Models of Complex Physical Systems Using Cell-DEVS

Javier Ameghino Alejandro Troccoli

Departamento de Computación
FCEN – Universidad de Buenos Aires
Planta Baja. Pabellón I.
Ciudad Universitaria (1428)
Buenos Aires. Argentina.

Gabriel Wainer

Systems and Computer Engineering Department
Carleton University
4456 Mackenzie Building
1125 Colonel By Drive
Ottawa, ON. K1S 5B6. Canada.

E-mail: gwainer@sce.carleton.ca

Abstract

We present the definition of diverse models of physical systems using the Cell-DEVS paradigm. Cell-DEVS is an extension of the DEVS formalism that allows the definition of cellular models. We have developed a tool implementing these theoretical concepts, making easy the definition of cell spaces with explicit timing delays. Diversity of problems can be attacked in a simple fashion, reducing the development times of complex models. A wide variety of models have been developed using this approach, and here we include examples of a fire spreading model with different conditions, formation of a watershed and robots in a manufacturing plant. These examples allow us to show the potential application of the formalism and related tools to attack different problems.

1. Introduction

Recent advances in computer technology have influenced simulation techniques to become an effective approach to understand physical systems. In recent years, grid-shaped cellular models have gained popularity in this sense [1, 2]. In particular, Cellular Automata [3] have been widely used with these purposes. A Cellular Automata is defined as an infinite n-dimensional lattice of cells whose values are updated according to a local rule. This is done simultaneous and synchronously using the current state of the cell and the state of a finite set of nearby cells (known as the neighborhood).

Cellular automata usually require large amounts of compute time, mainly due to its synchronous nature. The use of a discrete time base is also constrains the precision of the model. Likewise, independent timing properties for each cell cannot be easily described. Cell-DEVS [4] solves these problems by using the DEVS paradigm [5] to provide discrete event approach to build cell spaces. The goal is to build discrete-event cell spaces, improving their definition by making the timing specification more expressive.

In this work we introduce the definition of several complex physical models using the paradigm. It has been shown that its application of the paradigm produces substantial reductions in the development times for cell-shaped models [6]. The goal here is to show the usefulness of the approach when applied in the definition of complex physical systems.

2. The DEVS Formalism

DEVS (Discrete EVent Systems specifications) is a modelling paradigm based on general systems theory [5]. A DEVS model is built using a set of behavioral components called **Atomic**, which can be combined to form **Coupled** ones. A DEVS atomic model can be formally described as:

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

where ***X*** represents a set of input events, **S** a set of states, and ***Y*** is the output events set. Four functions manage the model behavior: δ_{int} the internal transitions, δ_{ext} the external transitions, λ the outputs, and **D** the duration of a state. Each model is seen as having input and output ports to communicate with other models. The input and output events will determine the values to appear in those ports. The input external events are received in an input port, and the model specification should define the behavior under such inputs. The internal events produce state changes,

This work was partially supported by ANPCYT Project 11-04460 and UBACYT Project JW10.

whose results are spread through the output ports. The port influences will determine if these values should be sent to other models.

Coupled models are integrated by other DEVS basic models (atomic or coupled), providing a structural view. These models are formally defined as:

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

where ***X*** is the set of input events, and ***Y*** is the set of output events. **D** is an index of the components, and for each i $\in$ D, $\mathbf{M_i}$ is a basic DEVS model. $\mathbf{I_i}$ is the set of influencees of model *i*. For each j $\in$ I_i, $\mathbf{Z_{ij}}$ is the *i* to *j* translation function. Coupled models are composed by a set of basic models connected through input/output ports. The influencees of a model are used to define which output values must be sent to the others. The translation function uses an index of influencees, created for each model (I_i). This function defines which outputs of model M_i are connected to inputs in model M_j.

As previously explained **Cell-DEVS** [4] has extended the DEVS formalism, allowing the implementation of cellular models. Each cell is defined as an atomic model using timing delays, and it can be later integrated to a coupled model representing a cell space. Cell-DEVS atomic models can be specified as:

$$TDC=< X, Y, I, S, N, delay, d, \delta_{INT}, \delta_{EXT}, \tau, \lambda, D >$$

Here, **X** defines external input events, **Y** external output events, and **I** the model's interface. **S** is the set of states for the cell, and **N** is a set of input events. **Delay** defines the kind of delay for the cell, and **d** its duration. Finally, there are several functions: δ_{INT} for internal transitions, δ_{EXT} for external transitions, τ for local computations, λ for outputs and **D** for the state's duration function. Each cell uses N inputs to compute the future state. They are received through the model's interface, and are used to activate the local function. A delay can be associated with each cell, allowing deferring the transmission of the execution results [7]. **Transport** delays model a variable commuting time, and **inertial** delays have preemptive semantics (some scheduled events can be avoided). The model advances through the activation of the internal, external, output and state duration functions, as in other DEVS models.

