*Proceedings of the 2012 Winter Simulation Conference*
*C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher, eds*

# TUTORIAL: ADVANCED SPATIAL SYSTEMS WITH CELLULAR DISCRETE-EVENT MODELING AND SIMULATION

Gabriel Wainer

Dept. of Systems and Computer Engineering
Carleton University Centre for Visualization and Simulation (V-Sim)
Carleton University
1125 Colonel By Dr. Ottawa
ON K1S5B6, CANADA

## ABSTRACT

Grid-shaped cellular models have gained popularity as an effective approach to understand physical systems. Despite their usefulness to describe complex behavior, many cellular models require large amounts of compute time, mainly due to its synchronous nature. Besides this, cellular models do not describe adequately most of existing physical systems whose nature is asynchronous. In this tutorial we discuss different advanced methods for modeling and simulating cellular models. We introduce the main characteristics of the Cell-DEVS formalism, and show how to model cell spaces in an asynchronous environment.

## 1 INTRODUCTION

Simulation models of complex physical systems have generally used analytical methods and 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). Instead, methods like DEVS (Discrete EVent Specification) formalism (Zeigler et al. 2000) allow the specification of discrete event models. A different discrete time method has focused on the use of grid-shaped spatial models. In particular, Cellular Automata (CA) have been widely used for these purposes (Wolfram 1986). CA are synchronous, and the use of discrete time can constrain the precision of the model and limit the execution of complex models. Besides, CA do not describe adequately most of existing physical systems, whose nature is asynchronous.

The Cell-DEVS formalism (Wainer and Giambiasi 2002; Wainer 2009) deals with these issues by using discrete-event cell spaces (based on DEVS), improving their definition by making the timing specification more expressive. Cell-DEVS models are described using a hierarchical and modular specification, and can be thus easily combined with the different modeling formalisms that were successfully mapped as DEVS (Petri Nets, Queuing Networks, Finite State Machines, etc.). The DEVS and Cell-DEVS formalisms were implemented in a modeling and simulation tool, called CD++ (Wainer 2009, Bonaventura et al. 2012; Wainer and Liu 2009) which was successfully used to develop different types of systems: biological (ecological models, heart tissue, ant foraging systems, fire spread, etc.), physical (diffusion, binary solidification, excitable media, surface tension, etc.), artificial (robot trajectories, traffic problems, heat seeking devices, etc.), and others (Wainer and Castro 2010; Wainer 2007; Wainer 2006). The techniques we used enabled execution of the models using simulation engines which are completely independent from the modeling aspects. We have developed different kinds of simulation engines (centralized, parallel distributed and real-time), which were used to execute the same models (Liu and Wainer 2011).

We show the main characteristics of the DEVS and Cell-DEVS formalisms, and how to model complex cell spaces in an asynchronous environment. The following sections focus on the application of these techniques for advanced cellular model definition.

## 2 DEVS AND CELL-DEVS

DEVS was originally defined as a discrete-event modeling specification mechanism (Zeigler et al. 2000).

It is derived from systems theory, and allows one to define hierarchical modular models that can easily be reused. 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 formally described by:

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

| | |
|---|---|
| $X$ | is the set of *external* events |
| $Y$ | is the set of *output* events |
| $S$ | is the set of *sequential* states; |
| $\delta_{ext}: Q \times X \rightarrow S$ | is the *external state transition function*; where |

$Q := \{ (s, e) \mid s \in S, 0 \leq e \leq ta(s) \}$ and $e$ is the elapsed time since the last state transition.

| | |
|---|---|
| $\delta_{int}: S \rightarrow S$ | is the *internal state transition function*; |
| $\lambda : S \rightarrow Y$ | is the *output function;* |
| $ta : S \rightarrow R_0^+ \cup \infty$ | is the *time advance function*; |

Each model is seen as having input (***X***) and output (***Y***) ports to communicate with other models. The input and output events determine the values to appear in those ports. The input external events are received in input ports, and the specification of the external transition function (**$\delta_{int}$**) defines the behavior under such inputs. The internal transition function (**$\delta_{ext}$**) is activated after the lifetime of the present state has been consumed, which is defined by the time advance (**ta**) function. Its goal is to produce an internal event, which leads to a state change. The desired results are spread through output ports by the output function (**λ**), which executes before the internal transition.

A DEVS coupled model is composed of atomic or coupled submodels, and they are formally defined:

$$CM = < X_{self}, Y_{self}, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}>$$

$D$ is a set of components; for each $i \in D$,
$M_i$ is a component with the constraint that $M_i = < X_i, Y_i, S_i, \delta_{i\ ext}, \delta_{i\ int}, \lambda_i, ta_i)$
for each $i \in D \cup \{ self \}, I_i$ is the set of influences of $i$.
for each $j \in I_i\ Z_{i,j}$ is a function, the $i$ - *to* -$j$ output-input translation
$I_i$ is a subset of $D \cup \{ self \}$, $i$ is not in $I_i$,
$Z_{self,j} : X_{self} \rightarrow X_j$ with $Z_{i, self} : Y_i \rightarrow Y_{self}$ ; $Z_{i,j} : Y_i \rightarrow X_j$
*select* : subset of D → D such that for any non-empty subset $E$, $select\ ( E ) \in E$

A coupled model groups several DEVS models together into a compound model that can be regarded, due to the closure property, as another DEVS model. Each model is seen as having input (***X***) and output (***Y***) ports to communicate with other models. Coupled models are defined as a set (**D**) of basic components (**$M_i$** atomic or coupled), which are interconnected. The translation function (**$Z_{ij}$**) is in charge of converting the outputs of a model into inputs for the others. To do so, an index of influencees (**$I_i$**) is created for each model. This index defines that the outputs of the model $M_i$ are connected to inputs in the model $M_j$, where j is an element of $I_i$. A coupled model can have its own input and output events, as defined by the $X_{self}$ and $Y_{self}$ sets. Input-output mappings are defined by the $Z$ function selects a tie-breaking function activated in the case of models having simultaneous activation.

The Cell-DEVS formalism (Wainer and Giambiasi 2002; Wainer 2009) was defined as an extension to Cellular Automata combined with DEVS. Cell-DEVS extends the DEVS formalism, allowing the implementation of cellular models with explicit timing delays. A Cell-DEVS model is an n-dimensional grid, whose cells hold state variables and a set of rules that are in charge of updating the cell state according to a local rule. This is done using the present cell state and those of a finite set of nearby cells (called its *neighborhood*). Cell-DEVS improves execution performance by using a discrete-event approach where each cell is a DEVS atomic model.

