Proceedings of the 2004 Winter Simulation Conference
R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

MODELING AND SIMULATION OF COMPLEX SYSTEMS WITH CELL-DEVS

Gabriel A. Wainer

Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Dr.
Ottawa, ON K2G 6G3. CANADA

ABSTRACT

Cell-DEVS enables efficient execution of complex cellular models. The goal of Cell-DEVS is to build discrete-event cell spaces, improving their definition by making the timing specification more expressive. Different models built using Cell-DEVS were implemented in a modeling and simulation tool (CD++, crated following the formal specifications of the DEVS formalism). The applications range from biological systems to complex artificial systems. In this tutorial, we will introduce the main characteristics of Cell-DEVS, showing how to model complex cell spaces in an asynchronous environment. We will focus on the application of these techniques to improve model definition, which enables reducing development times of these models. We use a wide variety of previously defined examples in different domains of applications to illustrate the use of the techniques.

1 INTRODUCTION

The advance of science and technology in the last centuries has relied on models defining the properties of systems under study. In most cases, models were defined using mathematical representations, which enabled mathematical analysis techniques. However, these methods showed to be infeasible for studying very complex natural systems, and the artificial systems developed in the second half of the 20th century. Computers provided alternative methods of analysis. Models can be executed using computer simulation, allowing users to experiment with "virtual" systems. Computer simulation has enabled the analysis of natural and artificial systems with a level of detail unknown in earlier stages of scientific development.

Most of the early developments on modeling and simulation in digital computers were based on the use of differential equations for modeling, and time stepped numerical integration as simulation vehicle. Even today, most of the scientists and engineers prefer to use this approach. Simulation of continuous systems on digital computers requires discretization. Classical methods as Euler, Runge-Kutta, Adams, etc., are based on discretization of time resulting in a discrete time simulation model (Press et al. 1986).

In the last 20 years, a radically different technique, called Cellular Automata (CA) gained popularity. A Cellular Automaton represents a physical system organized as n-dimensional infinite lattice whose elements hold a state value and a very simple computing apparatus. The composite behavior of thousands of these cells can fully reproduce the behavior of a real system. A global transition function updates the state of every cell in the space through individual updates of the discrete values in each cell by using the present value for the cell and a finite set of neighboring cells. Conceptually, these local functions are computed synchronously and in parallel, using the state values of the present cell and its neighbors.

CA, originally defined by J. Von Neumann and S. Ulam, have received much attention recently (Wolfram 2002). Despite these efforts, CA still have several problems that constrain their power, usability and feasibility to analyze complex systems:

- The use of a discrete time base for cell updates constrains the precision and efficiency of simulated models. In order to achieve higher accuracy, smaller time steps must be used, increasing the demands of computing power.
- The discrete time implementation of the formalism makes it very difficult to handle time-triggered behavior in each of the cells, which is usually required in complex applications.
- CA do not describe adequately most of existing physical systems whose nature is asynchronous.

The Cell-DEVS formalism (Wainer and Giambiasi 2000) was defined in order to attack these problems. CA are defined using discrete variables for time, space and system states. Instead, Cell-DEVS is based on the DEVS (Discrete Event systems Specification) formalism (Zeigler,

Kim and Praehofer 2000), a continuous time technique. The goal of Cell-DEVS is to build discrete-event cell spaces, improving their definition by making the timing specification more expressive.

DEVS is an increasingly accepted framework for understanding and supporting the activities of modeling and simulation. DEVS is a sound formal framework based on generic dynamic systems, including well defined coupling of components, hierarchical, modular construction, support for discrete event approximation of continuous systems and support for repository reuse. DEVS theory provides a rigorous methodology for representing models, and it does present an abstract way of thinking about the world with independence of the simulation mechanisms, underlying hardware and middleware (Zeigler, Kim and Praehofer 2000). Different modeling formalisms were successfully mapped as DEVS (Petri Nets, Queuing Networks, Finite State Machines, etc.). Therefore, we can now build multiparadigm models, including cellular models that can interact with others described using different modeling techniques. DEVS and Cell-DEVS were implemented in a modeling and simulation tool, called CD++ (Wainer 2002). This toolkit was successfully used to develop different types of systems: biological (ecological models, electrical activity of the heart tissue, ant foraging systems, fire spread, etc.), physical (diffusion, binary solidification, heat transfer, etc.), artificial (traffic problems, seeking devices, etc.), and others (Ameghino and Wainer 2000; Ameghino, Troccoli, and Wainer 2001; Ameghino, Glinsky, and Wainer 2003; Muzy et al. 2002; Lo Tártaro, Torres, and Wainer 2001; Troccoli et al. 2002; MacSween and Wainer 2004). The models execute through the activation of an abstract simulation engine that is completely independent from the models themselves. Consequently, we were able to develop different kinds of simulators (stand-alone, parallel, distributed and real-time), which were used to execute all the models we will present in the following sections.

2 THE CELL-DEVS FORMALISM

A real system modeled with DEVS is described as a composite of submodels, each of them being behavioral (atomic) or structural (coupled). A DEVS atomic model is can be informally described as in Figure 1.

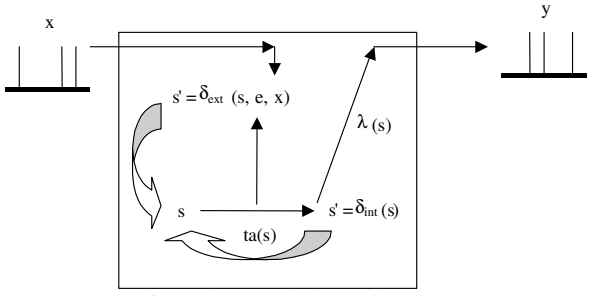


Figure 1: Informal Description of an Atomic Model

Each atomic model can be seen as having an interface consisting of *input* (x) and *output* (y) *ports* to communicate with other models. Every *state* (**s**) in the model is associated with a *time advance* (**ta**) function, which determines the duration of the state. Once the time assigned to the state is consumed, an internal transition is triggered. At that moment, the model execution results are spread through the model's output ports by activating an *output function* ($\boldsymbol{\lambda}$). Then, an *internal transition function* ($\boldsymbol{\delta_{int}}$) is fired, producing a local state change. Input external events (those events received from other models) are collected in the input ports. An external transition function ($\boldsymbol{\delta_{ext}}$) specifies how to react to those inputs.

A DEVS coupled model is composed by several atomic or coupled submodels, as in Figure 2.

