 and 30 minutes to reproduce, however they cannot reproduce at temperatures below 8ºC. If the temperature is close to 8ºC, the reproduction requires more time. The bacteria are destroyed when exposed to temperatures higher than 60ºC for a period of 10 minutes or to high acid (pH) environments.

We focus on modeling the bacterium concentration while the temperature varies. The rest of the variables that may affect the experiment are assumed to be appropriate for the normal growth of the bacteria.

The evolution of the bacteria over the surface of a fish is modeled using a Cell-DEVS component. We couple a DEVS component to introduce temperature changes between -10ºC and 0ºC. The temperature in a cell is calculated as the average of its neighbors, and the diffusion time is 1000ms. We have two surfaces, the first representing the concentration of bacteria, and the second showing the variation of temperature.

The rules that govern the reproduction of bacteria are the following:

1. If the cell temperature is lower than 8ºC for a period of 10000ms then the bacterium does not grow.

2. If the cell temperature is within 8ºC and 60ºC for a period of 30000ms then the bacterium grows.

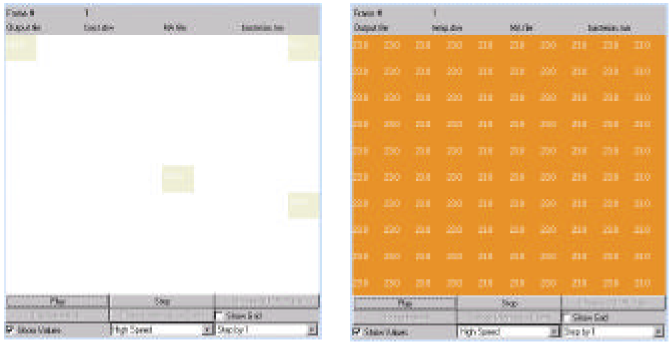
3. If the cell temperature is above 60ºC for a period of 10000ms then the bacterium dies.

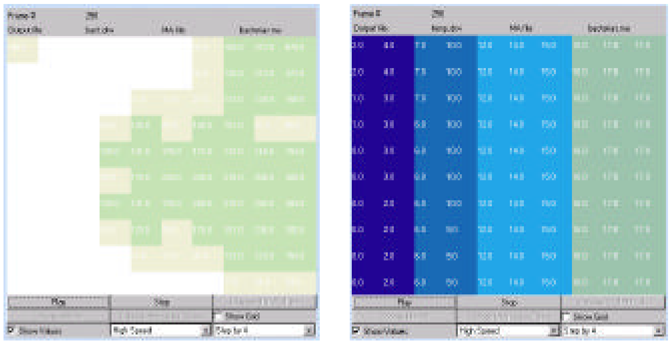
We use inertial cell delay and define that a cell reaching the concentration of 100 germs begins infecting the neighboring cells.

```
[top]
components : contamination Coldgenerator@Generator
link : out@Coldgenerator  inputCold@contamination

[Coldgenerator]
distribution : exponential          mean : 3
initial : 1   increment : 0

[contamination]
type : cell   dim : (10, 10, 2)
delay : inertial    border : nowrapped
neighbors : (-1,-1,0) (-1,0,0) (-1,1,0) (0,-1,0)
(0,0,0) (0,1,0) (1,-1,0) (1,0,0) (1,1,0) (0,1,1)
(-1,-1,1) (-1,0,1) (-1,1,1) (0,-1,1) (0,0,1)
(1,-1,1)  (1,0,1)  (1,1,1)
link : inputCold  in@contamination(0,0,1)
localtransition  : Evolution
portInTransition : in@contamination(0,0,1) setCold
zone : Temperatures { (0,0,1)..(9,9,1) }

[Temperatures]
rule: { ( if((-1,-1,0)!= ?,(-1,-1,0),0) + if((-
1,0,0)!=?,(-1,0,0),0) + if((-1,1,0)!= ?,(-
1,1,0),0) + … }                 1000     { t }

[Evolution]
rule: 0  10000 { cellpos(2) = 0 and (0,0,1) > 60 }
rule: {round(if((0,0,0)*2 > 99,0.7,1)*(0,0,0)*2) +
if((-1,-1,1)!=? and … } 30000 { cellpos(2)=0 and
(0,0,1) > 8 and statecount(?) = 5 * 2 }
…
rule: {(0,0,0)} 10000    { cellpos(2) = 0 }

[setCold]
rule:     { uniform(-10,0)}  500      { t }
```