Once the behavior of one cell is defined, they can be combined into a coupled model:

$$GCC=< Xlist, Ylist, I, X, Y, n, \{t1,...,tn\}, N, C, B, Z >$$

Here, **Xlist** is an input coupling list, **Ylist** is an output coupling list and **I** represents the model interface. **X** is the set of external input events and **Y** the external output events. The **n** value defines the dimension of the cell space, **{t1,...,tn}** is the number of cells in each dimension and **N** is the neighborhood set. **C** is the cell space, **B** is the set of border cells, and **Z** a translation function. The coupled model is built as an array of atomic cells, each connected to its neighborhood. The space borders are provided with different behavior, or their cells are connected with those in the opposite border. Finally, the Z function defines internal and external couplings.

These specifications were used as the theoretical base to develop a modelling tool called **CD++** [8]. The definitions of DEVS and Cell-DEVS, were used to define executable models. DEVS atomic models can be defined as C++ functions (by defining the behavior of the functions δ_{int}, δ_{ext}, λ, and D). Instead, Cell-DEVS models are built using a specialized language.

First, we allow the definition of coupled models by including their size, neighborhood and borders (as described in the formal specifications). Then, the cell behavior (defined by the local transition function) is described using rules with the following syntax: VALUE DELAY { CONDITION }. If the *CONDITION* of a rule is satisfied, the state of the cell will change to the designated *VALUE* after a *DELAY*. If the condition is not valid, the following rule is evaluated. A wide range of functions and operators can be used to define these rules. Input/output ports allow to integrate cellular models to other DEVS.

Details about the tool can be found in [8, 9], where several simple models where presented. The following sections will be devoted show how more complex physical models can be simulated using CD++.

3. Forest Fires

Forest fires destroys important resources, hence, enormous efforts have been made to prevent them. One way of attacking this problem is by using modelling and simulation. Many forest fires models have been developed to study how the fire spreads under different environmental conditions. The use of GIS (Geographical Information Systems) has extended the analysis of burnt areas and helped to determine risk factors.

A well known model for fire propagation in forests is due to Rothermel [10]. Based on environmental and vegetation conditions, it computes the ratio of spread and intensity of fire. 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). We have used the NFFL (Northern Forest Fire Laboratory), that classifies vegetation in 13 groups representing the majority of existing forest types in the region.

When Rothermel's rules are applied to a given fuel model and environmental parameters, it can determine the spread ratio (i.e. the distance and direction the fire moves in a minute). The first step is to use the fuel model, the speed and direction of the wind, the terrain topology and the dimensions of the cellular space to obtain the spread ratio in every direction. These values are used to write a specific model for the given parameters using CD++. For instance, the following figure shows the values obtained for a fuel model group number 9, a SE wind of 24.135 km/h and a cell size of 15.24 x 15.24 m.

```
Wind direction = 45.000000 (bearing)
Wind speed = 8.045000 [kph] NFFL model = 1
Cell Width = 15.240000 [m] (E-W)
Cell Height  = 15.240000 [m] (N-S)
Max. Spread = 17.967136 [mpm]
0° Spread = 5.106976 [mpm] Distance = 15.2400 [m]
45° Spread = 17.967136   Distance = 21.552615
90° Spread = 5.106976    Distance = 15.240000
135° Spread = 1.872060   Distance = 21.552615
180° Spread = 1.146091   Distance = 15.240000
225° Spread = 0.987474   Distance = 21.552615
270° Spread = 1.146091   Distance = 15.240000
315° Spread = 1.872060   Distance = 21.552615
```

Figure 1. Parameter definition computed using the Rothermel model.

