NEW EXTENSIONS TO THE CD++ TOOL

Daniel A. Rodríguez and Gabriel A. Wainer
Departamento de Computación
Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires
(1428) Pabellón I. Ciudad Universitaria. Buenos Aires. Argentina.
e-mail:{drodrigu,gabrielw}@dc.uba.ar

Keywords: Modeling methodology: DEVS models, cellular automata, Cell-DEVS models, Simulation methods: Discrete-event simulation, Simulation tools.

ABSTRACT

This work describes some of the extensions included into a tool used to study, model and simulate cellular models. The environment is based on the Cell-DEVS paradigms. The main extensions are devoted to define generic cell spaces, and are based on the formal definitions for n-dimensional Cell-DEVS models. A cell specification language used to define the model's behavior was redefined to include these extensions. In this way, very complex cell based systems can be built in a simple fashion, allowing reductions in the development, checking and maintenance times of the components.

INTRODUCTION

The **DEVS** (Discrete EVents Systems specifications) formalism was proposed in (Zeigler 1976), with the goal of modeling complex discrete events models. **Cell-DEVS** (Wainer and Giambiasi 1998; Wainer 1998) is a paradigm that extended the DEVS formalism to improve the definition of cellular models. In this paradigm, each cell is defined as an atomic model using transport or inertial delays (Giambiasi and Miara 1976). A coupled model that includes a group of these cells constitutes a cellular model.

Cell-DEVS atomic models can be specified as

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

X is the set of external input events;
Y is the set of external output events;
I represents the model's modular interface;
S is the set of sequential states for the cell;
q is the cell state definition;
N is the set of states for the input events;
d is the transport delay for the cell;
d_{int} is the internal transition function;
d_{ext} is the external transition function;
t is the local computation function;
l is the output function; and
D is the state's duration function.

A Cell-DEVS coupled model is defined by:

$$GCC = < Xlist, Ylist, I, X, Y, n, \{t_1,...,t_n\}, N, C, B, Z, select >$$

Ylist is the output coupling list;
Xlist is the input coupling list;
I represents the definition of the interface for the modular model;
X is the set of external input events;
Y is the set of external output events;
n is the dimension of the cell space;
$\{t_1,...,t_n\}$ is the number of cells in each of the dimensions;
N is the neighborhood set;
C is the cell space;
B is the set of border cells;
Z is the translation function; and
select is the tie-breaking function for simultaneous events.

CD++ (Barylko *et al.* 1998) is a tool that allows implementing the theoretical concepts specified by the DEVS and Cell-DEVS formalisms. A specification language permits the creation of coupled models, the initial configuration for the atomic models, and the creation of external events to be used during the simulation. The original version of CD++ permits the creation of two-dimensional cellular automata, where the state of a cell has a binary or three-state value.

The goal of this work is to present a new set of extensions done to the CD++ tool. The extensions done are based on the theoretical concepts defined in (Wainer 1998). That work presented a formal definition for n-dimensional Cell-DEVS models, providing a sound base to develop cellular models.

CD++

CD++ is a modeling tool that was defined using the basic concepts defined in (Zeigler 1984; Zeigler 1990; Wainer 1998). Two basic abstract classes were defined: *Models* and *Processors*. The first are used to represent the behavior of the atomic and coupled models, while the second implement the simulation mechanisms.

The *Atomic* class implements the behavior of the atomic models. The *Coupled-Model* class implements the mechanisms of the coupled models. For the case of a cellular model, a special atomic model is used to represent to each cell. To do so, *AtomicCell* and *CoupledCell* are defined as subclasses of *Atomic* and *Coupled* respectively. *AtomicCell* extends the behavior of the atomic models, to define the functionality of the cell space. On the other hand, the *CoupledCell* class permits the management of a group of atomic cells.