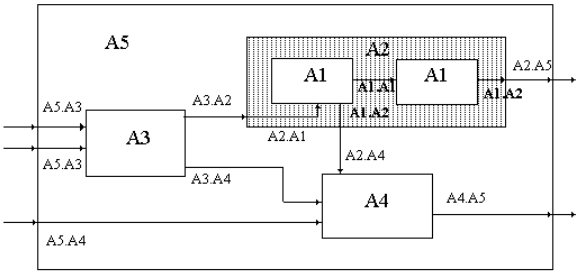


Figure 2: Informal Description of a Coupled Model

Coupled models are defined as a set of basic components (atomic or coupled), which are interconnected through the model's interfaces. The model's coupling defines how to convert the outputs of a model into inputs for the others, and how to handle inputs/outputs from/to external models.

Cell-DEVS combines CA and DEVS, allowing the implementation of cellular models with timing delays. Cell-DEVS improves execution performance of cellular models by using a discrete-event approach. It also enhances the cell's timing definition by making it more expressive. Each cell is defined as a DEVS atomic model, and it can be later integrated to a coupled model representing the cell space, as showed in Figure 3.

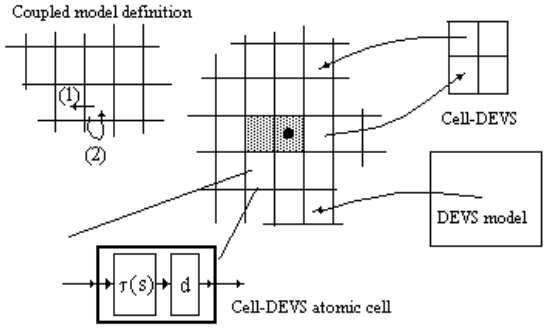


Figure 3: Informal Description of Cell-DEVS

Each cell uses N inputs to compute its next state. These inputs, which are received through the model's interface, activate a local computing function (**τ**). A delay (**d**) can be associated with each cell. The state (**s**) changes can be transmitted to other models, but only after the consumption of this delay. Two kinds of delays can be defined: *transport* delays model a variable commuting time (every state change is transmitted), and *inertial* delays, which have preemptive semantics (scheduled events can be discarded). This is informally presented in Figure 4.

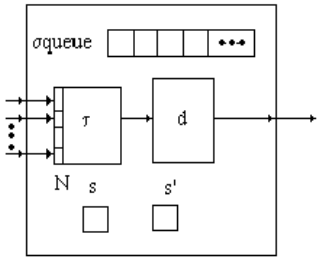


Figure 4: Description of a Cell-DEVS Atomic Model

Once the cell behavior is defined, a coupled Cell-DEVS can be created by putting together a number of cells interconnected by the neighborhood relationship. A coupled Cell-DEVS is composed of an array of atomic cells, with given size and dimensions. Border cells can have a different behavior due to their particular locations, which can result in a non-uniform neighborhoods. Finally, the model's couplings permit connecting these models with other external submodels.

CD++ (Wainer 2002) is a modeling tool that was defined using the formal specifications of Cell-DEVS, and the basic simulation techniques introduced in (Wainer and Giambiasi 2000; Zeigler, Kim and Praehofer 2000). The toolkit includes facilities to build DEVS and Cell-DEVS models. DEVS Atomic models can be programmed and incorporated onto a class hierarchy programmed in C++. Coupled and Cell-DEVS models are defined using a built-in language. Cell-DEVS coupled model specification includes the definition of the size and dimension of the cell space, the shape of the neighborhood and borders. The cell's local computing function is defined using a set of rules with the form:

POSTCONDITION DELAY { PRECONDITION }

These indicate that when the *PRECONDITION* is satisfied, the state of the cell will change to the designated *POSTCONDITION*, whose computed value will be transmitted to other components after consuming the *DELAY*. If the precondition is *false*, the next rule in the list is evaluated until a rule is satisfied or there are no more rules. Figure 5 shows the definition of a very simple example implementing the "Life" game (Gardner 1970).

The Cell-DEVS coupled model is defined by its size (*width=20, height=20*), its border (*wrapped*, meaning that the cells in one border communicate its results to neighbors in the opposite border), the shape of the neighborhood, and the type of delay (*transport*).

```
[life]
width : 20              height : 20
delay : transport       border : wrapped
neighbors : (-1,-1) (-1,0) (-1,1)
neighbors : (0,-1)  (0,0)  (0,1)
neighbors : (1,-1)  (1,0)  (1,1)
localtransition : life-rule
[life-rule]
Rule: 1 10 { (0,0) = 1 and ( truecount = 3
                          or truecount = 4 ) }
Rule: 1 10 { (0,0) = 0 and truecount = 3 }
Rule: 0 10 { t }
```

Figure 5: Definition of the Life Game

The rules define the behavior of each cell in the model. In this case, they state that an active cell (*(0,0) =1*) remains active when the number of active neighbors is 3 or 4 (*truecount* indicates the number of active neighbors) using a transport delay of 10 ms. If the cell is inactive (*(0,0) =0*) and the neighborhood has 3 active cells, the cell becomes active. In every other case, the cell remains inactive (*t* indicates that whenever the rule is evaluated, a *True* value is returned). CD++ is able to interpret these specifications, and execute a simulation of this model. The following sections are devoted to show how to define and execute different applications using this method. We will divide the examples to be discussed in three different areas: models of physical systems, models of biosystems, and artificial systems.

3 MODELS OF PHYSICAL SYSTEMS

Cellular models are well tailored to model a wide variety of complex physical systems, as we can see in (Wolfram 2002). In this section, we will show how to implement some of these systems using Cell-DEVS. Our first example defines a model of excitable media, a phenomenon appearing in magnetic fields. The model is defined in Figure 6.

```
[ExMedia]
dim: (9,9)  delay: transport  border: wrapped
neighbors : (-1,-1) (-1,0) (-1,1) (0,-1)
neighbors : (0,1) (1,-1) (1,0) (1,1) (0,0)
localtransition : Ex-rules

[Ex-rules]
rule : 0 100 {(0,0)=0 and statecount(2)=0 }
rule : 2 100 {(0,0)=0 and statecount(2)>0 }
rule : 1 100 { (0,0) = 2 }
rule : 0 100 { (0,0) = 1 }
rule : { (0,0) } 100 { t }
```

Figure 6: Definition of an Excitable Media Model

The model, originally presented in (Ameghino and Wainer 2000), we can recognize three states: resting, excited or recovering. Following our discussion in Section 2, the Cell-DEVS coupled model here defined has 9x9 cells, and 9 adjacent neighbors. The model is wrapped, and it uses transport delays. The *Ex-rules* section represents the local computing function. If the cell and its neighbors are not excited (value *0*), the cell remains resting. Resting cells with excited neighbors (value *2*) become excited. The third and fourth rules represent the cell transitioning towards the recovery state. In every other case, the cell keeps its present state. Figure 7 shows the execution results for this model using different neighborhoods. In Figure 7a, we use all the 9 adjacent neighbors (Moore neighborhood). In Figure 7b, we only use four adjacent cells (N-S-E-W), and Figure 7c shows the execution on an a hexagonal lattice (illustrated on a square grid).