Using Cell-DEVS has different advantages. First, we have asynchronous model execution, which we showed results in improved simulation execution times. Explicit timing constructions permit defining complex conditions for the cells in a simple fashion. As DEVS models are closed under coupling, integration with other types of models in different formalisms is possible. The independent simulation engines permit these models to be executed interchangeably in single-processor, parallel or real-time simulators.

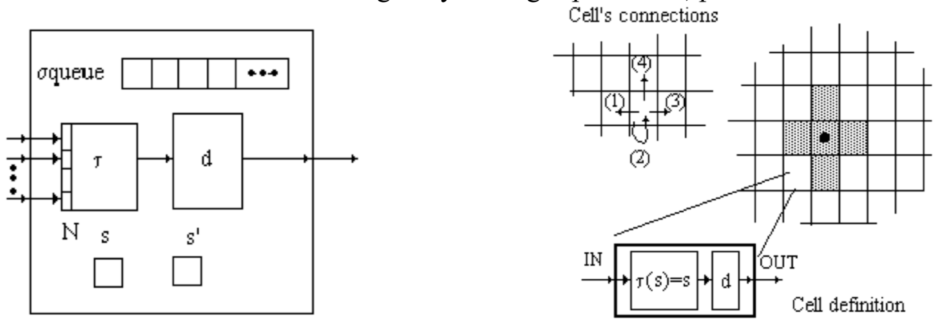


Figure 1: (a) Description of a Cell-DEVS atomic model (b) A coupled model description

A Cell-DEVS atomic model is formally defined as:

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

| | |
|---|---|
| $X$ | is a set of external input events; |
| $Y$ | is a set of external output events; |
| $S$ | is the set of sequential states for the cell; |
| delay | is the type of delay: transport or inertial; |
| d | is the transport delay for the cell; |
| $\delta_{int}$ | is the internal transition function; |
| $\delta_{ext}$ | is the external transition function; |
| $\tau$ | is the local computing function; |
| $\lambda$ | is the output function; and |
| D | is the state's duration function. |

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 (using a **queue** to keep every cell change), and *inertial* delays, which have preemptive semantics (scheduled events can be discarded). Once the cell behavior is defined, a coupled Cell-DEVS can be created by putting together a number of cells interconnected by a neighborhood relationship. A Cell-DEVS coupled model is informally presented in Figure 1 (b). A coupled Cell-DEVS model is formally defined as:

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

| | |
|---|---|
| *Xlist* | is the input coupling list; |
| *Ylist* | is the output coupling list; |
| $X$ | 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; and |
| $Z$ | is the translation function. |

This specification defines a coupled model composed of an array of atomic cells. Each cell is connected to the cells defined in the neighborhood, but as the cell space is finite, either the borders are provided with a different neighborhood than the rest of the space, or they are "wrapped", meaning that cells in one border are connected with those in the opposite one. Finally, the Z function defines the internal and external coupling of cells in the model. This function translates the outputs of m-th output port in cell $C_{ij}$ into values for the m-th input port of cell $C_{kl}$. Each output port will correspond to one neighbor and each input port will be associated with one cell in the inverse neighborhood.

## 3 THE CD++ TOOLKIT

CD++ is a modeling and simulation tool that was defined using the specifications presented in the previous section, and the basic simulation techniques introduced in Zeigler et al. (2000), Wainer (2009). 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 models can be defined using a built-in specification language. Cell-DEVS models are built following the formal specifications for DEVS models (informally presented in the previous section). CD++ makes use of the independence between modeling and simulation provided by DEVS, and different simulation engines have been defined for the platform: a stand-alone version, a Real-Time simulator, and a Parallel simulator.

The behavior specification of a cell is defined using a set of rules, each indicating the future value for the cell's state if a precondition is satisfied, with the form:

POSTCONDITION DELAY { PRECONDITION }

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. The local computing function evaluates the first rule, and if the precondition does not hold, the following rules are evaluated until one of them is satisfied or there are no more rules.

```
[ex]
width : 20     height : 40    border : wrapped
neighbors : (-1,-1) (-1,0) (-1,1) (0,-1)  (0,0)  (0,1) (1,-1)  (1,0)  (1,1)
localtransition : tau-function

[tau-function]
rule: 1 100 {(0,0)=1 and (truecount=8 or truecount=10)}
rule: 1 200 {(0,0) = 0 and truecount >= 10 }
rule: (0,0) 150 { t }
```

Figure 2: A Cell-DEVS specification in CD++

Figure 2 shows an example of a Cell-DEVS model which follows Cell-DEVS coupled model's formal definitions. In this case, *Xlist* = *Ylist* = { ∅ }. The set *{m, n}* is defined by *width-height*, which specifies the size of the cell space (in this example, *m*=20, *n*=40). The N set is defined by the lines starting with the *neighbors* keyword. The border (*B*) is wrapped. Using this information, the tool builds a cell space, I/O ports, and the Z translation function following Cell-DEVS specifications. Different operators are available to define rules and delays including: Boolean, comparison, arithmetic, neighborhood values, time, conditionals, angle conversion, pseudo-random numbers, error rounding and constants (i.e., gravitation, acceleration, light, Planck, etc.). In the example presented in Figure 2, the local computing function executes very simple rules. The first one indicates that, whenever a cell state is 1 and the sum of the state values in N is 8 or 10, the cell state remain in 1. This state change will be spread to the neighboring cells after 100 ms. The second rule states that, whenever a cell state is 0 and the sum of the inputs is larger or equal to 10, the cell value changes to 1. In any other case (*t* = true), the result remains unchanged, and it will be spread to the neighbors after 150 ms. As we can see, cells evolve using a discrete-event approach.

In order to show the basic definition of Cell-DEVS models, we present a simple model of the effects of parasitoids on population dynamics, using a model of the Diamondback Moth (DBM). The DBM is a pest native to Europe that has migrated and it is now found worldwide. The DBM has a 4 stage life cycle:

a) **Egg:** DBM lays 1-3 eggs on the underside of a leaf, which take 5-6 days to reach maturity.
b) **Larva**: this stage (split into 4 steps) lasts 10-21 days. The larvae feed on the plant, and grow. The time to maturity depends on environmental factors (temperature, food abundance, etc.).
c) **Pupa**: this stage lasts 5-15 days
d) **Adult**: once they have reached adulthood, DBM begin reproducing. A female DBM can lay upwards of 1600 eggs during its 16-day lifespan.

The total time it takes a DBM to reach maturity is about 32 days, however new generations will appear 21 to 51 days apart depending on conditions (Agriculture and Agri-Food Canada, 2008). The larval stage of the DBM destroys leafy crops (cabbage, lettuce, etc.). In Europe, DBM is controlled by natural means, as there are approximately 25 species of parasitoids that prey on the DBM and thus act as a population control. But it is a plague in the rest of the world and pesticides used to control their population become less effective after several generations (due to genetic mutation).