The simulation is based on the interchange of messages between the different processors. Each message contains information to identify the sender/receiver, the time of the event, and the content that consists in a port and a value for it. Different messages are used: *X* (which represents an external event), *Y* (represents the model output), * (represents an internal event), and *done* (indicating that the model has finished its task).

This work was partially supported by the UBACYT research project TX-004, “Concurrency in distributed systems”.

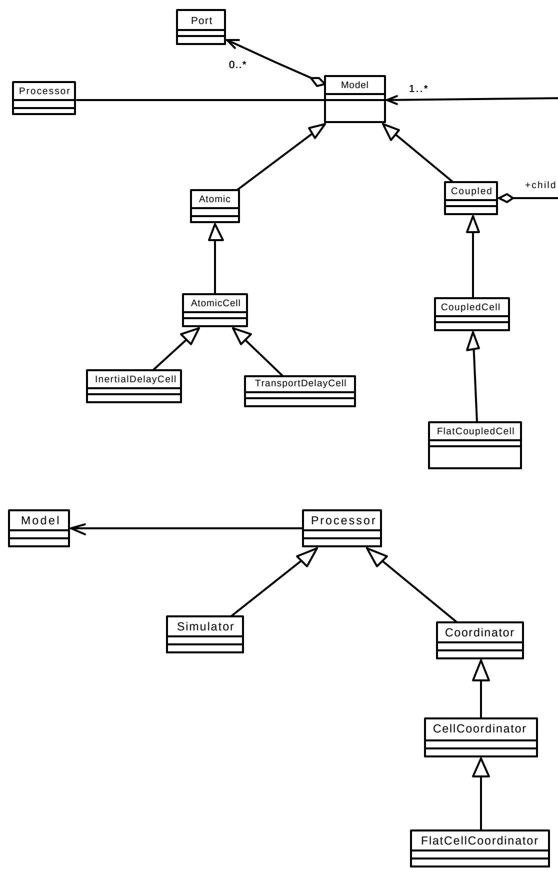

Figure 1. Cell-DEVS Models and Processors.

EXTENDED CD++

The CD++ tool includes an interpreter for a specification language that allows describing the behavior of each cell of a cellular model, including its delay and neighborhood. In addition, it allows to define the size of the cell space and their connection with other DEVS models, the border and the initial state of each cell. To do so, the theoretical definitions of the Cell-DEVS formalism were used.

The behavior specification for a cell is defined using a set of rules, each indicating the value for the cell's state if a condition is satisfied. The output of the model should be delayed by using a specified time. If the condition is not valid, the next rules in the list are evaluated until a rule is satisfied or there are no more rules.

In the latter case, an error will be raised, aborting the simulation process. This error indicates that the model specification is incomplete. The existence of two or more rules with same condition but with different state value or delay is also detected, avoiding the creation of ambiguous models. In this situation, the simulation will be aborted. Instead, when two different rules are valid, but their result is the same, a warning is raised but the simulation continues.

The original tool used three-state, based on the work presented in (Wainer and Giambiasi 1998). When the conditions of a rule are evaluated, a *True*, *False* or *Undefined* value can be obtained. In this case, if no condition is *True* but any is evaluated to *Undefined*, the state of the cell will be *Undefined*, and the default delay. In this situation, the tool alerts to the modeler, but the simulation is not interrupted.

Taking this basic behavior as a basis, several modifications were introduced. Some details will be presented in the following sections.

Extensions to the Specification Language

The use of three-state logic constrained the models that can be implemented, limiting the use of the tool in more general problems. Hence, the definitions of (Wainer 1998) were considered, allowing a cell to have a value in the set $\boldsymbol{R} \cup \{Undefined\}$.

The expansion of the set of values implied that the specification language should be modified. The complete grammar for the new language can be seen in (Rodríguez and Wainer 1999). The main operators available include:

- Boolean;
- Comparison;
- Arithmetic;
- Number types;
- Neighborhood values;
- Time;
- Conditionals;
- Angle conversion;
- Pseudo-random numbers;
- Error rounding;
- Predefined constants: pi, e, grav, accel, light, planck, etc.