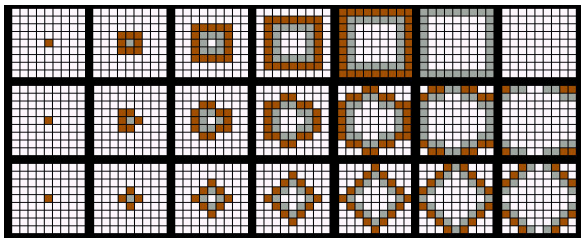


Figure 7: Results of ExMedia with Different Neighborhoods: (a) Moore; (b) Von Neumann; (c) Hexagonal Grid

The example in Figure 8 represents a model of surface tension, can be found in (Toffoli 1994), and it was previously defined as a Cell-DEVS in (Ameghino and Wainer 2000).

```
[Tension]
dim : (40,40)
delay : transport    border : wrapped
neighbors : (-1,-1) (-1,0) (-1,1) (0,-1)
neighbors : (1,-1) (1,0) (1,1) (0,0) (0,1)
localtransition : Ten-rules

[Ten-rules]
rule : 0 100 { statecount(0) >= 5 }
rule : 1 100 { t }
```

Figure 8: Surface Tension Model Specification

This Cell-DEVS uses a grid of 40x40, Moore neighborhood, transport delays and wrapped borders. We have two states: presence (value *1*) or absence (value *0*) of particles. This model represents a "majority vote" system. In each step, the new state depends of most neighbors. It remains in the cell if at least 5 of the 9 are occupied; otherwise, it becomes empty. Figure 9 shows how particles concentrate where there is more tension. The resulting behavior of the surface is a high level representation of the majority vote rules.

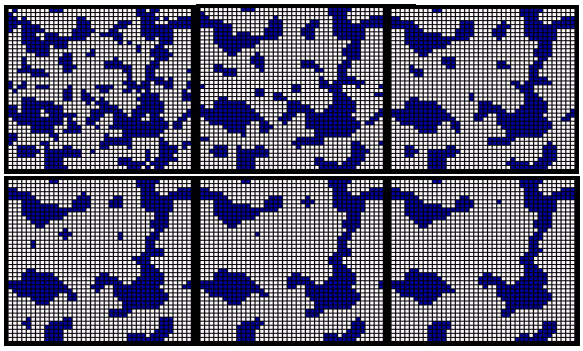


Figure 9: Execution Results of the Surface Tension Model

Flow-injection methods are analytical methods used for automated analysis of liquid samples. In a flow injection analyzer, a small, fixed volume of a sample is injected as a discrete zone using an injection device into a liquid carrier, which flows through a narrow tube. Because of convection at the beginning, and axial and radial diffusion later, the sample is progressively dispersed into the carrier as it is transported along the tube, producing reactive species that can be sensed (Troccoli et al. 2002).

We built a Cell-DEVS model describing the integrated conductivity in a flow-injection system. The system consists of a 0.025 cm radius tube, a 10.75 cm loop and a 9.25 reactor coil. A cell space of 25x 200 columns was defined, each cell representing a 0.001 x 0.1cm of a half tube section. Row 0 represents the center of the tube and row 24 the section of the tube walls. The value in each cell represents concentration of nitric acid.

To deal with convective transport and radial diffusion, the model reacts in two phases: transport and diffusion. Cell-DEVS models can be coupled with standard DEVS models. The coupling is done by linking a DEVS output port to a new cell's input port and defining a rule to be evaluated when a message is received through this new port. Here, the local computing function simulates the transport phase, and an external generator triggers the diffusion phase, as showed in Figure 10.

```
[Top]
components :  fia generator@ConstGenerator
link : out@generator diffuse@fia

[generator]
frequency : 00:00:00:014

[fia]
in : diffuse width : 200   height : 25
delay :   inertial border :   nowrapped
neighbors : (-1,-1) (-1,0) (-1,1) (0,-1)
neighbors : (0,0) (0,1) (1,-1) (1,0) (1,1)
localtransition :  transport
```

Figure 10: Definition of the FIA Coupled Model

The model is built as a coupled DEVS model with two components: a Cell-DEVS (named *fia*) representing the tube, and an atomic model (named *generator*). The generator has one output port (*out*) to send the diffusion-triggering event. This port is mapped to the *diffuse* input port of the *fia* model. This means all output events sent through the *out* port will be received as external events by the *fia* model through the *diffuse* port.

This model uses inertial delays. Thus, a cell with scheduled future value *f* will preempt this value if upon receiving an external event and evaluating the local transition rules, a new future value $f_1 \neq f_{is}$ obtained. In this case, f1 will be scheduled as the future value with a given delay *d*. The behavior of each cell is defined by this function, defined in Figure 11.

```
[transport]
rule : { (0,-1) } { 0.1 / ( 22.57878 * ( 1 -
    power( cellPos(0) * 0.001 + 0.0005 , 2)
      /0.000625))*1000 } { cellPos(1) != 0 }
rule : { 0.8 } { 0.1 / ( 22.57878 * ( 1 -
     power( cellPos(0) * 0.001 + 0.0005 , 2)
       /0.000625))*1000 } { cellPos(1) = 0 }
```

Figure 11: Definition of the Border Cells

The convective transport has been arbitrarily chosen from left to right. Thus, the local transition rule for the transport phase should set a cell's value to the current value of its (0,-1) neighbor cell. The rate at which this is done depends on the velocity of the flow (maximum at the center of the tube and decreasing towards its walls). This is stated in the first transport.

The delay is calculated using transport equations (Troccoli et al. 2002): for a pump with a constant flow of 1,33ml/min, the average speed is 11,29 cm/s. This value yields the number 22.57878 shown in the delay expression. In addition, `cellPos(0) * 0.001 + 0.0005` is the distance of the center of the cell to the center of the tube (*cellPos(0)* returns the cell's row).

The generic rule we have just given is only valid for all cells that have a valid (0,-1) neighbor. The left border cells (those in column 0) do not satisfy this prerequisite, stated in the condition component `cellPos(1) != 0`, and should therefore have a different rule.

The following rule is used for the left border cells. It simply states that for these cells the new value should be 0.8, which corresponds to the concentration of the carrier solution being pumped into the tube.

The model was run for 10s and the state of the whole cell space was logged every 100ms. Figure 12 shows a graphical representation of five different stages the FIA model (only half of the tube: the other half is symmetrical; the upper cells represent the center of the tube). The experiment starts at time 0, where the sample (white), is injected. At this moment, half of the tube contains the carrier solution (dark gray). The convective transport makes the sample disperse faster at the middle of the tube than near the walls. The experiment finishes when the whole tube contains the carrier solution only.