In Biology, predator-prey relationships like this one are often examined and modeled theoretically using discrete equations to determine population sizes at given times, using parameters derived from observations of real environments and shaped to fit the equations in order to model reality accurately. In (Tonnang et al. 2009), the authors used a purely mathematical approach to recreate the effect of introducing parasitoids (to determine the most efficient course of action to control its population). Our model (Khan et al 2012) is based on Tonnang et al. (2009). The parasitoids attack the DBM during the larval stage and inject their egg inside the host. Upon hatching, it destroys the host and consumes it, emerging in adulthood. The model identifies 7 different layers, each responsible for their own entities with limited interactions with each other and will have a pre determined set of parameters:

1. **Crop** (cabbage): it is assumed to have a fixed density (no reproduction or death). Each plant will only allow for a single egg to be laid on it, and only allow for a single reproductive pair to perch upon it.
2. **Egg and Larva**: the egg reaches maturity after 5 days; the larval stage is represented as a single stage, and will be vulnerable to attack during this period (10 days). After 10 days, a larva infected will be replaced by a parasitoid (either male or female).
3. **Pupae**: this stage lasts 5 days, after which a mature DBM will emerge if that location is empty; otherwise, it will die (representing an ecosystem with maximum support capacity).

4, 5: **Adult Female/Male DBM**: they have a lifespan of 16 days; they can travel (at random). The female searches for an empty plant and upon locating one will stay there until fertilized. To become fertilized male must occupy the same plant as the female for 1/10th of a day before leaving to find another mate. Upon fertilization, the female will leave the plant in search of an empty plant and a new mate.

6, 7: **Adult Female/Male Parasitoid**: they follow a similar search pattern than DBM, but the female searches for a plant occupied with larva.

Each cell contains two pieces of information, the direction of the next movement and the age of the cell. The movement is a two stage process. If the cell is a 1, then the next move has not been determined:

```
rule : { (0,0,0)+trunc(uniform(1,9))+ 1/1000 } 1 { trunc(0,0,0) = 1 and (0,0,0) != 0 }
```

The cell then becomes a random integer between 2 and 9. In the following step, we decide if a movement is feasible; in that case, it moves; otherwise, it reverts back to a 1 as follows:

```
rule : { (1,0,0) - 1 + 1/1000} 1 { (0,0,0) = 0 and trunc( (1,0,0) ) = 2 }
rule : 0 1 { trunc( (0,0,0) ) = 3 and (-1,1,0)=0 and trunc( (0,1,0) ) != 2 }
```

The age of the cell is stored in the decimal position. One complete movement cycle is equal to 1/500 or 0.002. For the purpose of this model one day is equal to 10 complete cycles or 0.02.

As said above, upon finding an empty plant the female remains in the cell. This is represented by the state 10, and once fertilization has occurred 11. This is performed by the following rules:

```
rule : { (0,0,0) + 9 + 1/1000 } 1 { trunc((0,0,0))=1 and trunc((0,0,-4))=1 }
rule : { (0,0,0) + 1/1000 } 1 { trunc((0,0,0))=10 and trunc((0,0,-1)) != 11}
rule : { (0,0,0) + 1 + 1/1000 } 1 {trunc((0,0,0))=10 and trunc((0,0,-1))=11}
rule : { (0,0,0) - trunc(uniform(1,9)) + 1/1000 } 1 { trunc((0,0,0)) = 11 }
```

A similar approach is taken with the males, with the difference being that they search for an available female, as follows:

```
rule : { (0,0,0) + 9 + 1/1000 } 1 {trunc((0,0,0))=1 and trunc((0,0,-6))=10 }
rule : { (0,0,0) + 1 + 1/1000 } 1 { trunc((0,0,0)) = 10 }
rule : { (0,0,0) - 10 + 1/1000 } 1 { trunc((0,0,0)) = 11 }
```

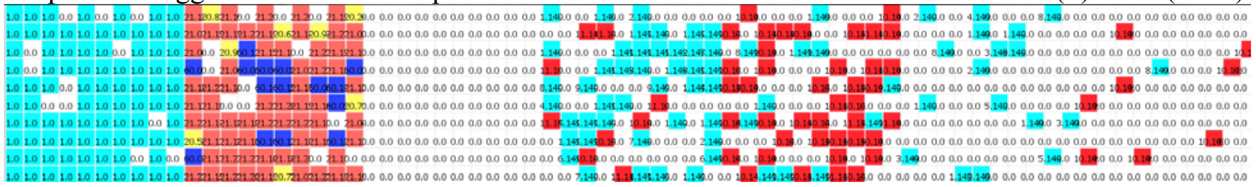
Finally, upon reaching a count of 0.32 the insect will die:

```
rule : 0 1 { remainder((0,0,0),1) > 0.32 }
```

The different parameters in the model can be adjusted to model different biological events accurately. In the following figures we show three different scenarios: annihilation of DBM population, disappearance of parasitoids, equilibrium. In the first scenario, we consider an initial population of parasitoids that is too high (and they are too effective predators), thus, they may destroy the population of DBM. This is undesirable since any reappearance of the DBM population would be allowed to grow uncontrollably (the parasitoid population would have died as well, since it is not a native species). In this test, the population density of DBM was set to 50%, the density of parasitoids 10% and the density of plants was 90%.

t= 00:00:00:150

Crop Egg and Larvae Pupae Female DBM Male DBM Parasitoid (F) P. (Male)



t= 00:00:01:000 (approximately 3 Generations)

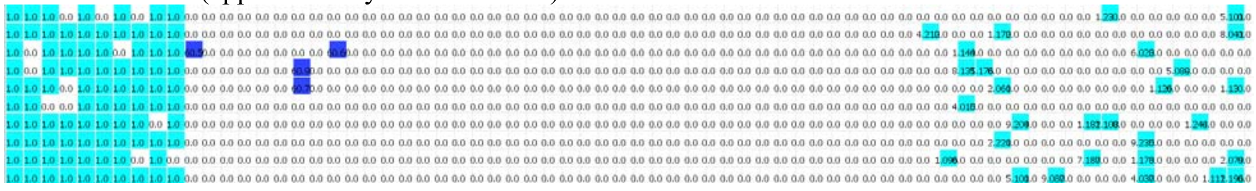
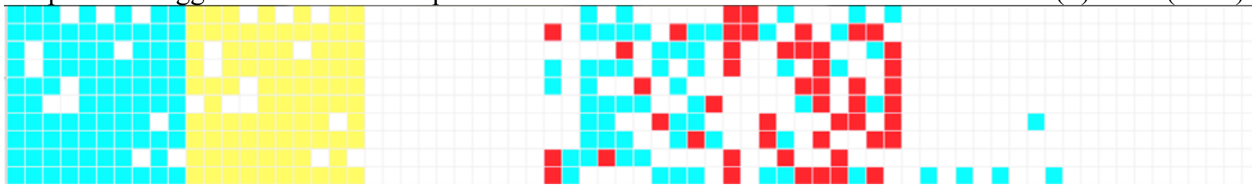