In every case, it was considered that any expression including the *Undefined* value would be *Undefined*. Due to the inclusion of probabilistic functions as conditions into the rules, it is possible to obtain a *False* result. In this case, even the model can be well specified, the condition would show an ambiguity. The tool automatically identifies this situation and assigns the *Undefined* value to the cell, informing this situation.

External Event arrival

The original version of CD++ followed a previous specification for external event management. In this case, when a cell is created, three ports are associated: *In*, *Out* and *NeighborChange*. The input port *In* connects external DEVS models with the cell. The input port *NeighborChange* define the input values arriving from the neighbors. Finally, the *Out* port connects the cell with the neighbors and other DEVS models. When an external message arrives to the cell through the *In* port, its value is queued, and it will be used to compute the new cell state.

The implementation has been modified to reflect the present specifications for Cell-DEVS models. First, the input ports *In* are created only for those cells connected with external DEVS models. On the other hand, the local computing function τ uses all the inputs, including the values sent by the neighboring cells, and the external messages arrived through the other input ports. The language was extended to define the actions to be taken for messages arriving from external models.

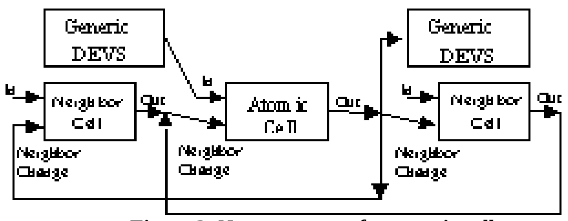

Figure 2. New structure of an atomic cell.

This mechanism is implemented by associating each cell with a list of tuples (*portIn*, *inFunction*, *lastValue*). *PortIn* contains the port name. *InFunction* identifies the function describing the behavior for a message arriving from *portIn*. Finally, *lastValue* contains the last value arrived to the port.

Definition of output behavior

As mentioned in the previous section, N-CD++ creates the *In* ports in a dynamic fashion, and they can be used to connect the cell space with other DEVS models.

In CD++, the *Out* port connects each cell with the neighbors and with other DEVS models. The behavior was redefined in N-CD++, to follow the formal definitions of (Wainer 1998). In this case, several output ports can exist. The first one is automatically created, and it is called *Out*. It is used to connect the present cell with the neighbors. The *Out* port can also be used to connect a cell with other DEVS models, that will receive a new message each time the cell changes its internal state.

The rest of the output ports are created dynamically only for the cells that will send state values for other DEVS models, and their name can be defined by the modeler. Therefore, the new scheme for the port connection in each cell is the following:

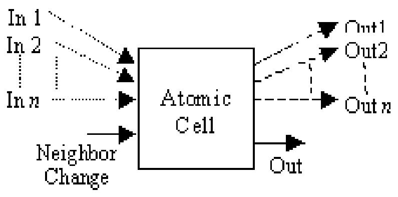

Figure 3. New structure for an Atomic cell.

Under certain situations, it is useful that a model can receive values different that those of the cell state. To permit this behavior, the **send** function was defined. It allows to specify the value to send through a port in a given moment. For instance, the rule:

{ new_value + **send**(P, V) } delay { condition }

says that if the *condition* is true, the new value for the cell will be the specified by the rule, and the V value will be sent through the P port. The cell is delayed as in the previous cases, using the kind of delay specified. The V value can be a constant or a complex expression to be evaluated at runtime.

Send can be defined as: **send**(string, real) $\rightarrow$ 0, that is, the function always returns a 0 value. The idea is that the use of this function allows to send a value without change the value of the cell, as it can be seen following:

{ (0,0) + **send**(port1, 15 * log(10)) } 100 { (0,0) > 10 }

The meaning of this expression is that, if the value of the present cell is greater than 10, the cell keeps the present value. It outputs the value of the expression 15*log(10) through the *port1*, and delays the output during 100 time units. Now, there are several output ports. Hence, when the internal transition function changes the cell state, the value is sent through the *Out* port. The remaining ports are reserved explicitly for values using the **send** function.