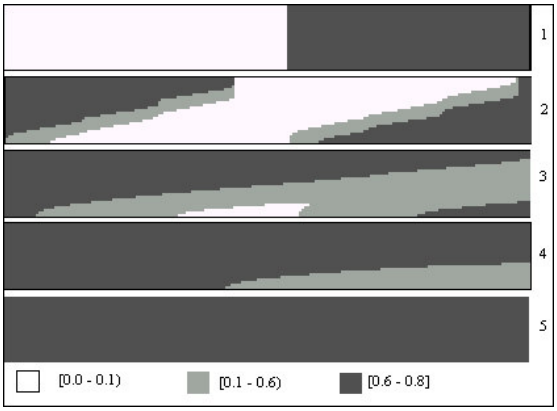


Figure 12: Different Execution Stages of the FIA Model

4 MODELS OF BIOSYSTEMS

In this section, we introduce a number of models with application to biological systems. Our first model, presented in Figure 13, defines the behavior of a flock of birds, previously presented in (Ameghino and Wainer 2004). The motion of a flock resembles a fluid, as an emerging behavior, which is the result of the individual interaction between birds in the flock.

```
[boids]
dim:(20,20) delay:  transport border: wrapped
neighbors : (-2,-2) (-2,-1) (-2,0) (-2,1) (-
2,2)  (-1,-2)  (-1,-1)  (-1,0)  (-1,1)  (-1,2)
(0,-2) (0,-1) (0,0) (0,1) (0,2) (1,-2) (1,-1)
(1,0) (1,1) (1,2) (2,-2) (2,-1) (2,0) (2,1)
...
[fly-rule]
rule:  {  1+if(((-2,-2)>100000),1,0)+if(((-2,-
1)>100000),1,0)+ if(((-2,0)>100000),1,0)+
if(((-2,1)>100000),1,0)+if(((-2,2)>100000)
,1,0) +if(((-1,-2)>100000),1,0)+if(((-1,-1)
>100000),1,0)+if(((1,0)>100000),1,0)+
if(((-1,1)>100000),1,0)+if(((-1,2)>100000)
,1,0)+ if(((0,-2)>100000),1,0)+if(((0,-1)
>100000),1,0)+ if(((0,1)>100000),1,0)+
if(((0,2)>100000),1,0)+if(((1,-2)>100000)
,1,0)+if(((1,-1)>100000),1,0)+
if(((1,0)>100000),1,0)+if(((1,1)>100000),1,0)
+if(((1,2)>100000),1,0)+if(((2,-
2)>100000),1,0)+if(((2,-1)>100000),1,0)+
if(((2,0)>100000),1,0)+if(((2,1)>100000),1,0)
+if(((2,2)>100000),1,0)}{90+trunc((0,0)/10-
10000)*10}
...
```

Figure 13: Specification of the Flock of Birds Model

We modeled the behavior of an individual bird, based on the following behavior rules (Reynolds Craig 1987):

- Collision avoidance with nearby flock mates.
- Attempt to match velocity with nearby mates.
- Attempt to stay close to nearby flock mates

The birds fly in certain direction at a certain speed. Their field of vision is 300°, but they only have good forward sight. Based on these rules we built a Cell-DEVS model to simulate the birds' fly. Each cell of the model represents a space of 4 m^2. The cell state codification represents the direction of the bird (1:NW; 2:N; 3:NE; 4:W; 6:E; 7:SW; 8:S; 9:SE) and its speed. For example, the cell value 10004 represents a bird flying west (unit value equal to 4) at 1 second/cell. In order to avoid collision, when two or more birds want to move to the same place, they change their direction at random.

Figure 14 shows the execution of the model using CD++: when one bird sees the other, they start flying together.

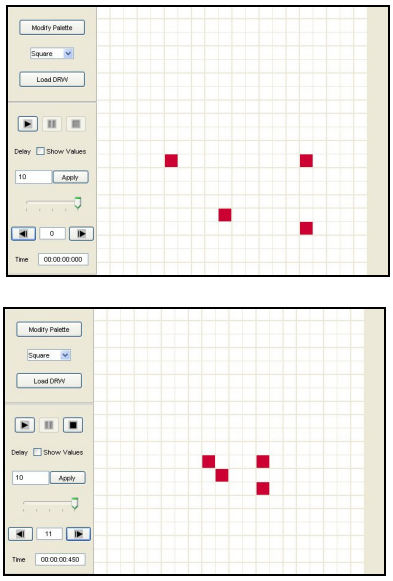


Figure 14: Joining Behavior (a) Four Birds Flying Isolated; (b) Birds Flying Together

In (Ameghino, Glinsky, and Wainer 2003) we presented a model of the reproduction of the Vibrio Parahaemolyticus bacterium, a marine germ that lives in the coast and estuaries. Its reproduction takes place at 15ºC either in the skin or the intestine of a fish. To survive, the bacteria need temperatures ranging from 15ºC to 43ºC. They need between 20 and 30 minutes to reproduce, however they cannot do it below 8ºC. If the temperature is close to 8ºC, reproduction takes longer. Bacteria are destroyed when exposed to temperatures higher than 60ºC over 10 minutes.

We couple a DEVS component to introduce temperature changes between -10ºC and 0ºC. Figure 15 shows the specification of the model in CD++. We first declare *Coldgenerator*, a DEVS model that generates cold temperatures using an exponential distribution function with the specified parameters. The Cell-DEVS model *contam* is defined including size, neighborhood shape, type of delay, and borders. In this case, we also define the input ports and connections with *Coldgenerator*.

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 temperature is below 8ºC for 10s, bacterium does not reproduce.

2. If the temperature is within 8ºC and 60ºC during 30s, then bacteria reproduce.

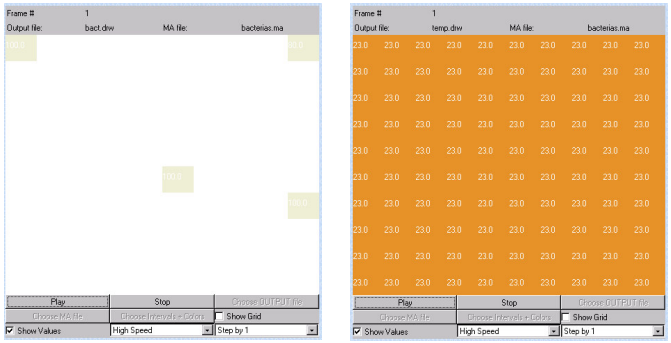
3. If the temperature is above 60ºC during of 10s, the bacteria die.

We use inertial cell delay and define that a cell reaching the concentration of 100 germs begins infecting the neighboring cells. The *Temperature* section represents the local computing function for the behavioral temperature model. The *Evolution* rules describe the bacterium behavior. The *setCold* section states the range of temperatures generated by the DEVS component.