Figure 1. Specification of the bacteria mode

Figure 1 shows the specification of such a model using the CD++ tool. Firstly, we declare the top model's components, *Coldgenerator* and *contamination*, and its links. Secondly, we define *Coldgenerator*, a DEVS model that generates cold temperatures using an exponential distribution function with the specified parameters. Thirdly, the Cell-DEVS model *contamination* is defined. We specify several parameters to describe this model, such as its size, neighborhood shape, type of delay, borders, input ports and connections with *Coldgenerator*. Finally, we describe the rules for the Cell-DEVS component. For each rule, the value, the delay and the condition are specified. The *Temperature* section represents the local computing function for the behavioral temperature model. A single rule defines the temperature as the average of the temperature of the neighboring cells. The *Evolution* rules describe the bacterium behavior. The *setCold* section states the range of temperatures generated by the DEVS component.


(a)


(b)

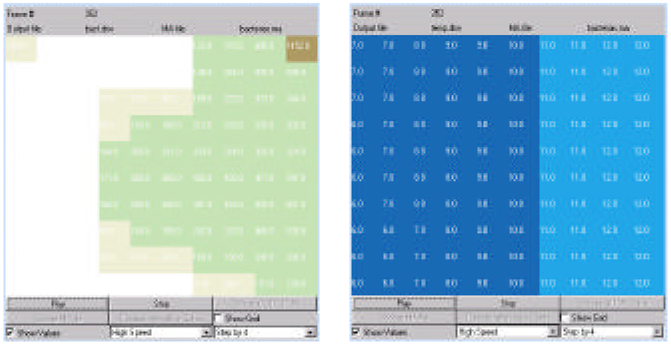

(c)

Figure 2. Results of bacteria propagation: (a) initial concentration; (b) after 1.5 hour; (c) after 4 hours

Figure 2 illustrates the results obtained when this model is executed, showing the evolution of the bacteria over the surface of a fish in a freezer for 4 hours. The left side represents the bacteria concentration, depicted in grayscale. The white areas represent regions where bacteria are not present, as a result of the extremely low temperature. Darker shades represent higher concentrations of bacteria. The right side represents the different temperatures of the surface, where darker shades represent colder temperatures.

The next example is an ecological model of ants following a specific path from an anthill to a source of food. When an ant finds food, it returns to the anthill leaving a hormone called pheromone on its path. The other ants use this hormone as a signal that leads to the source of food.

The model assumes that there are only 4 ants seeking food. Each cell in the Cell-DEVS space represents the ground (some vegetation) where there might be food. Hence, the cell state can be one of the following: vegetation that may contain pheromone; an ant seeking food; an ant following pheromone; food in the ground; or an ant following pheromone and returning to the anthill with food.

To avoid the collision of ants, when two or more ants want to move to the same place, they all stay in their positions and change the direction using a random variable until one ant can actually move. When an ant takes food from the ground, it changes its course to the opposite direction and follows the pheromone path to return to the anthill. In the case that there is no pheromone, the ant moves in a random way seeking the anthill or another pheromone path but leaving its pheromone to build the path for other ants.

Table 1 describes the cell state codification using a 5-digit natural number.

Table 1. Cell state codification of the ants model

Ten of thoudsands	Thoudsands unit	Hundred	Ten	Unit
0	0	F	F	D
1	0	0	C	C
2	Q	0	0	D
3	0	0	0	0

Where:

F	Pheromone concentration. It can vary from 1 to 99.
D	Ant direction 0: North, 1: East, 2: South, or 3: West
C	Quantity of food. It can vary from 1 to 99.
Q	Flag used to indicate if an ant is carrying or not food. Value 1 means carrying food, and 0 means ant seeking food

Figure 3 shows the execution of the model using CD++. The black cells represent two ants seeking food and the gray cells leading the paths in the upper left area of the graph represent two ants carrying food. The source of food is located in the lower right section of the graph, and different gray colors represent the concentration of pheromone showing the way to the food.