Figure 3: Annihilation test scenario (Khan et al. 2012)

As we can see, the large initial population of parasitoids allowed them to overrun the DBM population and resulted in their complete destruction. Our second example shows a case where the initial density of parasitoids was 2%, and, consequently, their population disappears.

t= 00:00:00:100

Crop Egg and Larvae Pupae Female DBM Male DBM Parasitoid (F) P. (Male)



t= 00:00:00:300 (approximately 1 Generations)

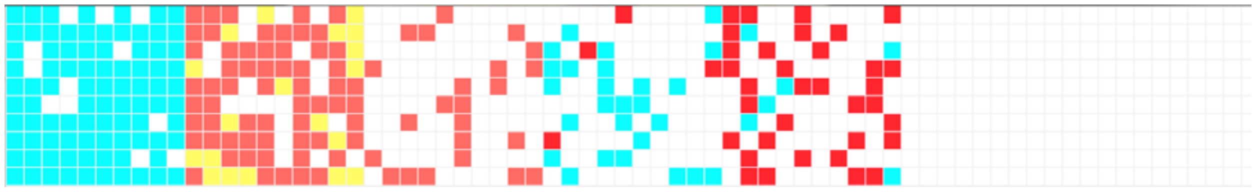


Figure 4: Low level of parasitoid scenario (Khan et al. 2012)

The last execution result presented here considers a case of equilibrium, in which the initial conditions of Figure 4 were not changed, but the parasitoid population density was changed to be 3%.

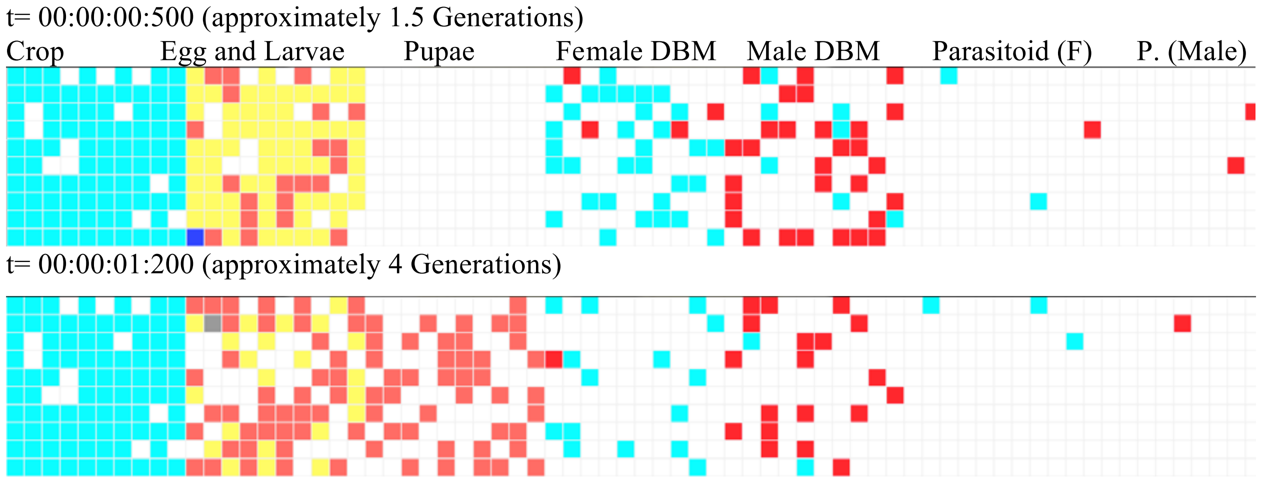


Figure 5: Equilibrium (Khan et al. 2012)

The results show that the parasitoids managed to maintain a relative equilibrium for a notable length of time, supporting the idea that released parasitoids may be able to control the population sizes of DBM.

## 4 ADVANCED CELL-DEVS MODELS IN CD++

CD++ was originally based on Cellular Automata, whose formal specification includes only one state variable on each cell. As seen in the previous section, when complex models are needed, multiple state variables provide a better mechanism for modeling the problem at hand. As we can see, we can allocate a different state variable on a different layer in a Cell-DEVS model; use the integer/decimal part of the state variable to manipulate information that is more complex. Likewise, each cell is only allowed to transfer information through a fixed set of input/output ports: one for inputs and one for outputs, which are automatically created with the cell and not accessible to the modeler. In order to improve the model definition, we expanded the modeling language (López and Wainer 2004), allowing adding state variables on each cell as follows:

```
StateVariables: pend temp vol
StateValues: 3.4 22 -5.2
InitialVariablesValue: initial.var
```

The first one declares the list of state variables for every cell. The second one declares the default initial values for these states variables. The last sentence provides the name of a file where the initial values for some particular cells are stored, using the following format:

```
(0,0,1) = 2.8 21.5 -6.7
(2,3,7) = 6 20.1 8
```

The values assigned to the variables follow the order in which they are listed in the sentence *StateVariables*. Then, state variables can only be referenced from within the rules that define the cells' behavior (preceded by a **$**), as follows.

```
rule: {(0,0,0)+$pend} 10 { (0,0,0)>4.5 and $vol<22.3 }
```

A second extension allows using multiple I/O ports to communicate with the neighbors. They are defined as a list of neighbor port names as follows:

```
NeighborPorts: alarm weight number
```

The input and output neighbor ports share the names, making it possible to automatically calculate the influences: an output port from a cell will influence exclusively the input port with the same name in every cell in its neighborhood. In the example three ports are declared (*alarm*, *weight* and *number*). When a cell outputs a value through one of these ports, it will be received by all its neighbor cells through their input ports with the same name. A cell can read the value sent by one of its neighbors, specifying the input port. Both the cell and port must be specified, separated by a tilde (~):

```
  rule : 1 100 { (0,1)~alarm != 0 }
```

In this case, if the cell receives an input in the *alarm* port from the cell to the right, and that value is not 0, the cell status will change to 1. This change will be transmitted through the default output port 100 time units after.