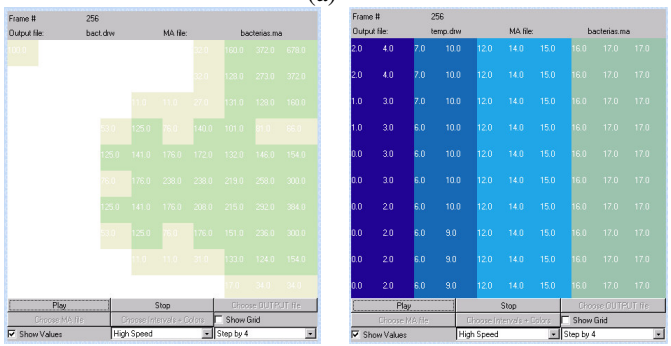
```
[top]
components : contam Coldgenerator@Generator
link : out@Coldgenerator inputCold@contam

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

[contam]
dim : (10, 10, 2)     border : nowrapped
delay : inertial   localtransition: Evolution
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@contam(0,0,1)
portInTransition : in@contam(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  {  cell-
pos(2)=0 and (0,0,1)>8 and statecount(?)=10 }
…
rule: {(0,0,0)} 10000 { cellpos(2) = 0 }
```

Figure 15: Specification of the Bacteria Model

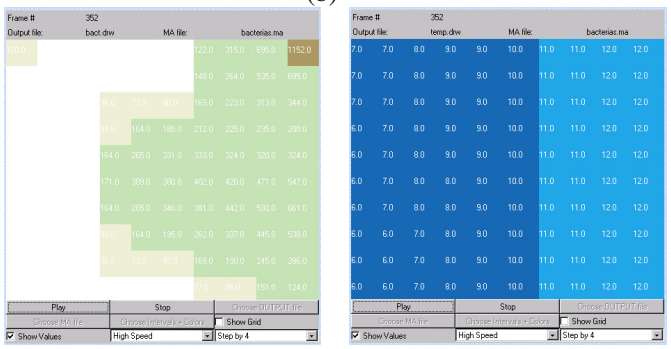
Figure 16 illustrates the results obtained when this model is executed, showing the evolution over the surface of a fish for 4 hours. The left side represents the bacteria concentration (white areas represent absence of bacteria; darker shades represent higher concentrations). The right side represents the temperatures of the surface (darker is colder).



(a)



(b)



(c)

Figure 16: Results of Bacteria Propagation: (a) Initial Concentration; (b) After 1.5 hours; (c) After 4 hours

The following example, also introduced in (Ameghino, Glinsky, and Wainer 2003), represents the behavior of ants following a specific path from an anthill to a source of nourishment. When an ant finds food, it returns to the anthill leaving a hormone (pheromone) on its path; the others use this as a signal leading to the source of food. To avoid collisions, if two or more ants want to move to the same place, they all stay in their positions and change the direction at random until one of them actually moves. When an ant finds food, it changes its course and follows the pheromone path to return to the anthill. In the case that there is no pheromone, the ant moves at random, seeking the anthill or another pheromone path. The example here presented assumes that each cell in the Cell-DEVS space represents a section of soil, whose state can be one of the following: pheromone; ant seeking; ant following pheromone; food; or ant returning to the anthill with food.

Figure 17 describes the model specification. We define the dimensions of the cell space, neighborhood and the rules that define the behavior of an ant. We use different macro definitions to avoid long statements in the specification. 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. Figure 18 shows the simulation results.

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

Figure 17: Specification of the Ants Model

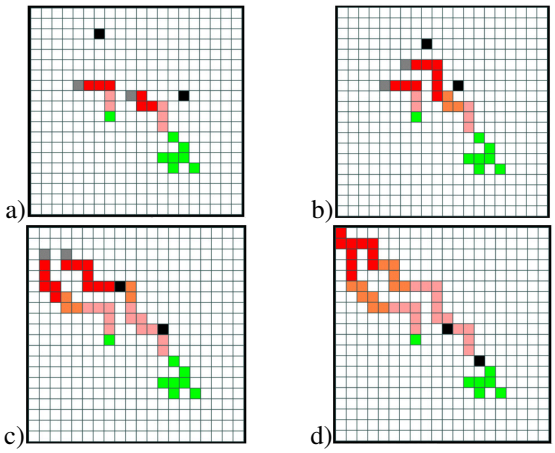


Figure 18: Two Ants: a) Returning; b) Seeking Food; c) Found Pheromone; d) Get to the Anthill

The figure 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 figure, and different gray colors represent the concentration of pheromone showing the way to the food.

Our last ecological model, originally presented in (Ameghino, Troccoli, and Wainer 2001), represents the behavior of forest fires under different environmental conditions. This Cell-DEVS model has been built using a well-known model for fire propagation in forests is due to Rothermel (Rothermel 1972), which computes the ratio of spread and intensity of fire based on environmental and vegetation conditions. Three parameter groups determine the fire spread ratio: a) vegetation type (caloric content, mineral content and density); b) fuel properties (the vegetation is classified according to its size); and c) environmental parameters (wind speed, fuel humidity and field slope).

Our first step was to use a fuel model, the speed and direction of the wind, the terrain topology and the dimensions of a region to obtain the spread ratio in every direction (fuel model group number 9, a SE wind of 24.135 km/h and a cell size of 15.24 x 15.24 m).

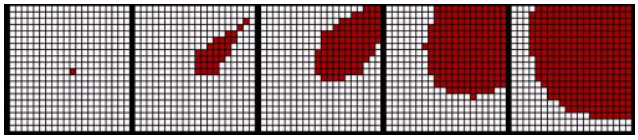
Figure 19 shows a 20x20 Cell-DEVS representing the terrain and vegetation. In this case, the state variables use a value 0 to indicate the absence of fire and a value different to 0 to indicate the time when fire has started.

```
[ForestFire]
dim: (20,20)   localtransition : FireBehavior
border: nowrapped delay : inertial
neighbors  :  (-1,-1)  (-1,0)  (-1,1)  (0,-1)
(0,0)  (0,1) (1,-1)  (1,0)  (1,1)

[FireBehavior]
rule:{(1,-1)+(21.5526/17.9671)}     {(21.5526/
17.9671)*60000} {(0,0)=0 and 0<(1,-1)}
rule:{(1,0)+(15.24/5.1069)} {(15.24/
5.1069)*60000} {(0,0)=0 and 0<(1,0)}
rule:   {(0,-1)+(15.24/5.1069)}   {(15.24   /
5.1069)*60000} {(0,0)=0 and 0<(0,-1)}
rule:{(-1,-1)+(21.5526/1.8720)}   {(21.5526   /
1.8720)*60000} {(0,0)=0 and 0<(-1,-1)}
rule:{(1,1)+(21.5526/1.8720)}        {(21.5526
/1.8720)*60000} {(0,0)=0 and 0<(1,1)}
rule: {(-1,0)+(15.24/1.1460)} {(15.24/1.1460)
*60000} {(0,0)=0 and 0<(-1,0)}
rule:{(0,1)+(15.24/1.1460)} {(15.24 / 1.1460)
*60000} {(0,0)=0 and 0<(0,1)}
rule:{(-1,1)+(21.5526/0.9874)}   {(21.5526   /
0.9874)*60000} {(0,0)=0 and 0<(-1,1)}
rule : {(0,0)} 0 { t }
```