The following modification of the Life game, is used to show several of the changes done to the tool:

```
[top]
components : life
in : in
out : outG1 outG2
link : out1@life outG1
link : out2@life outG2
link : in in@life

[life]
type : cell
width : 2
height : 2
delay : transport
defaultDelayTime : 1
border : wrapped
neighbors : life(-1,-1) life(-1,0) life(-1,1)
neighbors : life(0,-1)  life(0,0)  life(0,1)
neighbors : life(1,-1)  life(1,0)  life(1,1)
initialvalue : 0
in : in
out : out1 out2
link : in in@life(1,1)
link : output1@life(1,1) out1
link : output2@life(1,1) out2
portInTransition : in@life(1,1)  specialRule
localtransition : nothing-rule
zone : generateOut { (1,1) }

[nothing-rule]
rule : { (0,0) } 1 { t }

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

[generateOut]
rule:{(0,0)+send(output1,9.9999)} 1 {(0,0)>=10}
rule :{(0,0)+send(output2,3.3333)} 1 {(0,0)<10}
```

Figure 4. Example of use of the new features.

The cellular model here defined uses the interconnection pattern shown in the following figure (the ports used to connect neighbors are not showed here).

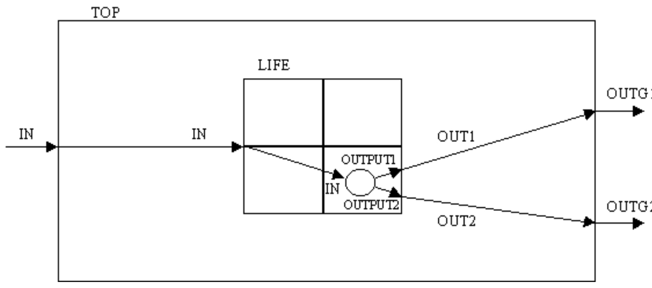

Figure 5. Coupling scheme of the previous example.

As it can be seen, the higher level coupled model (top) uses one input and two output ports. Only one component is included (life), using a 2x2 cell space. The *Link* directive is used to create a coupling between the *portOut* and *portIn* ports for each of the models involved. After, the basic parameters for the model are defined: dimension, delay, border, neighbors and external coupling. Each of these constructs is mapped into one of the sets defined by the Cell-DEVS formalism.

The *portInTransition* directive allows to define the name of the function to be used when a message arrives through the specified port. In this case, the *specialRule* is executed. The function *portValue*(*x*) allows to get the value of the last message arrived through the port *x* of the cell. Besides, the instruction *thisPort* returns the name of the port where the message has arrived.

The local computing function, called *nothingRule*, does nothing. There is a special zone (*generateOut*), used to define a couple of cells that will be used to output values using the send function.

Support for N-dimensional models

The real systems that can be studied using cellular models are usually represented by using models in two or three dimensions. Several theoretical problems can be defined as cellular models with four or more dimensions. The original version of the tool only allowed to define two-dimensional cell spaces. This constraint reduces the use of the tool in more general problems. Therefore, the main modification done to the tool was devoted to allow the definition of n-dimensional models. In this case, the formal specification of (Wainer 1998) was again used to have a formal basis to specify the model behavior.

CD++ was implemented by storing the cells states in a two-dimensional array of $d_1 \cdot d_2$, where the element (x_1, x_2), $x_i \in [0, d_i -1]$, is in the position $x_1 + x_2 \cdot d_1$. In an analogous fashion, N-CD++ uses an array of $\prod_{i=1...n} d_i$ to store the states for the cellular automata with dimension $(d_1, d_2, ..., d_n)$, and in this case $(x_1, x_2, ..., x_n)$ occupies the position $\sum_{i=1...n} x_i \cdot (\prod_{k=1...i-1} d_k)$.