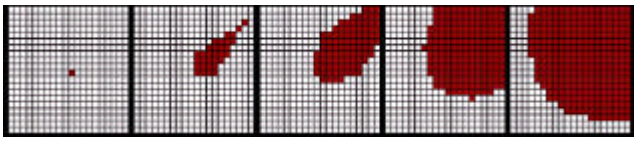
These parameters were used to write a specific cellular model for this case using the CD++ tool. The following specification shows a 20 by 20 Cell-DEVS representing the terrain and vegetation. Every parameter defined corresponds to the specification of the coupled Cell-DEVS presented in section 2. In this case, the state variables use a 0 value to indicate the absence of fire and a value different to 0 indicates the time the fire has started on that cell.

The rules define the behavior of the local computing function. They are devoted to detect the presence of fire in the eight neighboring cells. If there is fire in one of them, then present 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) + (21.552615/17.967136), which is the time the fire will start in the cell. As the spread ratio is 17.967136 mpm and a cell has a diagonal of 21.552615 m, it will take (21.552615/17.967136) minutes for the fire to reach the a cell once it has started in its SW neighbor. Therefore, we use a delay of (21.552615/17.967136)*60000 ms after which the present cell state will spread to the neighbors. The remaining rules represent a similar behavior for the remaining neighbors.

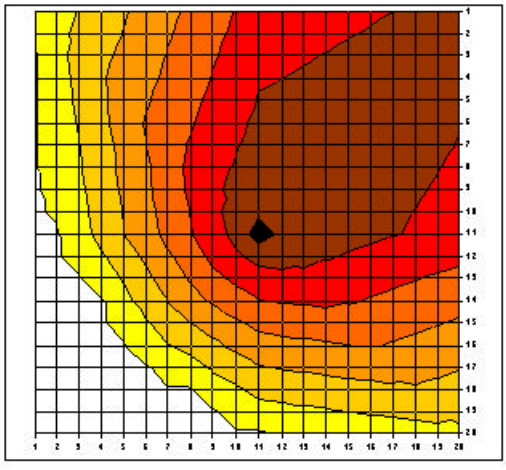
```
[ForestFire]
type : cell              dim : (20,20)
delay : inertial         border : nowrapped
neighbors : (-1,-1) (-1,0) (-1,1) (0,-1)  (0,0)
(0,1) (1,-1)  (1,0)  (1,1)
localtransition : FireBehavior

[FireBehavior]
rule : {(1,-1)+(21.552615/17.967136)} {(21.552615
/ 17.967136)*60000} {(0,0)=0 and 0<(1,-1)}
rule : {(1,0)+(15.24/5.106976)} {(15.24 /
5.106976)*60000} {(0,0)=0 and 0<(1,0)}
rule : {(0,-1)+(15.24/5.106976)} {(15.24 /
5.106976)*60000} {(0,0)=0 and 0<(0,-1)}
rule : {(-1,-1)+(21.552615/1.872060)} {(21.552615
/ 1.872060)*60000} {(0,0)=0 and 0<(-1,-1)}
rule : {(1,1)+(21.552615/1.872060)} {(21.552615 /
1.872060)*60000} {(0,0)=0 and 0<(1,1)}
rule : {(-1,0)+(15.24/1.146091)} {(15.24 /
1.146091)*60000} {(0,0)=0 and 0<(-1,0)}
rule : {(0,1)+(15.24/1.146091)} {(15.24 /
1.146091)*60000} {(0,0)=0 and 0<(0,1)}
rule : {(-1,1)+(21.552615/0.987474)} {(21.552615
/ 0.987474)*60000} {(0,0)=0 and 0<(-1,1)}
rule : {(0,0)} 0 { t }
```

Figure 2. Definition of a fire forest model.

The results of running this model are shown below. This behavior has been discussed previously in [11, 12] and others. We want to show that a complex model as this one can be easily defined using the formal specifications of Cell-DEVS and related tools.



(a)



(b)

Figure 3. (a)Fire propagation results; (b) A two-hour period (each zone represents 20 minutes).

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 to cellular automata 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 of time than a non burning cell in the opposite direction. Another advantage is that expressing a timing delay is done in a natural fashion, allowing the modeler to reduce the development time related with timing control programming.

Another advantage is that the complexity of this physical phenomenon is such that the inclusion of other external influences is difficult to be considered. The following subsections we will show the potential use of the formalism by including external factors to attack the fire.