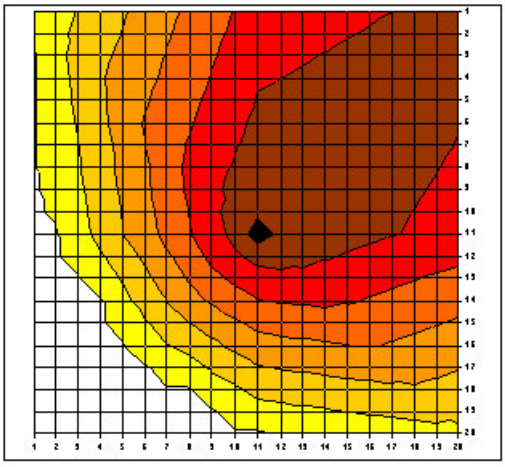
Figure 19: Definition of a Fire Forest model

The rules defining the behavior of the local computing function are devoted to detect the presence of fire in the eight neighboring cells. If there is fire in one, the cell will burn. For instance, the first rule checks if the current cell is not burning (*(0,0)=0*) and if the SW neighbor has started to burn (*0<(1,-1)*). If this condition holds, the value will be (1,-1)+(21.5526/17.9671), which is the time to spread the fire in the cell. As the spread ratio is 17.9671 mpm and a cell has a diagonal of 21.5526 m, it will take (21.5526/17.9671) minutes for the fire to reach the a cell once it has started in its SW neighbor. Therefore, we use a delay of (21.5526/17.9671)*60000 ms after which the present cell state will spread to the neighbors.

The results of the execution of this model are presented in Figure 20. As we can see, the burning time of a cell depends on the spread ratio in the direction of the burning cell. This value is used as the delay component for the rules. It is important to notice that the cells are updated at different times, as set by a rule's delay component. This is a clear departure from the classical approach of CA, where all active cells are updated at the same time. A non-burning cell in the direction of the fire spread will be updated in a shorter period than a non-burning cell in the opposite direction. Another advantage is that expressing a timing delay is done naturally.



(a)



(b)

Figure 20: (a) Fire Propagation Results; (b) A Two-our Period (each zone represents 20 minutes)

5 MODELS OF ARTIFICIAL SYSTEMS

We have defined a number of models of artificial systems, some of which will be introduced in this section. The first example presented previously in (Lam and Wainer 2003), is used to solve path planning on a maze. The algorithm effectively blocks off every dead-end path in the maze,

making every free cell that is accessible from only one direction (i.e. three wall cells around it) a dead end and therefore not part of the solution. These cells become new wall cells, and this procedures is repeated until the system remains in a steady state. In this state, the only remaining free cells represent the solution(s) to the maze. If there is no solution, the entire array of cells will be wall cells. These rules were translated into the coupled model definition presented in Figure 21.

```
[maze]
dim : (20, 20)       delay : transport
border : nowrapped
neighbors : (-1,0) (0,-1) (0,0) (0,1) (1,0)
localtransition : maze-rule

[maze-rule]
rule : 1 100 { (0,0)=0 and (truecount=3 or
                            truecount=4) }
rule : 0 100 { (0,0)=0 and truecount<3 }
rule : 1 100 { t }
```

Figure 21: Maze-solving Specification in CD++

The results are showed in Figure 22, which include the graphical displays of a maze with a given initial state.

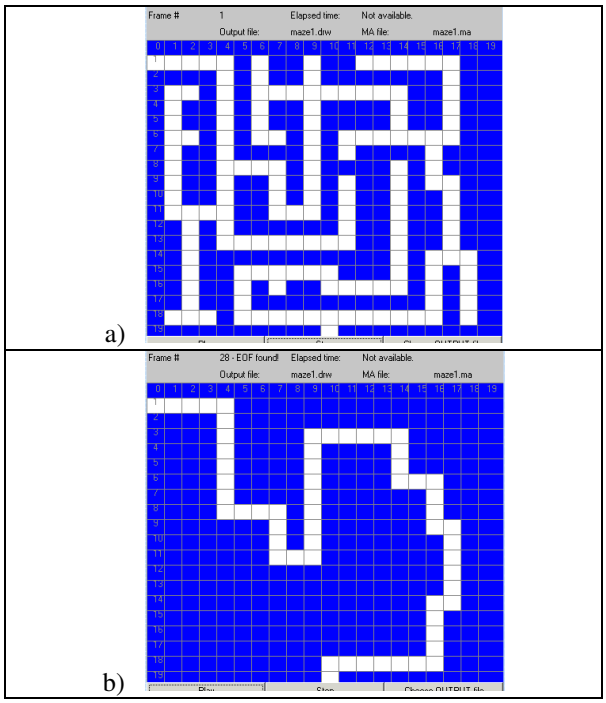


Figure 22. a) Original Maze; b) After Processing

Our next example presents the movement of robots using predefined paths in an industrial plant. Robots are used to carry a load from the source point where it is produced, to a destination point where it is consumed. The robots can move N-S-E-W, following predefined routes at different speeds. There may be more than one robot on each route. A robot stops when it detects a nearby robot on the same route. In addition, routes can have crossing points, so there is a potential risk for collisions. The plant is represented by a 20x20 Cell-DEVS. This cellular model is linked to four different DEVS models, each devoted to generate a load at the source points (12, 19), (0,10), (9,0) and (19,6).