The identifier '**:=**' is used to assign values to state variable, and they are placed in a new section of the rules (between curly brackets) including a list of assignments, each one followed by a semi-colon.

```
  ; <port assignments>          [ <assignments> ]          <delay>    <precondition>
  rule: { (0,0,0)+1 }   { $temp:=$vol/2; $pend:=(0,1,0); }   10     { (0,1,0) > 5.5 }
```

If the precondition is true, the variable *temp* will be assigned half of *vol*, *pend* will be assigned the value of the neighbor cell (0, 1, 0), and the value of *vol* will remain unmodified. This assignments are executed immediately (they *are not delayed*). The output value is only transmitted after the delay. As one might need to output values through many ports at the same time, the assignment can be used as many times as needed. For instance, the following rule is such that, if we receive a value larger than 50 from the port *number* in the cell to the right, we wait 100 time units, and generate an output of 1 in the *alarm* port; we copy the *weight* received from the cell to the left into the *weight* output port.

```
  rule: { ~alarm := 1; ~weight := (0,-1)~weight; } 100   { (0,1)~number > 50 }
```

An excerpt of the whole grammar can be found in the Appendix.

## 4.1 Modeling Clouds in Visual Systems

The generation of clouds for real-time simulators visualizations is challenging; consequently, billboard images (which do not evolve) are used. In Dobashi (1999) the authors showed how to model cloud behavior using cellular automata. In this section we show how to use CD++ generate such cloud behavior, extending the model by Dobashi (1999) with input ports to inject events and control the evolution of the system. We also show how to model the wind moving the clouds in the neighborhood (Khan et al. 2012).

The model by Dobashi uses a 3D neighborhood, and provides four simple functions to represent the transformation from water vapor to clouds, and then the dissolution of clouds (using random distributions). Four state variables are used: *hum* (vapor), *cld* (clouds), *act* (a phase change from vapor to cloud), and *ext* (cloud extinction). A cell must first have humidity (*hum*); then, if there are neighbors in the process of changing, it enters the transformation stage (*act*). After this, a cloud is formed (*cld*), and it remains present until extinguished (which happens when a cell has neighbors already in the dissolution process). Once fading is complete (*ext* = true); the cloud is removed from the cell.

The following figure shows an excerpt of the model in CD++. *Act*, *hum*, and *ext* where represented using Boolean state variables. The presence of clouds, *cld* is with a value of 10. The macros *hum-change*, *act-change*, and *ext-change* implement the transition rules. We use a normal distribution applied uniformly to all cells in the neighborhood. To adjust the behavior of the model, the average and standard deviation values are different for each state variable

```
[cloudGen]
type : cell                                 dim : (20,20)
delay : transport                           border : wrapped
neighbors : (0,0) (-2,0) (-1,0) (1,0) (2,0) (0,-2) (0,-1) (0,1) (0,2)
localtransition : cloud-gen

stateVariables : hum act ext
stateValues :       0   0   0
NeighborPorts:  hum act ext
```

```
[cloud-gen]
rule : { ~out := 10; ~ext := $ext; ~act := $act; }
 {$hum := #macro(hum-change); $ext := #macro(ext-change); $act := #macro(act-change);}
              100       {$ext = 0 AND ((0,0) = 10 OR $act = 1)}
rule : { ~out := 0; ~ext := $ext; ~act := $act; }
 {$hum := #macro(hum-change); $ext := #macro(ext-change); $act := #macro(act-change);}
              500       {$ext = 1 AND (0,0) = 10}
rule : { ~out := 0; ~ext := $ext; ~act := $act; }
 {$hum := #macro(hum-change); $ext := #macro(ext-change); $act := #macro(act-change);}
              100       {t}
#BeginMacro(ext-change)
( if( ( $ext = 0 AND (0,0) = 10 AND
    ( (1,0)~ext = 1 OR (-1,0)~ext = 1 OR (0,1)~ext = 1 OR (0,-1)~ext = 1 OR
     (2,0)~ext = 1 OR (-2,0)~ext = 1 OR (0,2)~ext = 1 OR (0,-2)~ext = 1  )      )
              OR normal(0.5,0.1) > 0.5, 1, 0)  )

#BeginMacro(act-change)
( if( ( $act = 0 AND $hum = 1 AND
    ( (1,0)~act = 1 OR (-1,0)~act = 1 OR (0,1)~act = 1 OR (0,-1)~act = 1 OR
     (2,0)~act = 1 OR (-2,0)~act = 1 OR (0,2)~act = 1 OR (0,-2)~act = 1  )    )
              OR normal(0.5,0.1) > 0.5, 1, 0)   )

#BeginMacro(hum-change)
   ( if(($hum = 1 AND $act = 0) OR normal(0.5,0.1) > 0.5, 1, 0) )
```

Figure 6: Model specification (Khan et al. 2012)

Two other state variables exist: one to control the generation of clouds (*avggen* in the macro *act-change*), and the other (*avgext* in the macro *ext-change*) to control the dissolution of clouds. Several inputs were added to the atomic model, in order to specify the *avggen* and *avgext* value for the cells they are connected to. The transition functions defined previously were expanded to accommodate this new behavior. Two new macros were defined to update the average values, *avggen-change* and *avgext-change*. The model also defines two additional transition functions used when for providing inputs to a specified cell: *set-average-gen* and *set-average-ext*. These functions, combined with manipulating the output *avggen* and *avgext* variables ensure the average state variable information is diffused correctly.

```
[set-average-gen]
~avggen := portValue(thisPort);
$avggen := $avggen + portValue(thisPort); $avgext := #macro(avgext-change);

[cloud-gen]
~avggen := 0;  ~avgext := 0;
$hum := #macro(hum-change);
$ext := #macro(ext-change);
```

Figure 7: Distribution transition functions (Khan et al. 2012)

Figure 7 shows the components of the new transition functions that update the various average distribution variables. The process by which these variables are updated when an input occurs is as follows:

1. The function *set-average-gen* (or *set-average-ext* respectively) is called when an input occurs.
2. The cell's state variable is added to the value in the input port (adding to the current average).
3. The cell puts the value of the input on the appropriate average output port.
4. The neighboring cells update their average by using *avgext-change* or *avggen-change*, adding the scaled input value obtained from the original cell output port.
5. Each cell then sets the average output ports to 0 (to prevent positive feedback).

The influence of wind defined as follows: if wind is present in a cell (along with clouds), the wind would blow the cloud into a neighboring cell (and vice versa). The variables *windup* and *windright* indicate that the wind is blowing from the S or from the W (negative values are used for N or E, and combinations for the appropriate diagonals). Wind variables are static during the simulation, and are set as initial conditions. The transition rules are such that, when a cloud is present in the cell and these variables do not equal zero the cloud is blown into an adjacent cell, as follows.

```
{~out=10 ….}  {…}    { if( #macro(blown-cld-in), 300, 100 ) }
  { (#macro(blown-cld-in) AND NOT #macro(ext-cld)) OR (#macro(gen-cld) AND NOT
     #macro(blown-cld-out)) }

{~out=0 ….} {…}  { if( #macro(blown-cld-in), 300, 300 ) }
{  #macro(blown-cld-out) OR #macro(ext-cld) }
```

Figure 8: (a) Cloud Generation with Wind  (b) Cloud Dissolution with Wind

Two additional macros were added to simplify the transition rules. *Blown-cloud-in* determines if any cell in the neighborhood has wind blowing in the right direction and has a cloud; in that case, the cell becomes cloudy. *Blown-cloud-out* controls if the cell should lose its current cloud due to wind.