As it was seen in the previous section, each cellular model is specified by using a language that allows to define its dimension, shape of the neighborhood, and initial values for the cells. The language was adapted to include references to cells in n-dimensional cell spaces.

Another modification is related with the homogeneity of the cellular models. In CD++, the neighborhood for a cell only can be defined by adjacent cells. In addition, the shape of the neighborhood is the same in all the cell space. At present, N-CD++ allows to define a neighborhood that can be defined independently for each of the cells in the space.

Another useful change allows to define different *zones* in the cell space. Each zone is defined by a set of cells determined by the cell range $\{(x_1, x_2, ..., x_n)...(y_1, y_2, ..., y_n)\}$. Each zone is associated to a different set of rules. Using this capability, different zones into the same cellular model can present different behavior. Hence, a zone defined by a range of cells is defined by the set of cells $(t_1, t_2, ..., t_n)$ of the cellular automata, such that $t_i \in [\min(x_i, y_i), \max(x_i, y_i)]$ $\forall i \in [1, n]$.

APPLICATION EXAMPLES

This section is devoted to show the use of the tool through different application examples. The emphasis has been put into the new features provided and not in the applications. The first example defines a three-dimensional version for the Life game. After, a model of heat diffusion in a room will be considered.

Modification of the Life game

We will start to show the use of the tool and its main extensions by presenting a simple variation of the 'Life' game (Gardner 1970). In this case, we will consider a population of active cells represented by '1' values, distributed in an area of 7x7x3 cells. Some of them contain the '0' value, indicating the absence of life in the cell. A new being is born when the cell has over 10 living neighbors. Besides, a cell will remain alive when the neighborhood contains 8 or 10 living neighbors. Otherwise, the cell will "die". In this case, the neighborhood will have 3x3x3 cells.

The following figure shows a description for this model using the cell description language:

```
01 [top]
02 components : 3dl
03
04 [3dl]
05 type : cell
06 dim : (7,7,3)
07 delay : transport
08 defaultDelayTime  : 100
09 border : wrapped
10 neighbors: 3dl(-1,-1,-1) 3dl(-1,0,-1)
          3dl(-1,1,-1)
11 neighbors: 3dl(0,-1,-1)  3dl(0,0,-1)
          3dl(0,1,-1)
12 neighbors: 3dl(1,-1,-1)  3dl(1,0,-1)
          3dl(1,1,-1)
13 neighbors: 3dl(-1,-1,0)  3dl(-1,0,0)
          3dl(-1,1,0)
14 neighbors: 3dl(0,-1,0)   3dl(0,0,0)
          3dl(0,1,0)
15 neighbors: 3dl(1,-1,0)   3dl(1,0,0)
          3dl(1,1,0)
16 neighbors: 3dl(-1,-1,1)  3dl(-1,0,1)
          3dl(-1,1,1)
17 neighbors: 3dl(0,-1,1)   3dl(0,0,1)
          3dl(0,1,1)
18 neighbors: 3dl(1,-1,1)   3dl(1,0,1)
          3dl(1,1,1)
19 initialvalue: 0
20 initialCellsValue: 3dl.val
21 localtransition: 3dl-rule
22
23 [3dl-rule]
24 rule : 1 100 { (0,0,0) = 1 and
        (truecount = 8 or truecount = 10) }
25 rule : 1 100 { (0,0,0) = 0 and
         truecount >= 10 }
26 rule : 0 100 { t }
```

Figure 6. Description of a variation of the Life game.

The lines 1 and 2 are used to define the higher level coupled model, that, in this case, is defined by the model 3dl (Three dimensional Life). Lines 4 to 19 are used to define the dimension parameters for the cell space. The kind of delay and the shape of the neighborhood are also included. The line 20 is used to define a file containing the initial values for the cells. Line 21 defines the name of the local computing function that is used to update the cells values in each phase of the simulation. This function is defined in lines 23 to 26.