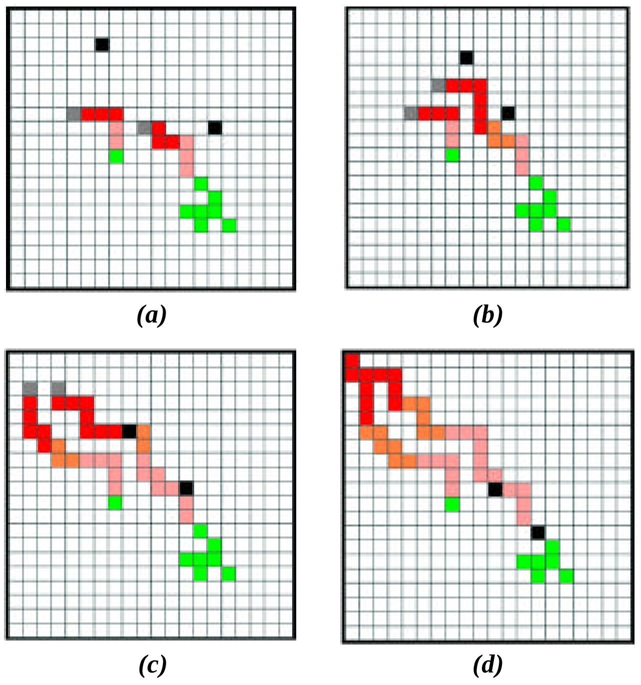


Figure 3. Ants moving on the ground: (a) two ants returning to the anthill and two ants seeking food; (b) two ants found pheromone; (c) both ants found the source of food using the pheromone path; (d) ants returning to the anthill following the pheromone path.

Figure 4 describes the model specification. We define the dimensions of the cell space, neighborhood, initial values and, finally, the rules that define the behavior of an ant.

```
[ant]
type : cell     dim : (20,20)
delay  : transport      border : nonwrapped
neighbors : (0,-2) (-1,-1) (0,-1) (1,-1) (-2,0)
neighbors : (-1,0) (0,0)   (1,0)  (2,0)  (-1,1)
neighbors : (0,1)  (1,1)   (0,2)
…
[rules]
rule : { (0,0) + 2 } 1000 { #Macro(isAnt00) and
#Macro(dir00) = 0 and ((#Macro(isAnt19) and
#Macro(dir19) = 3) or (#Macro(isAnt99) and
#Macro(dir99) = 1) or (#Macro(isAnt08) and
#Macro(dir08) = 2))}
rule : { (0,0) + 2 } 1000 { #Macro(isAnt00) and
#Macro(dir00) = 1 and ((#Macro(isAnt19) and
#Macro(dir19) = 2) or (#Macro(isAnt20) and
#Macro(dir20) = 3) or (#Macro(isAnt11) and
#Macro(dir11) = 0)) }
…
rule : { 21003 } 1000 { #Macro(isAntB00) and
#Macro(dir00) = 2 and #Macro(isAntB91) and
#Macro(dir91) = 1 }
rule : { 0 }     1000 { #Macro(isAntB00) and
#Macro(dir00) = 2 and #Macro(isNothingAnt01) }
…
rule : {(0,0)} 10 {t}
```

Figure 4. Specification of the ants model

We use different macro definitions to avoid large statements in the specification of rules, reducing development time. In this case, macros provide an easy mechanism for frequent statements such as checking the existence of an ant, food or pheromone in the neighboring cells. Hence, the rules specify the behavior of an ant based on its direction, current location, and the value of the adjacent cells.

The following example represents people in a metro station waiting for the train. We restrict the model to only two groups of people and only one railroad car with two doors. People can either get in or get out from the train. We use a Cell-DEVS model to represent the metro station and the people moving on the platform, and a simple DEVS generator to model people arriving to the metro station. When a train arrives to the metro station, it is often the case that a person in the railroad car wants to get out but finds people trying to get in it on the platform. In this example we focus on the problems derived from this situation.