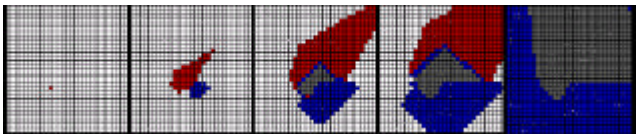
4.1 Forest fires under the influence of rain

Cell-DEVS allows to easily include new rules. In this case we have defined a rainstorm moving to the SE, extinguishing the fire on burning cells. To allow this behavior, the following rules were added to the previous model:

```
rule : -1 {60000*3}   {(0,0)=0 and ((-1,0)=-1 or
(0,1)=-1 or (-1,0)=-2 or (0,1)=-2)}
rule : -2 {60000*3.5} {(0,0)>0 and ((-1,0)=-1 or
(0,1)=-1 or (-1,0)=-2 or (0,1)=-2)
rule : -3 {60000*4.5} {(0,0)=-2}
rule : -4 {60000*5}   {(0,0)=-3}
```

Figure 4. Forest fire. Rules defining rain.

Negative values define the effects of the rain. A cell whose value is -1 is a wet cell where no fire was presented previously. A value of -2 or -3 indicates the cell was previously on fire and is now cooling down, and a value of -4 means the fire on that cell is extinguished. The first rule in the previous figure defines rain spreading to the SW. The second defines the cooling process on a burning cell, and the third and fourth ones represent advance in the cooling process. The model assumes that the fire on a cell will take 16 minutes to extinguish (in stages of different length).



(a) (b) (c) (d) (e)

Figure 5. Fire evolution with rain. (a) Start. (b) Fire advance and rain (dark gray) (c, d) Rain cooling fire areas (light gray). (d) Rain extinguished fire areas.

It is important to notice that if any of the cells is scheduled to start burning and it gets wet before the fire starts, it will not burn. This was easily defined by an inertial delay, which preempts any scheduled event if a new event from a neighbor cell before the scheduled time, and the present cell gets a different value.

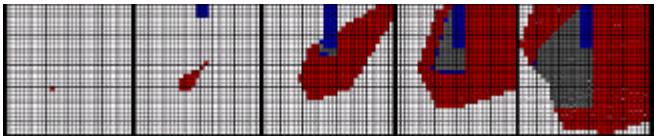
4.2 Firefighter influence on a fire

The following extension allows to analyze the influence of firefighters in the region. A negative value is still used for wet or cooling cells, a positive value for burning cells, but the way in which the water is spread has been changed.

```
rule : -1 60000     {(0,0)=0 and (-1,0)=-1}
rule : -2 {60000*7} {(0,0)>0 and ((-1,1)=-1 or (-
1,1)=-4) }
rule : -3 {60000*9} {(0,0)=-2}
rule : -4 {60000*9} {(0,0)=-3}
```

Figure 6. Rules defining firefighter behavior.

In this case, firefighters move from north to south spreading water to non burning vegetation. Once they reach a burning cell they will hold their positions till the fire is extinguished, and then they will move towards SW.



(a) (b) (c) (d) (e)

Figure 7. Fire evolution with firefighters. (a) Start (b) Firefighters spread coolant from N to S (c) Fire spreading, firefighter zones cooled down (light gray) (d, e) areas of fire extinguished.

4. Robot movement in a plant

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. The model defined in this section will show how to define this coupling. We model the movement of robots 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 North, South, East or West 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. Routes are one-way; once the load is delivered, robots are taken off the floor, back to their starting point. The model here presented supposes the robots move using predefined routes, specified as a floor diagram.