The results obtained when executing this model can be seen in the figure 7 (dividing the three-dimensional space into three planes). In this case, the execution starts with a high number of living cells, but the initial distribution makes that the population is not stable, and its number turns to be reduced. Finally, in the instant 00:00:01:000, the population is extinguished.

```
Time: 00:00:00:000
      0123456      0123456      0123456
     +-------+    +-------+    +-------+
    0|1      |   0|       |   0|1      |
    1|1 1  11|   1|11   11|   1|  111  |
    2| 1   1 |   2|   11 1|   2| 1 11  |
    3|       |   3|  1  11|   3|     11|
    4|  1  11|   4|  1 1  |   4| 1   11|
    5|  11  1|   5|   1 1 |   5| 11  1 |
    6|1  1  1|   6| 1   1 |   6| 1 11 1|
     +-------+    +-------+    +-------+
Time: 00:00:00:100
      0123456      0123456      0123456
     +-------+    +-------+    +-------+
    0| 1    1|   0|11    1|   0| 1    1|
    1|1 1   1|   1|1     1|   1|1 11  1|
    2|11  1 1|   2|1    1 |   2|11   11|
    3|    111|   3|  1 1 1|   3|    1 1|
    4|       |   4|     11|   4|       |
    5|1  111 |   5|1 111 1|   5|1  11 1|
    6|       |   6| 1     |   6| 1  1  |
     +-------+    +-------+    +-------+
Time: 00:00:00:200
      0123456      0123456      0123456
     +-------+    +-------+    +-------+
    0|1     1|   0|      1|   0|1     1|
    1| 11  1 |   1| 1   1 |   1| 11  1 |
    2|    11 |   2| 1    1|   2|       |
    3|1   1  |   3|1   11 |   3|1   11 |
    4|   1111|   4|   11  |   4|   1111|
    5|1  1   |   5|1  1  1|   5|1  1  1|
    6|11   11|   6|11   11|   6|11  111|
     +-------+    +-------+    +-------+

                 .     .     .
Time: 00:00:00:900
      0123456      0123456      0123456
     +-------+    +-------+    +-------+
    0|       |   0|       |   0|       |
    1|       |   1|       |   1|       |
    2|       |   2|       |   2|       |
    3|       |   3|       |   3|       |
    4|1    1 |   4|1    11|   4|1    1 |
    5|       |   5|       |   5|       |
    6|       |   6|       |   6|       |
     +-------+    +-------+    +-------+
```

Figure 7. Execution results for the modified Life game.

Heat Diffusion

This example considers the air into a room represented by cellular automaton. Each cell of the space contains a temperature value. In each stage of the simulation, the temperature of the cell is calculated as the average of the values of the neighborhood, whose shape can be seen in the following figure.