We define three classes of people: those who want to get out from the train and go to the platform exit; those who want to get in the train using the door A, and those who want to get in the train using the door B.

Figure 5 shows seven slides that resemble people arriving to the train station. Two light gray cells located on the right side of each slide represent the platform entrance. The gray cells represent people who want to get in the train using the door A, placed in the upper part of the Cell-DEVS grid. The dark gray cells represent people who want to get in the train using the door B, placed in the lower part of the grid. The rightmost slide in the figure shows two groups of people standing in the border of the platform waiting for the doors to open.

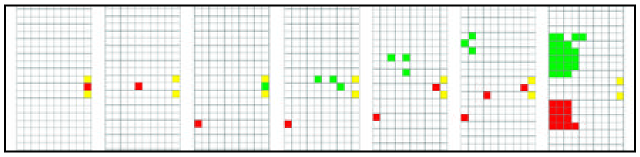


Figure 5. Execution results of metro station model

Figure 6 shows in detail the conflict of people trying to get in the railroad, represented by gray cells, that find people trying to get out from it using the same door, represented by dark gray cells. The light gray cell located in the left side of each slide denotes door A.

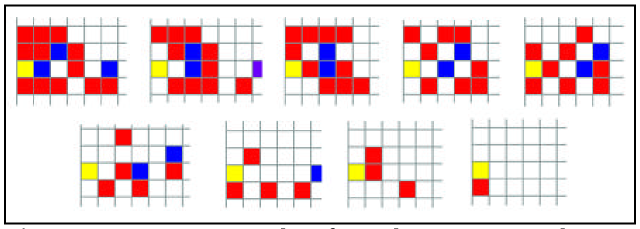


Figure 6. Execution results of people getting in and out using door A

Figure 7 shows the implementation of such a model using CD++. The components of the model are declared, the cell space is specified as *arriving*, and the DEVS component that generates people, called *PeopleGenerator*, is defined using an exponential distribution function. The rules represent the movement of the people using a combination of the direction (1: S; 2: E; 3: N; 4: W) and the door to be reached (1: A; 2: B). The rules determine the behavior of each person considering not only these two values, but also the existence of individuals in the neighboring cells. Hence, a person moves towards an adjacent cell based on the group she belongs to, its current location, direction, and state of the nearby cells.

```
[top]
components : arriving PeopleGenerator@Generator
in :   doorA    doorB
link : out@PeopleGenerator inputPeople@arriving
link : doorA    inputDoorA@arriving
link : doorB    inputDoorB@arriving

[arriving]
type : cell    dim : (30,10)   delay : inertial
defaultDelayTime : 500         border : nowrapped
neighbors : (0,-2) (-1,-1) (0,-1) (1,-1) (-2,0)
neighbors : (-1,0) (0,0)   (1,0)  (2,0)  (-1,1)
neighbors : (0,1)  (1,1)   (0,2)
in : inputPeople  inputDoorA   inputDoorB
link : inputPeople in@arriving(15,9)
link : inputDoorA  in@arriving(9,0)
link : inputDoorB  in@arriving(19,0)
…
[rules]
rule : 0  500 { (0,0)=24 and (0,1)=0 }
rule : 0  500 { (0,0)=23 and (-1,0)=0
              and (-1,-1)!=24 }
rule : 0  500 { (0,0)=22 and (1,0)=0
              and (1,-1)!=24 and (2,0)!=23 }
…
rule : 6  700 { ( (0,0)=1 ) }
rule : 7  700 { ( (0,0)=2 ) }
rule : 8  700 { ( (0,0)=3 ) }
…
rule : 19 500 { ( (0,0)=0 ) and ( (0,-1)=14 )
              and (1,0)!=23 and (-1,0)!=22 }

[createPeople]
rule : { ( randint(1)*10 ) + 1 } 10 { (0,0)=0 }
rule : {(0,0)} 10 {t}

[PeopleGenerator]
distribution : exponential
mean : 4       initial : 1      increment : 0

[doorHandler]
rule : 77 1 { (0,0)!=77 and (0,0)!=99 }
rule : 99 1 { (0,0)=77 }
rule : 0  1 { t }
```