The following figure shows an execution example, in which we use different input values over time for cloud dissolution and generation (providing an improved manner of controlling the cloud generation).

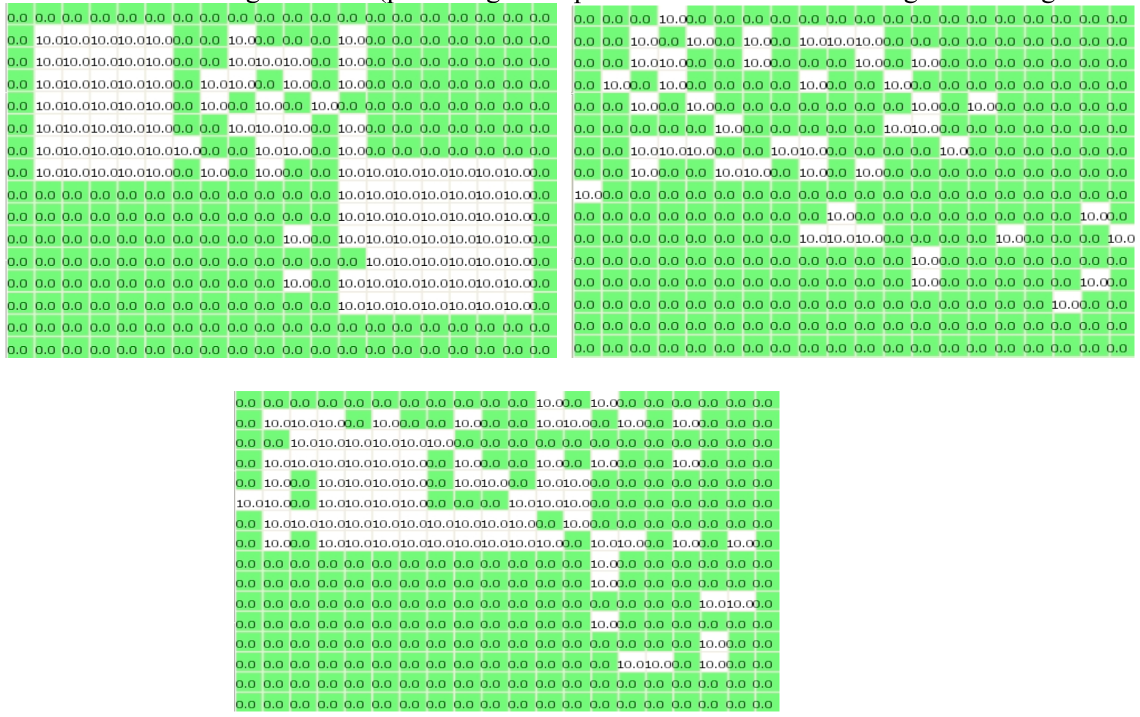


Figure 9: Cloud generation at times 01:000; 01:500; 05:000. (Khan et al. 2012)

## 4.2 A Model of the Mountain Pine Beetle Epidemic

The Mountain Pine Beetle (MPB) epidemic (Taylor et al. 2006) is an outbreak of that kills Lodgepole Pine and is causing extensive damage. These beetles (http://en.wikipedia.org/wiki/Mountain_pine_beetle) primarily infect Lodgepole pine trees (http://en.wikipedia.org/wiki/Lodgepole_pine), which are harvested for commercial purposes. It is estimated that the MPB has killed 283 million m$^3$ of pine trees in BC, Canada from 1990 to 2005 Bone et al. (2007). It is also estimated that up to 80 percent of all harvestable Lodgepole pines could be killed by 2013 if nothing is done to stem their progression (http://www.shim.bc.ca/atlases/fbc/ss3/Forest.html). Currently, the industry best practices are to clear cut any forest infected with MPB to slow or stop infestation in the surrounding area, producing a loss of revenue.

In Bone et al. (2007) the authors showed how to model the spread of the MPB effectively using CA. In this section we show a Cell-DEVS model based on the one defined by Bone et al. and we show extensions in which various types of forest features are included: a river, a lake, and an incomplete clear cut.

The CA model for MPB progression involves looking at the total susceptibility (TS) value for each forest unit, obtained by simple multiplication of the size (S) of the trees in the unit, the proportion (P) of Lodgepole pine trees in the unit, the density (D) of trees in the unit, and the location factor (L), representing the proximity to a known infestation. Size S reflects the fact that larger trees are more susceptible to infestation; P and D reflect the fact that MPB will only infect pine trees and tend to do better in dense forest. Parameter L represents the beetle's ability to move; places closer to infestations are more likely to become infected. Once these parameters are defined, an allometric function ($y=0.05.x^{-1.3}$) is used to determine if the total susceptibility will result in an infestation. Once a cell is infected, there is a 100% certainty that all the Lodgepole pines in the area will be eventually infected. Empirical evidence suggests that up to 80% of MPBs in a given tree will die during the winter (i.e., each generation has 80% mortality).

The model can be summarized as follows (Khan et al. 2012):

• A 2D map that could include rivers, lakes, clear cuts, thinned trees, etc.
• Each timestep represents a year of real-time
• Each forest is in one of two states: dead or alive
• Each tree is only affected by trees in its neighborhood
• If a tree stand is alive, it monitors the amount of mountain pine beetle in the stand
• If the total concentration of MPBs is larger than the allometric function (tree mortality specified at the susceptibility level of the stand), the stand becomes infected and dies before the next season
• MPBs progress through the forest and are computed as the average concentration for the surrounding cells plus the current number of beetles less the number that died during the winter
• Trees cannot recover from a mountain pine beetle infestation