This coupled model, presented in Figure 23 contains 5 components: *Floor* (a Cell-DEVS) and *Source1-Source4* (DEVS random generators). Then, the model's coupling is defined (generators' output ports are connected to *Floor* input ports). Finally, we define the Cell-DEVS *Floor* coupled model parameters (size, borders, delay, etc.). In this example we show how to react to the external events received: the input ports *in1* to *in3* are coupled to the cell space; events arriving on port *in1* should be sent to the *in* port of cell (12,19).

```
[top]
components:       Floor       Source1@Generator
Source2@Generator Source3@Generator
link : out@Source1 in1@Floor
link : out@Source2 in2@Floor
link : out@Source3 in3@Floor

[Floor]
dim : (20,20)   localtransition : RobotsMov
delay : inertial   border : nowrapped
neighbors : (-1,0) (0,-1) (0,0) (0,1) (1,0)
in : in1 in2 in3 in4
link : in1 in@Floor(12,19)
link : in2 in@Floor(0,10)
link : in3 in@Floor(9,0)

[RobotsMov]
% ------ Robot 1 -----------------------------
rule : 10 1000 {(0,1)=1 and (0,0)=0 and
                                cellpos(1)!=4}
rule : 11 1000 {(0,1)=1 and (0,0)=0 and
                                cellpos(1)=4}
rule : 0 0 {(0,-1)=10 and (0,0)=1}
rule : 0 0 {(0,-1)=11 and (0,0)=1}
rule : 2 0 {(0,0)=11}
rule : 1 0 {(0,0)=10}

rule : 20 2000 {(-1,0)=2 and (0,0)=0 and
                                cellpos(0)!=17}
rule : 21 2000 {(-1,0)=2 and (0,0)=0 and
                                cellpos(0)=17}
rule : 0 0 {(1,0)=20 and (0,0)=2}
rule : 0 0 {(1,0)=21 and (0,0)=2}
rule : 2 0 { (0,0)=20 }
rule : 1 0 { (0,0)=21 }
% ------ Robot 2 -----------------------------
...
```

Figure 23. Model Definition for Robot Routes

We also included a part of the cell behavior for the Cell-DEVS model. In this case, a zero value is used if the cell is empty. A value different from zero will indicate the presence of a robot. A cell containing a route 1 robot can have the values 1, 10 or 11 if the robot is moving horizontally and 2, 20 or 21 if the robot is moving vertically. The *cellpos()* function is used to see if the robot is on the path, defining the predefined movement on the floor. The same applies for cell containing robots belonging to other routes.

A robot movement is done in three steps. For example, a route 1 robot at the source is indicated by a 1 in cell (12,19). This value says the robot is ready to move horizontally. The next cell on the route will receive a neighbor change event indicating that cell (12,19) has just changed to 1. Then, cell will get ready to receive the robot by acquiring a value of 10 or 11 after a delay of 1000 ms (step 1). The value 10 will be used if the robot continues horizontally and 11 if the robot must turn. Once this change is produced, the original cell that had a value of 1 will now change to 0 (step 2) indicating the robot is not longer present and the cell that had the value 10 or 11 will change to 1 or 2, respectively (step 3). The value of 1 will again indicate the presence of a robot that is about to move horizontally and the value 2 a robot that is about to move vertically. The collisions are avoided by only allowing step 1 to take place if the destination cell is empty, as expressed a condition statement.

Figure 24 shows different robots running at different speeds (according with their delays). The figure also shows the collision avoidance between two robots.

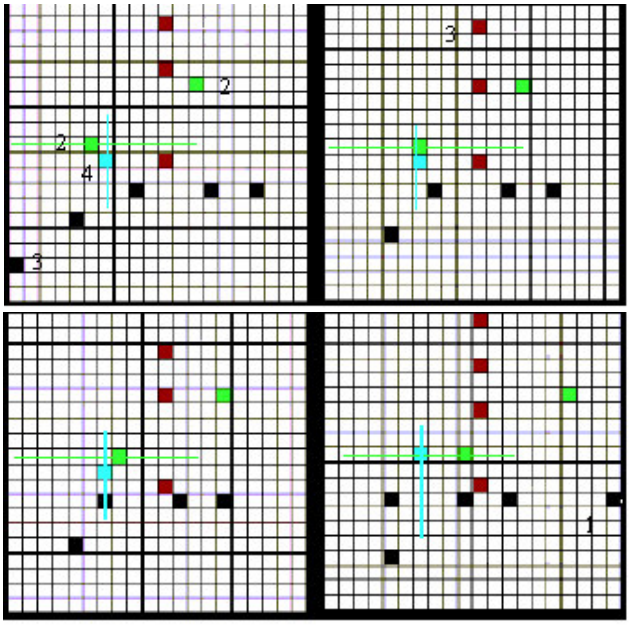


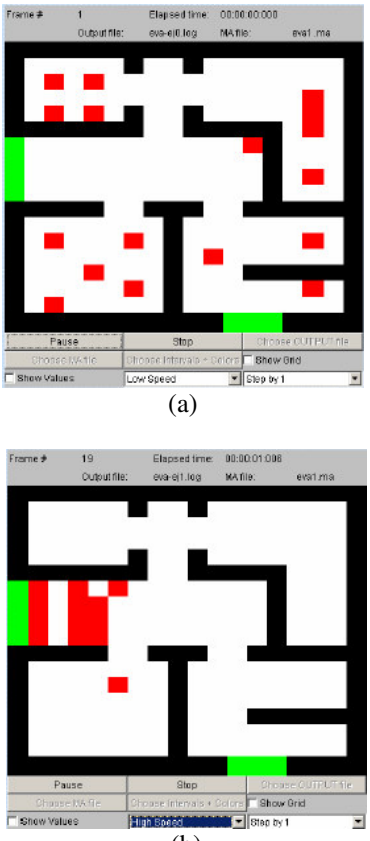
Figure 24: Executing the Robots Model (two robots reaching an intersection point)

Our last set of models, presented in (Ameghino, Glinsky, and Wainer 2003; Ameghino and Wainer 2004) is devoted to simulate evacuation processes. The model represents people moving through a room or group of rooms, trying to gather their belongings or related persons and to get out through an exit door. The goal is to understand where the bottlenecks can occur, and which solutions are effective to prevent congestion.

The basic idea was to simulate the behavior and movement of every single person involved in the evacuation process. A Cell-DEVS model was chosen with a minimum set of rules to characterize a person's behavior:

- A person, in normal state, goes to the most nearby exit.
- A person in panic goes in opposite direction.
- People move at different speeds.
- If the way is blocked, the person can decide to move away and look for another path.

Figure 25 shows the simulation results of this model. The gray cells represent people who want to escape using the exit doors. The black cells represent walls. Note that the leftmost part in the figure shows people waiting in the exit door.



(a)

(b)

Figure 25: (a) People Seeking an Exit. (b) After 15 seconds, People Found the Exit

We used two planes, one for the floor plan of the structure and people moving, and the other for orientation. Each cell in the grid is 0.4 m^2 (one person/cell). The orientation layer (Figure 26) contains information that serves to guide persons towards emergency exits. We assigned a potential distance to an exit to every cell of this layer.

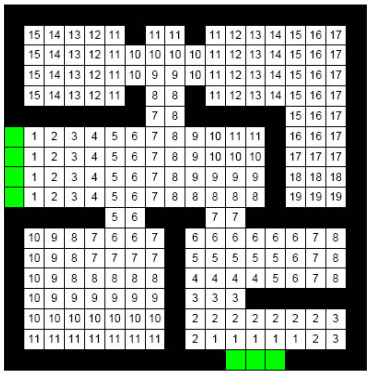


Figure 26: Cell-DEVS Layer used for Orientation

Figure 27 (Ameghino, Glinsky, and Wainer 2003) shows the execution of the same model in the context of the movement of persons waiting for subways in a subway station. The following figure resembles 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.