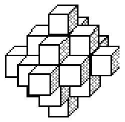

Figure 8. Neighborhood shape for the heat diffusion model

In addition, a heater is connected to the cells (2,2,1) and (3,3,0). The heater simulator generates a flow of temperatures between 24º C and 80º C with uniform distribution. On the other hand, a refrigerator is connected to the cells (1,3,3) and (3,3,2). The corresponding model is used to create values in the range [10, 15] also with uniform distribution. Both generators create values after *x* seconds, where *x* follows an exponential distribution with mean 40 seconds. The model definition done using the specification language is shown in the following figure:

```
[top]
components : room Heater@Generator
                 Cooler@Generator
link : out@Heater inputHeat@room
link : out@Cooler  inputCold@room

[room]
type : cell        dim : (4, 4, 4)
delay : transport
defaultDelayTime  : 100
border : wrapped
neighbors : room(-1,0,-1) room(0,-1,-1)
neighbors : room(0,0,-1) room(0,1,-1)
neighbors : room(1,0,-1) room(-1,1,0)
neighbors : room(-1,-1,0) room(-1,0,0)
neighbors : room(0,-1,0) room(0,0,0)
neighbors : room(1,-1,0) room(1,0,0)
neighbors : room(-1,0,1) room(0,1,0)
neighbors : room(0,-1,1) room(0,0,1)
               room(0,1,1)
neighbors : room(1,0,1) room(1,1,0)
neighbors : room(0,0,-2)   room(0,0,2)
               room(0,2,0)
neighbors : room(0,-2,0)   room(2,0,0)
               room(-2,0,0)
initialvalue : 24
in : HeatInput ColdInput
link : HeatInput in@room(3,3,0)
link : HeatInput in@room(2,2,1)
link : ColdInput  in@room(3,3,2)
link : ColdInput  in@room(1,3,3)
localtransition : heat-rule
portInTransition : in@room(3,3,0)  setHeat
portInTransition : in@room(2,2,1)  setHeat
portInTransition : in@room(3,3,2)  setCold
portInTransition : in@room(1,3,3)  setCold

 [heat-rule]
 Rule: {( (-1,0,-1)+(0,-1,-1)+(0,0,-1)+
        (0,1,-1) + (1,0,-1) + (-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) + (-1,0,1) + (0,-1,1) +
       (0,0,1)+(0,1,1)+(1,0,1)+(0,0,-2) +
        (0,0,2)+(0,2,0)+(0,-2,0)+(2,0,0) +
           (-2,0,0) ) / 25 } 1000 { t }
[setHeat]
rule : { uniform(24,80) } 1000 { t }

[setCold]
rule : { uniform(-45,10) } 1000 { t }

[Heater]
distribution : exponential
mean : 10
initial : 1
increment : 0

[Cooler]
distribution : exponential
mean : 10
initial : 1
increment : 0
```

Figure 9. Definition of the heat diffusion model

Here, lines 1 to 4 are used to define the upper level model, and the components. The couplin g between these models are also shown in these lines.

The coupling scheme of the models is shown in the following figure.

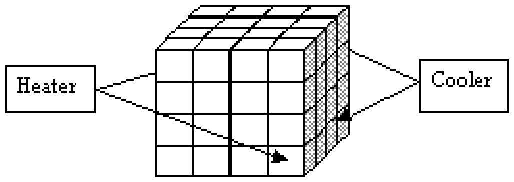

Figure 10. Coupling scheme of the heat diffusion model

Lines 6 to 32 are used to define the model representing the room to be studied. It is composed by a cellular automata of 10x10x4 cells. Initially, every cell in the space has a temperature of 24° C (as seen in the line 22). Lines 34 and 35 are used to define the function used to compute the present value for the cell, as an average between all the neighboring cells.

Lines 36 and 37 are used to define a function that generates a temperature value in the range [24, 80], using a uniform probabilistic distribution. These values are received through the In port of the cells (2,2,1) and (3,3,0), as it is defined in the lines 24 y 25. Likewise, lines 39 and 40 are used to define the function that generates temperatures in the range [-45, 10] with uniform distribution. These values will be received through the In port of the cells (1,3,3) and (3,3,2). The remaining lines are used to define the heat and cold generators, which send temperature values with a frequency of *x* seconds. In this case, the values of *x* are defined by an exponential distribution, with an average value of 40 seconds.