The plant is represented by a 20 by 20 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).

```
[top]
components : Floor Source1@Generator
Source2@Generator Source3@Generator
Source4@Generator
link : out@Source1 in1@Floor
link : out@Source2 in2@Floor
link : out@Source3 in3@Floor
link : out@Source4 in4@Floor

[Floor]
type : cell           dim : (20,20)
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)
link : in4 in@Floor(19,6)
localtransition : RobotsMov

[RobotsMov]
% ------ Robot 1 -----------------------------
rule : 10 1000 {(0,1)=1 and (0,0)=0 and cell-
pos(1)!=4}
rule : 11 1000 {(0,1)=1 and (0,0)=0 and cell-
pos(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 cell-
pos(0)!=17}
rule : 21 2000 {(-1,0)=2 and (0,0)=0 and cell-
pos(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 8. Model definition for robot routes.

This model definition, which follows the DEVS and Cell-DEVS specifications for coupled models, begins by a coupled model with 5 components (*Floor*, a Cell-DEVS, and *Source1-Source4*, load generators built as DEVS models). Then, the model's influencees are defined (generators output ports are connected to *Floor* input ports). Finally, we define the Cell-DEVS *Floor* coupled model parameters (size, borders, delay, etc.). The Xlist and Ylist are defined following, allowing to define how the external events received. In this case, the input ports *in1* to *in4* will be coupled to the cell space: events arriving on port *in1* should be sent to the *in* port of the cell at position (12,19). The behavior of the DEVS generators is not included here (see, for example, DEVS generators in [5]). The tool allows their definition according to the formal specifications of section 2 for DEVS atomic models.

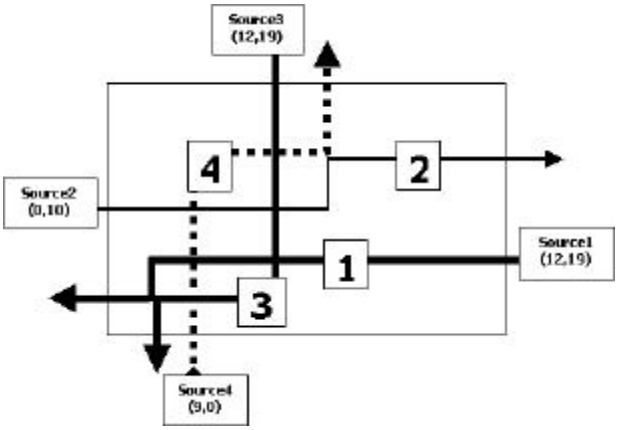


Figure 9. Floor plan with robots routes.