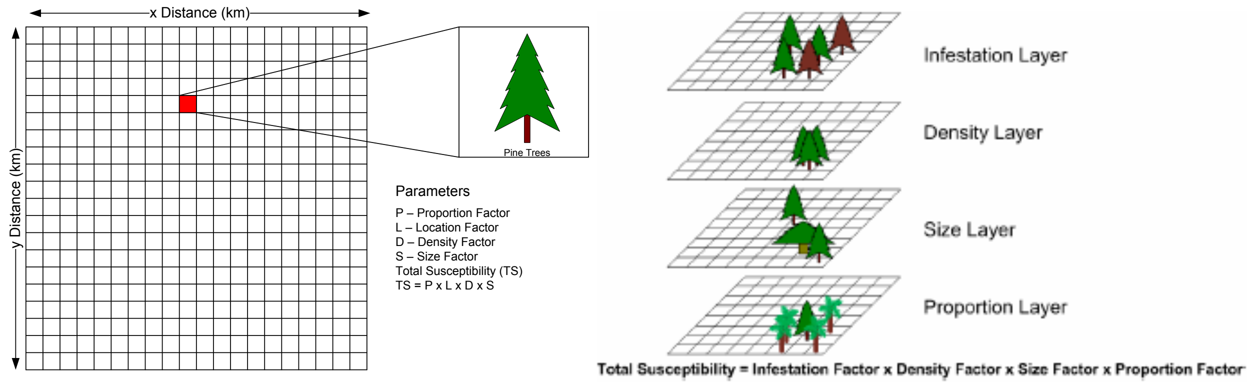


Figure 10: Model Organization and Model Layers/State Variables (Khan et al. 2012)

The model has been built using a multi-layer approach and also multiple state variables, including:

1. Mortality: it represents the trees that have been killed by mountain pine beetle
2. MPB: represents the movement of the MPBs and their relative concentration on any given cell
3. Forest factors: size, density and proportion factors for given stand. Changing these values alter how the MPBs spread in that stand

The model includes two planes with 16x16 cells (and 64x64), using non wrapped borders. The infection plane (plane 0) has 1 cell initially infected at (1,1). All the other cells are set to zero. Plane 1 is the density layer and contains all zeros except for places where it is desired that no trees grow (those locations are set to 1). The rules for each plane are based on the equations above, and are defined as shown in Figure 11:

```
[mpb-rule]
rule : {~nmpb :=1;} 100 {(0,0)~nmpb >= 1}      %NPB Travel + NBP Annual mortality
rule : {~nmpb :=#Macro(mpbAvg);~TS := 1;} 100 {#Macro(surround) = 1 and
          0.05*power(0.4*(1 - $density), -1) < (0,0)~nmpb}
```

```
rule : {~nmpb :=#Macro(mpbAvg);~TS := 1;} 100 {#Macro(surround) > 1 and
       #Macro(surround) <=4 and 0.05*power(0.6*(1 - $density), -1) < (0,0)~nmpb}
rule : {~nmpb :=#Macro(mpbAvg);~TS := 1;} 100 {#Macro(surround) > 4 and
       0.05*power(0.8*(1 - $density), -1) < (0,0)~nmpb}     %TS Calculations

#BeginMacro(surround)
(-1,-1)~TS + (-1,0)~TS + (-1,1)~TS + (0,-1)~TS + (0,1)~TS +(1,0)~TS+(1,-1)~TS+(1,1)~TS

#BeginMacro(mpbAvg)
((-1,-1)~nmpb + (-1,0)~nmpb + (-1,1)~nmpb + (0,-1)~nmpb + (0,1)~nmpb + (1,0)~nmpb +
        (1,-1)~nmpb + (1,1)~nmpb)/8 + ((0,0)~nmpb*0.2)
```

Figure 11: Model definition (Khan et al. 2012)

The cells exchange the total susceptibility to MPB infection, as well as the concentration of MPB in the cell. Ports *TS* and *nmpb* represent these two exchanges of data. A state variable is also used for the cell's factors.

The following figures show three separate simulation areas, showing different aspects of the simulation. The first square represents the tree mortality (showing dead trees); the second square shows the number of MPBs (as a concentration). The last square shows the forest factors (density, size, proportion of Lodgepole pine). A color code was used to identify different aspects: green (light gray) for trees in good health (not infected, and without infected trees around), yellow (white) for trees with a low risk of infection (i.e., there are infected cells around), orange (darker gray) for moderate risk of infection (less than half of the neighbours around the current cell are infected), red (black) for cells with high risk of infection (more than half of the neighbors are infected), and gray (or infected dead).

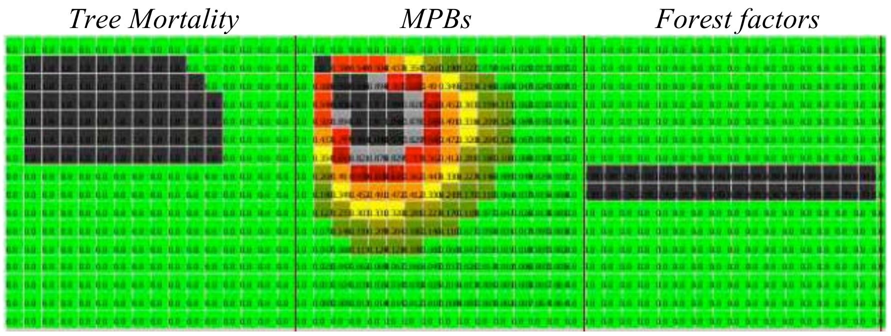


Figure 12: The river and forest model (Khan et al. 2012)

Figure 12 shows a model with a source of constant infection at (1, 1) and a forest bisected by a river (which the MPB are unable to cross). The figure shows an intermediate step with the progression of the epidemic impeded by the river.

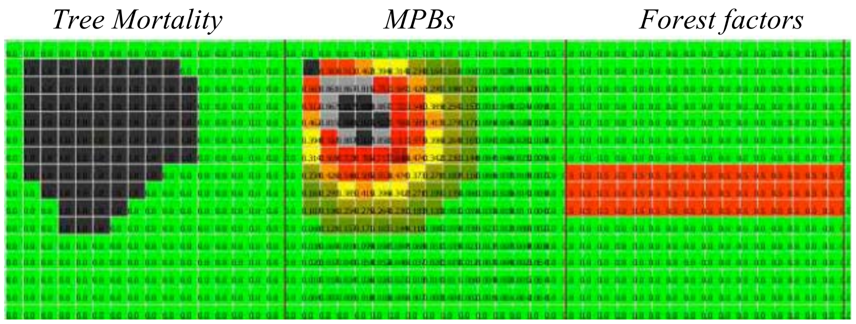


Figure 13: Forest Thinning Model (Khan et al. 2012)

Figure 14 shows how forest thinning can slow the progression of the MPB (the last plane has a region with moderate density of Lodgepole Pine); thinning is used to control the mountain pine beetle and has shown promise for reducing the spread of the beetle. We can see that the progression over the lower density area is slowing the progression of the MPB.