Figure 7. Specification of the metro station

3. CONCLUSION

Cell–DEVS allows describing physical and natural systems using an n-dimensional cell-based formalism. Input/output port definitions allow the definition of multiple interconnections between Cell-DEVS and DEVS models. Complex timing behavior for the cells in the space can be defined using very simple constructions. The CD++ tool implements the Cell-DEVS formalism and entitles the definition of complex cell-shaped models. We showed how to develop several Cell-DEVS models using the CD++ toolkit, which provides a general framework to define and simulate complex generic models.

4. ACKNOWLEDGMENT

This work was partially funded by the Natural Sciences and Engineering Research Councel of Canada (NSERC), the High Performance Virtual Computing Lab (HPVcl), and the Institute of Robotics and Intelligent Systems (IRIS, Canada).

REFERENCES

[1] Lapidus, L.; Pinder, G.F. Numerical Solution of Partial Differential Equations in Science and Engineering, Wiley, New York, 1982.

[2] Weisbuch, G. Complex Systems Dynamics, Addison-Wesley, 1991.

[3] Sipper, M. “The emergence of cellular computing”. IEEE Computer. July 1999. Pp. 18-26.

[4] Talia, D. “Cellular processing tools for high-performance simulation”. IEEE Computer. September 2000. Pp. 44 –52.

[5] Wolfram, S. “Theory and applications of cellular automata”. Vol. 1, *Advances Series on Complex Systems*. World Scientific, Singapore, 1986.

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

[7] Wainer, G.; Giambiasi, N. “Timed Cell-DEVS: modelling and simulation of cell spaces”. In *Discrete Event Modeling & Simulation: Enabling Future Technologies*. 2000. Springer-Verlag

[8] Wainer, G. “CD++: a toolkit to develop DEVS models”. Software - Practice and Experience. Vol. 32, pp. 1261-1306. 2002.

[9] Ameghino, J.; Wainer, G. “Application of the Cell-DEVS formalism using N-CD++.” In Proceedings of the Summer Computer Simulation Conference. July 2000.

[10] Ameghino, J.; Troccoli, A.; Wainer, G. “Modelling and simulation of complex physical systems using Cell-DEVS”. In Proceedings of the 33rd SCS Summer Multiconference on Computer Simulation. Seattle, WA. USA. 2001.

JAVIER AMEGHINO has received a B. Sc. (1997) and M. Sc. in Computer Sciences (2000) from the Universidad de Buenos Aires, Argentina. At present he works at HP (Buenos Aires), and he is an Adjunct Research Assistant at the Computer Science Department of the Universidad de Buenos Aires, where he is currently a first year Ph.D. student. He is the coordinator of the ParDEVS lab in the same Deparatment.

EZEQUIEL GLINSKY has received a B. Sc. (2000) and M. Sc. in Computer Sciences (2002) from the Universidad de Buenos Aires, Argentina. He is currently a first year Master in Electrical Engineering student at the Department of Systems and Computer Engineering in Carleton University, Ottawa, Canada.

GABRIEL WAINER received the M.Sc. (1993) and the Ph.D. degrees (1998, with highest honours) at the Universidad de Buenos Aires, Argentina, and Université d'Aix-Marseille III, France. He is Assistant Professor at the Systems and Computer Engineering, Carleton University (Ottawa, Canada). He was Assistant Professor at the Computer Sciences Dept. of the Universidad de Buenos Aires, Argentina, and a visiting research scholar at the Arizona Center of Integrated Modelling and Simulation (ACIMS, University of Arizona). He has published more than 70 articles in the field of operating systems, real-time systems and Discrete-Event simulation. He is author of a book on real-time systems and another on Discrete-Event simulation. He has been the PI of several research projects, and participated in different international research programs. Prof. Wainer was a member of the Board of Directors of The Society for Computer Simulation International (SCS). He is the coordinator of a group on DEVS standardization. He is Associate Editor of the Transactions of the SCS. He is also a Co-associate Director of the Ottawa Center of The McLeod Institute of Simulation Sciences.