The following figure shows a part (first two levels) of the results generated by the tool when the model is executed:

```
Time: 00:00:00:000
      0    1    2    3       0    1    2    3
 +--------------------+ +--------------------+
0| 24.0 24.0 24.0 24.0| | 24.0 24.0 24.0 24.0|
1| 24.0 24.0 24.0 24.0| | 24.0 24.0 24.0 24.0|
2| 24.0 24.0 24.0 24.0| | 24.0 24.0 24.0 24.0|
3| 24.0 24.0 24.0 24.0| | 24.0 24.0 24.0 24.0|
 +--------------------+ +--------------------+
Time: 00:00:02:000
      0    1    2    3      0    1    2    3
 +--------------------+ +--------------------+
0| 24.6 24.0 24.6 23.8| | 24.0 24.0 27.1 23.8|
1| 23.2 24.0 24.8 24.4| | 24.0 25.5 25.5 24.0|
2| 24.6 25.5 26.1 25.4| | 27.1 25.5 25.5 25.4|
3| 24.6 25.2 26.1 23.0| | 23.8 25.5 25.4 25.4|
 +--------------------+ +--------------------+
       ...           ...            ...
Time: 00:00:15:738
0    1    2    3         0    1    2    3
 +--------------------+ +--------------------+
0| 24.2 24.2 24.2 24.2| | 24.2 24.2 24.2 24.2|
1| 24.2 24.2 24.2 24.2| | 24.2 24.2 24.2-24.0|
2| 24.2 24.2 24.2 24.2| | 24.2 24.2 24.2 24.2|
3| 24.2 24.2 24.2-43.1| | 24.2 24.2 24.2 24.2|
 +--------------------+ +--------------------+
       ...           ...            ...
```

Figure 11. Simulation results of the heat diffusion model

In simulated time 00:00:01:000, the heater and cooler produce changes in the cells where they are connected. Consequently, the state of the neighbors of these cells will change in time 00:00:02:000. In the following stages, the rest of the cells will also change, following the rules of the model. Finally, in simulated time 00:00:15:738, the refrigerator will produce a change in the cells (1,3,3) and (3,3,2), adding the values -24 y –43.1. This values will produce future changes in the neighboring cells.

CONCLUSION

This work introduced an extension to the tool CD++ used for the modeling and simulation of Cell-DEVS models. This formalism allows hierarchical construction of the models, which improves the development, checking and maintenance phases. Ten-fold improvements in the development cycle could be achieved by using the tool for common problems.

The extensions introduced to the tool allow to represent new models in other domains for the state variables. It also offers the possibility to use several functions, which permits the creation of complex models in a simple fashion.

The new input/output port definitions allow to define multiple interconnection between Cell-DEVS or DEVS models. The main changes were devoted to include n-dimensional models, making the tool a general framework to define and simulate complex generic models.

The tool is public domain, and it can be found at http://www.dc.uba.ar/people/profesores/wainer. It was constructed using a standard version of C++. At present, it is running into different platforms (including Windows 95/NT, Linux, AIX, HP -UX and Solaris).

REFERENCES

Barylko, A.; Beyoglonián, J.; and Wainer, G. 1998. "CD++: a tool to develop binary Cell-DEVS models" (in Spanish). In *Proceedings of the XXII Latin-American Conference on Informatics.* (Quito, Ecuador).

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

Giambiasi, N. and Miara, A. 1976. "SILOG: A practical tool for digital logic circuit simulation". In *Proceedings of the 16th. D.A.C.*, San Diego, U.S.A.

Rodríguez, D. and Wainer, G. 1999. "Redefinition of a specification language for Cell-DEVS models". In *Proceedings of Information Systems Analysis and Synthesis, ISAS'99*. Florida, USA.

Wainer, G. 1998. "Cellular models with explicit timing delays". Ph. D. Thesis. DIAM/IUSPIM. Université d'Aix -Marseille III. France.

Wainer, G. and Giambiasi, N. 1998. "Specification, modeling and simulation of timed Cell-DEVS spaces". Technical Report 98-007. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. Submitted for publication.

Zeigler, B. 1976. *Theory of Modeling and Simulation*. Wiley, N.Y.

Zeigler, B. 1984. *Multifaceted Modeling and discrete event simulation*. Academic Press.

Zeigler, B. 1990. "Object-oriented simulation with hierarchical modular models". *Academic Press*.