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. Then, valid values for a cell containing a route 2 robot will be 3,30, 31, 4, 40, 41; for cells containing a route 3 robot they will be 5, 50, 51, 6, 60, 61 and for cells containing a route for 4 robot 7, 70, 71, 8, 80, 81.

A robot moves 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.

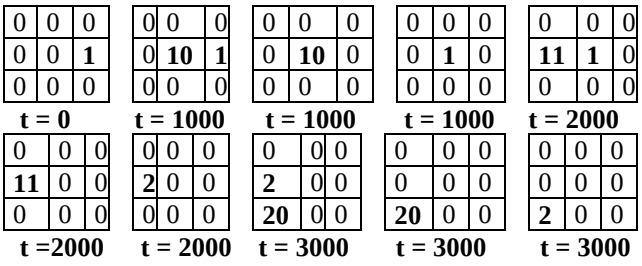


Figure 10. Route 1 movements. (a) Starting point (b) Step 1 (c) Step 2 (d) Step 3 (e) Turning point. Step 1(f) Step 2 (g) Step 3 (h) New movement with change of direction. Step 1 (i) Step 2 (j) Step 3

The results of running this model are shown below:

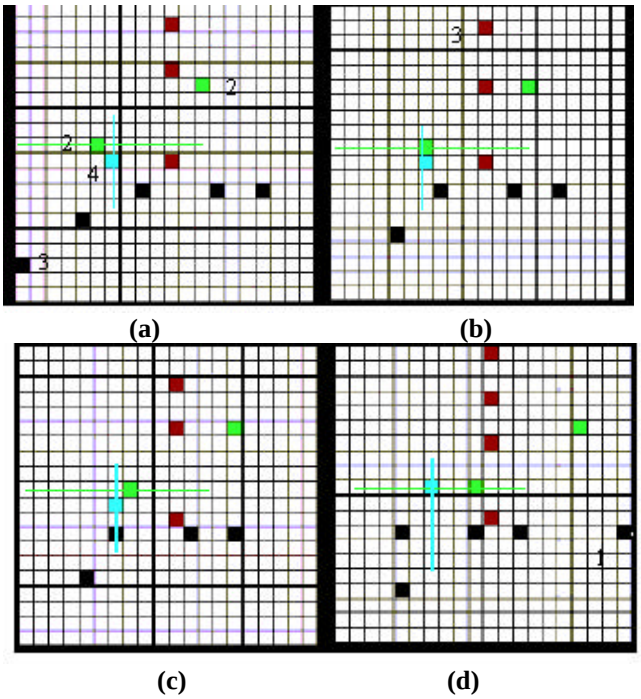


Figure 11. Executing the robots model (showing two robots reaching an intersection point).

The figure shows different robots running at different speeds (according with their delays). The figure also shows the collision avoidance between a robot in route 2 and other in route 4 (marked in the figure).

5. A Watershed Model

In [9] we presented a watershed model previously defined in [13], which was built using Cell-DEVS. A watershed is a natural region that acts as the water-receiving area of a drainage basin. The water that accumulates has different origins: rain, rivers and snow melting from mountains, as shown in the following figure.

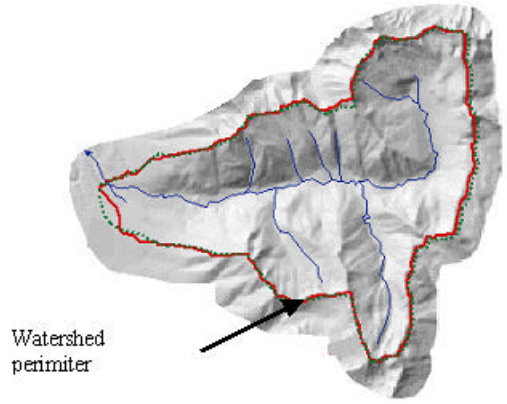


Figure 12. Topology of a watershed

A watershed is made of different vertical layers: air, vegetation, surface waters, soil, ground water, and bed-rock. The model in [13] represented the water flow and accumulations based on the characteristics of the different layers. A watershed was divided into cellos where the water accumulation was computed as shown in the following figure.

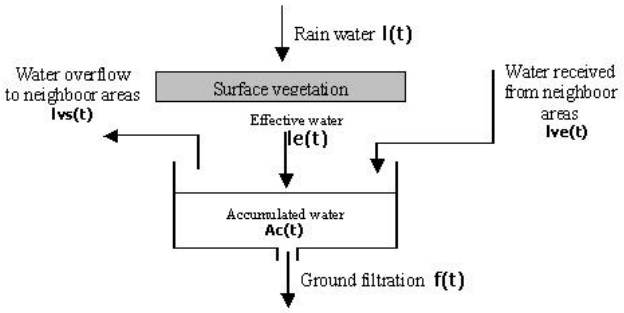


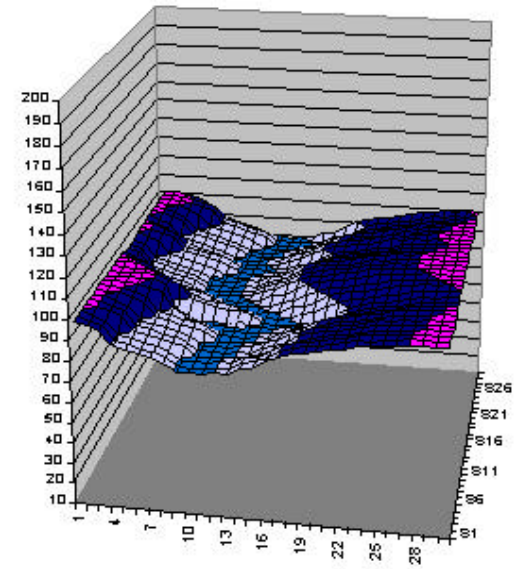
Figure 13. Hydrology model [13].