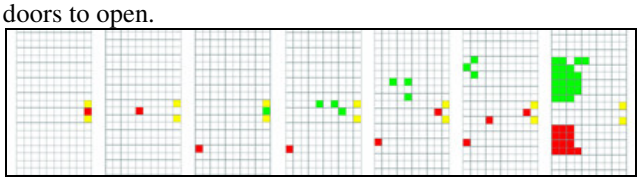


Figure 27: Execution Results of Metro Station Model

Figure 28 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.

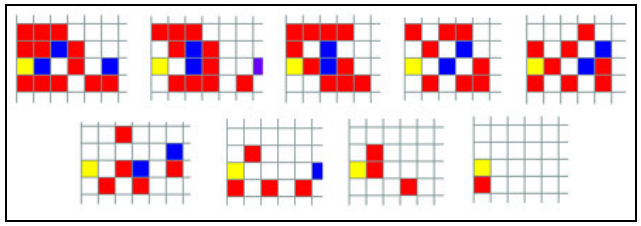


Figure 28: People Getting In and Out Using Door A

CONCLUSION

Cell–DEVS allows describing physical systems using an n-dimensional cell-based formalism. Input/output port definitions allow defining multiple interconnection 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, based on the formalism permits defining complex cell-shaped models using a high-level specification language.

We showed that different kinds of applications can be easily developed, allowing the study of complex problems through simulation, which, otherwise, could not be attacked. Finally, the use of a formal base improves the development, checking and maintaining phases, facilitating the testing and reuse of their components.

The discrete event nature of the formalism provides better precision and performance, due to the independent timing for each cell. If a cell state does not change, it is deactivated up to the arrival of a new external event, thus, improving CPU use without needing small time slots.

The tool and the examples are the public domain and they can be obtained in:
<http://www.sce.carleton.ca/faculty/wainer/>

ACKNOWLEDGMENTS

This work was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canadian Foundation for Innovation (CFI).

REFERENCES

Ameghino, J., Glinsky, E., and Wainer, G. 2003. Applying Cell-DEVS in Models of Complex Systems. In *Proceedings of Summer Simulation Multiconference*. Montreal, Quebec. Canada.

Ameghino, J., Troccoli, A., and Wainer, G. 2001. Modeling and simulation of complex physical systems using Cell-DEVS. In *Proceedings of the 33rd SCS Summer Computer Simulation Conference*. Seattle, Washington. USA.

Ameghino, J., and Wainer, G. 2000. Application of the Cell-DEVS paradigm using CD++. In *Proceedings of the 32nd SCS Summer Computer Simulation Conference*. Vancouver, Canada.

Ameghino, J., and Wainer, G. 2004. Using Cell-DEVS for modeling complex cell spaces. Internal Report; Department of Systems and Computer Engineering. Carleton University. Submitted for publication.

Gardner, M. 1970. The fantastic combinations of John Conway's New Solitaire Game 'Life'. *Scientific American*. 23 (4). pp. 120-123.

Lam, K., and Wainer, G. 2003. Modeling of maze-solving problems using Cell-DEVS. In *Proceedings of the 2003 SCS Summer Computer Simulation Conference*. Montreal, Quebec. Canada.

Lo Tártaro, M., Torres, C., and Wainer, G. 2001. Defining models of urban traffic using the TSC tool. In *Proceedings of 2001 Winter Simulation Conference*. B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds. pp. 1056-1063. Arlington, Virginia. USA.

MacSween, P., and Wainer, G. 2004. On the Construction of Complex Models Using Reusable Components. In *Proceedings of SISO Spring Simulation Interoperability Workshop*. Arlington, Virginia. USA.

Muzy, A., Wainer, G., Innocenti, E., Aiello, A., and Santucci, J.F. 2002. Cell-DEVS quantization techniques in a Fire Spreading application. In *Proceedings of 2002 Winter Simulation Conference*. E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds. pp. 542-549. San Diego, CA. USA.

Press, W.H., Flannery B.P., Teukolsky, S.A., and Vetterling, W.T. 1986. *Numerical Recipes*. Cambridge University Press, Cambridge.

Reynolds Craig, W. 1987. Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graphics.* 21(4), pp. 25-34.

Rothermel, R. 1972. A mathematical model for predicting fire spread in wildland fuels. Research Paper INT-115. Ogden, UT: U.S. Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station.

Toffoli, T. 1994. Occam, Turing, von Neumann, Jaynes: How much can you get for how little? (A conceptual introduction to cellular automata). In *Proceedings of the International conference on Cellular Automata for Research and Industry '94*. Rende, Italy.

Troccoli, A., Ameghino, J., Iñón, F., and Wainer, G. 2002. A flow injection model using Cell-DEVS. In *Proceedings of the 35th IEEE/SCS Annual Simulation Symposium.* San Diego, CA. U.S.A.

Wainer, G. 2002. CD++: a toolkit to develop DEVS models. *Software - Practice and Experience*. 32, pp. 1261-1306.

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

Wolfram, S. 2002. *A new kind of science.* Wolfram Media, Inc.

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

AUTHOR BIOGRAPHY

GABRIEL A. WAINER received a M. Sc. (1993) and Ph.D. degree (1998, highest honors) of the Universidad de Buenos Aires (UBA), Argentina, and Université d'Aix-Marseille III, France. He is Assistant Professor at the SCE Dept., Carleton University (2000-). He was Assistant Professor at the Computer Sciences Dept. (UBA, 1997-2000), and a Visiting Research Scholar at ACIMS (University of Arizona) and LSIS (CNRS, France). He published over 80 articles on simulation and real-time systems. He is author of a book on real-time systems, and one on discrete-event simulation. He is Associate Editor of the Transactions of the Society for Computer Simulation (SCS). He was PI of several research projects (NSERC, Precarn IRIS, IBM Scholars, Usenix, CFI, CONICET, ANPCYT). He was member of the IPC of more than 30 conferences, and a reviewer for different journals and research agencies. Prof. Wainer is a member of the Board of Directors and Chair of the Standards Committee of the SCS, and a chair of the SISO DEVS standardization Study Group. He is also a Associate Director of the Ottawa McLeod Institute of Simulation Sciences, and chair of the Ottawa M&SNet. His current research interests are related with modeling methodologies and tools, parallel/distributed simulation and real-time systems. His e-mail address is <gwainer@sce.carleton.ca>, and his web address is <www.sce.carleton.ca/faculty/wainer>.