Basically, what this model says is that the height of accumulated water depends on the rain water that reaches the ground, the water received from neighbor cells, the water that overflows to neighbor cells and the water the ground absorbed. Based on the equations for this model, the CD++ model shown in the following figure was developed to simulate the accumulation of water under the presence of constant rain (7,62 mm/hour). The original model in [9] assumed the soil in the whole watershed area was of the same type. A new model here presented defines areas of different soil type: grass and rocks.

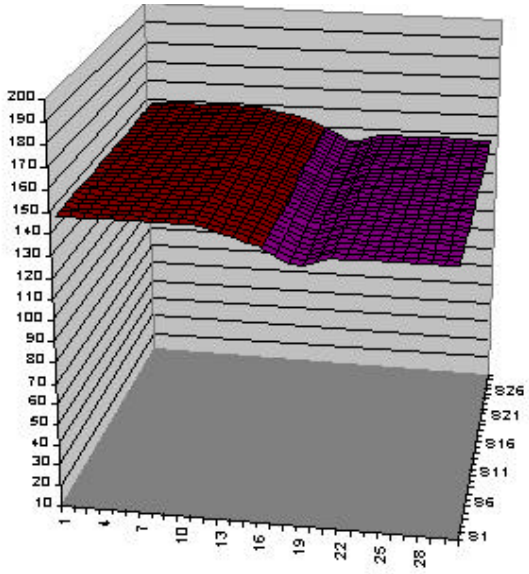
```
[Watershed]
type : cell              dim : (30,30,2)
delay : inertial         border : nowrapped
neighbors : (-1,0,0) (0,-1,0) (0,0,0) (0,1,0)
(1,0,0)(-1,0,1) (0,-1,1) (0,0,1)(0,1,1) (1,0,1)
zone : grass { (0,0,0)..(29,10,0) }
zone : stones { (0,20,0)..(29,29,0) }
localtransition : Hydrology
[grass]
rule : {0.07 + (0,0,0) - if(((((0,0,1) +
(0,0,0))>((-1,0,1) + (-1,0,0)))),(((((0,0,0) +
(0,0,1) - (-1,0,0) - (-1,0,1))/1000) *
(0,0,0))/1000),0) - if(((((0,0,1) +
(0,0,0))>((1,0,1)  +  (1,0,0)))),(((((0,0,0) +
(0,0,1) - (1,0,0) - (1,0,1))/1000) *
(0,0,0))/1000),0) - if(((((0,0,1) +
(0,0,0))>((0,-1,1)+(0,-1,0)))),(((((0,0,0) +
(0,0,1) - (0,-1,0) - (0,-1,1))/1000) *
(0,0,0))/1000),0) - if(((((0,0,1) +
(0,0,0))>((0,1,1) + (0,1,0)))),(((((0,0,0) +
(0,0,1) - (0,1,0) - (0,1,1))/1000) *
(0,0,0))/1000),0) + if(((((-1,0,1) + (-
1,0,0))>((0,0,1) + (0,0,0)))),((((-1,0,0) + (-
1,0,1) - (0,0,0) - (0,0,1)) * (-1,0,0))/1000),0)
+ if(((((1,0,1) + (1,0,0))>((0,0,1) +
(0,0,0)))),((((1,0,0) + (1,0,1) - (0,0,0) -
(0,0,1)) * (1,0,0))/1000),0) + if(((((0,-1,1) +
(0,-1,0))>((0,0,1) + (0,0,0)))),((((0,-1,0) +
(0,-1,1) - (0,0,0) - (0,0,1)) * (0,-
1,0))/1000),0) + if(((((0,1,1) +
(0,1,0))>((0,0,1) + (0,0,0)))),((((0,1,0) +
(0,1,1) - (0,0,0) - (0,0,1)) * (0,1,0))/1000),0)
} 1000 { cellpos(2)=0 }
rule : { (0,0,0) } 1000 { t }

[stones]
rule : {0.09 + (0,0,0) - if(((((0,0,1) +
(0,0,0))>((-1,0,1) + (-1,0,0)))),(((((0,0,0) +
(0,0,1) - (-1,0,0) - (-1,0,1))/1000) *
(0,0,0))/1000),0) - if(((((0,0,1) +
(0,0,0))>((1,0,1)  +  (1,0,0)))),(((((0,0,0) +
(0,0,1) - (1,0,0) - (1,0,1))/1000) *
(0,0,0))/1000),0) - if(((((0,0,1) +
(0,0,0))>((0,-1,1)+(0,-1,0)))),(((((0,0,0) +
(0,0,1) - (0,-1,0) - (0,-1,1))/1000) *
(0,0,0))/1000),0) - if(((((0,0,1) +
(0,0,0))>((0,1,1) + (0,1,0)))),(((((0,0,0) +
(0,0,1) - (0,1,0) - (0,1,1))/1000) *
(0,0,0))/1000),0) + if(((((-1,0,1) + (-
1,0,0))>((0,0,1) + (0,0,0)))),((((-1,0,0) + (-
1,0,1) - (0,0,0) - (0,0,1)) * (-1,0,0))/1000),0)
+ if(((((1,0,1) + (1,0,0))>((0,0,1) +
(0,0,0)))),((((1,0,0) + (1,0,1) - (0,0,0) -
(0,0,1)) * (1,0,0))/1000),0) + if(((((0,-1,1) +
(0,-1,0))>((0,0,1) + (0,0,0)))),((((0,-1,0) +
(0,-1,1) - (0,0,0) - (0,0,1)) * (0,-
1,0))/1000),0) + if(((((0,1,1) +
(0,1,0))>((0,0,1) + (0,0,0)))),((((0,1,0) +
(0,1,1) - (0,0,0) - (0,0,1)) * (0,1,0))/1000),0)
} 1000 { cellpos(2)=0 }
rule : { (0,0,0) } 1000 { t }
```

Figure 14. Specification of a watershed model.

Cell-DEVS was defined as an n-dimensional paradigm. In this case, we are using a three dimensional model to defined a different behavior using overlapped planes. The model is a 30 by 30 cell model with two surfaces, one to represent the height of the water retained (surface 0) and one to represent the topography of the terrain (surface 1). Each cell represents an area of 1m by 1m. The value for a surface 0 cell represents the height of accumulated water and the one for a surface 1 cell represents the ground elevation. These values for ground elevation do not change through out the simulation, and they are used to calculate the water overflow to neighbor cells.



(a)



(b)

Figure 15. (a) Original topology (b) the watershed after rain water has accumulated.

We show two zones, representing the sets of cells that will model grass and rock areas. For each zone, different sets of rules apply. Each rule calculates the new water height by applying the hydrology model equation. These

rules represent the water accumulation changing the surface vegetation and ground filtration parameters showed in figure 13. We can see that the definition of different behavior int he same model is defined without much effort.

From the execution results we can see that, despite the shape of the original topology, water accumulates in the left part of the terrain faster than in the right part. This is due to the rocky soil defined in the rightmost area, which rejects most of the water. The center part of the figure has the higher filtration of the area due to the lack of vegetation, thus having the most profound height in the watershed.

6. Conclusion

Cell–DEVS allows to describe physical systems using an n-dimensional cell-based approach. Input/output port definitions allow defining multiple interconnection between Cell-DEVS or DEVS models. Complex timing behavior for the cells in the space can be defined using very simple constructions. Transport and inertial delays allow the modeler to make easier the timing representation of each cell in the space. The CD++ tool, based on the formalism entitles the definition of complex cell-shaped models using a high-level specification language. In this way, the construction of simulations is improved, enhancing their safety and development costs.

It was shown that different kinds of applications can be easily faced, 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.

REFERENCES

[1] SIPPER, M. "The emergence of cellular computing". IEEE Computer. July 1999. pp. 18-26.

[2] TALIA, D. "Cellular processing tools for high-performance simulation". Computer. September 2000 pp.44–52.

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

[4] WAINER, G.; GIAMBIASI, N. "Timed Cell-DEVS: modelling and simulation of cell spaces". In *Discrete Event Modeling & Simulation: Enabling Future Technologies*, to be published by Springer-Verlag. 2001.

[5] ZEIGLER, B.; KIM, T.; PRAEHOFER, H. "Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems". Academic Press. 2000.

[6] WAINER, G.; GIAMBIASI, N. "Application of the Cell-DEVS paradigm for cell spaces modelling and simulation.". Accepted for publication in *Simulation* (SCS journal). October 2000.

[7] GHOSH, S.; GIAMBIASI, N. "On the need for consistency between the VHDL language constructions and the underlying hardware design". *Proceedings of the 8th. European Simulation Symposium*. Genoa, Italy. Vol. I. pp. 562-567. 1996.

[8] RODRIGUEZ, D.; WAINER, G. "New Extensions to the CD++ tool". In *Proceedings of 31st SCS Summer Computer Simulation Conference*. 1999.

[9] AMEGHINO, J.; WAINER, G. "Application of the Cell-DEVS paradigm using N-CD++". *Proceedings of the 32nd SCS Summer Computer Simulation Conference*. Vancouver, Canada.

[10] ROTHERMEL, R. “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; 1972. 40 p.

[11] VASCONCELOS, M. "Simulation of fire behavior with a Geographical Information System". M.Sc. Thesis. The University of Arizona. 1988.

[12] VASCONCELOS, M.; GONÇALVES, A.; BARROS, F. "Dynamic Maps". In *Proceedings of AI, Simulation and Planning in High Autonomy Systems*. Tucson, Arizona. 2000.

[13] MOON, Y.; ZEIGLER, B.; BALL, G.; GUERTIN, D. P. "DEVS representation of spatially distributed systems: validity, complexity reduction". *IEEE Transactions on Systems, Man and Cybernetics*. pp. 288-296. 1996.

Keywords: DEVS models, Cellular models, Cell-DEVS models, Modelling methodologies, Simulation support systems: environments.