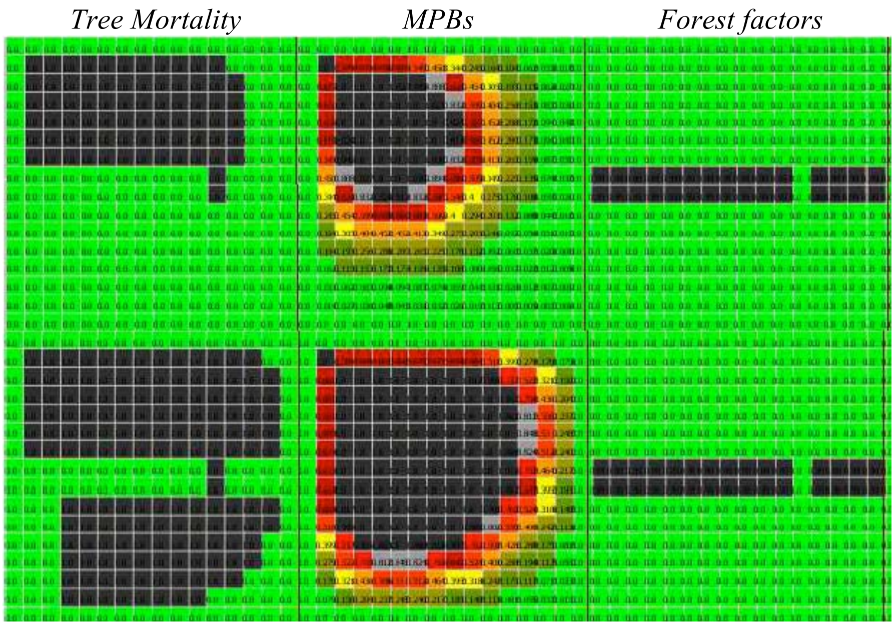


Figure 14: Incomplete clear cut model (Khan et al. 2012)

Figure 13 presents the results when the simulation includes an incomplete clear cut. In this case it is shown that the beetles are capable of traveling across the unclear cut land, and affect the trees on the other side. This example is of critical importance to the simulation of the effects of clear cutting forests for prevention of MPB outbreaks.

## 5 CONCLUSION

We have presented the Cell-DEVS formalism, and introduced several features CD++, a toolkit for DEVS modeling and simulation. The tool was built using the DEVS formal modeling technique, improving the development time of simulations. The tools are public domain and can be obtained at http://cell-devs.sce.carleton.ca. Different models were presented, showing advanced features of the tools, which showed how to simplify the modeling task for complex models. Cell-DEVS simplifies the construction of complex simulations, allowing a simple and intuitive model specification. Input/output port definitions allow defining multiple interconnections between Cell-DEVS and DEVS models. Complex timing behavior for the cells in the space can be defined using very simple constructions.

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.

## 6 ACKNOWLEDGMENTS

This work was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC). Different students participated in defining the models introduced in the paper: Daanish Khan,

Michael Lepard and Michael Van Schyndel. The complete models can be found at <http://www.sce.carleton.ca/faculty/wainer/wbgraf>.

## 7 REFERENCES

Agriculture and Agri-Food Canada. Government of Saskatchewan. 2008. *Diamondback Moth.* http://www.agriculture.gov.sk.ca/Default.aspx?DN=688b2f99-ad99-423d-900c-c01a1c45d8a1. Accessed: May 21, 2012.

Bonaventura, M., G. A. Wainer, and R. Castro. 2012. “A Graphical Modeling and Simulation Environment for DEVS”. Accepted for publication in *SIMULATION: Transactions of the Society for Modeling and Simulation International*. Accepted: December 2011.

Bone, C., S. Dragićević, and A. Roberts, A. 2007. "Evaluating forest management practices using a GIS-based cellular automata modeling approach with multispectral imagery". Environmental Modeling and Assessment. Vol. 12, No. 2. Springer. pp. 105-118.

Dobashi, Y., Nishita, T., and Okita, T. 1999. “Animation of Clouds using Cellular Automaton”. Proceedings of CGIM’99. Palms Springs, CA.

Khan, D., M. Lepard, and M. Van Schyndel. 2012. Advanced CD++ models: DBM, Clouds and MPB. http://www.sce.carleton.ca/faculty/wainer/wbgraf. Accessed: May 10, 2012.

Liu, Q. and G. A. Wainer. “Multicore Acceleration of DEVS Systems”. 2011. *SIMULATION: Transactions of the Society for Modeling and Simulation International.* doi: 10.1177/003754971141223.

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

Taylor, S. W., A. L. Carroll, R. I. Alfaro and L. Safranyik. 2006. In “The Mountain Pine Beetle: A Synthesis of Biology, Management and Impacts in Lodgepole Pine” (Safranyik, L. & Wilson, B. Eds.) Natural Resources Canada, Canadian Forest Service, Victoria, BC. pp. 67–94.

Tonnang, H., L.Nedorezov, H. Ochanda, J. Owino and B. Löhr. 2009. “Assessing the impact of biological control of Plutella xylostella through the application of Lotka–Volterra model”. *Ecological Modelling*, Vol. 220, No. 1, pp. 60-70.

Wainer, G. 2006. “Applying Cell-DEVS Methodology for Modeling the Environment”. *SIMULATION: Transactions of the Society for Modeling and Simulation International*. Vol. 82, No. 10, pp. 635-660.

Wainer, G. 2007. “Developing a software tool for Urban traffic Modeling”. *Software, Practice and Experience*. Vol. 37, No. 13, pp. 1377-1404.

Wainer, G. 2009. “Discrete-Event Modeling and Simulation: Theory and Applications”. Taylor and Francis.

Wainer, G., and R. Castro. 2010."A survey on the application of the Cell-DEVS formalism in cellular models". *Journal of Cellular Automata*. Vol. 5, No. 6. pp. 509-524.

Wainer, G., and N. Giambiasi. 2002. “N-Dimensional Cell-DEVS”. Discrete Events Systems: Theory and Applications, Kluwer. Vol. 12, No. 1. pp. 135-157.

Wainer, G., and Q. Liu. 2009. "Tools for Graphical Specification and Visualization of DEVS Models". *SIMULATION: Transactions of the Society for Modeling and Simulation International*. Vol. 85, No. 3, 131-158.

Wainer, G., and A. López. 2004. “Improved Cell-DEVS model definition in CD++”. Proceedings of ACRI 2004. Netherlands. Lecture Notes in Computer Science. Vol. 3305 Sloot, P.; Chopard, B.; Hoekstra, A. Eds.

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

Zeigler, B.P., T.G Kim and H. Praehofer, H. 2000. *Theory of Modeling and Simulation*. 2nd Ed. Academic Press.

## 8 APPENDIX – RULE LANGUAGE SPECIFICATION

```
Rule : AssignResult Result '{' BoolExp '}' | AssignResult '{' AssignSet '}' Result '{' 
BoolExp '}' ;
```

```
AssignResult :  Result | '{' PortSendSet '}' ;
```
This rule recognizes two sub-trees. The sub-tree **PortSendSet** is a list of output operations through output ports. The sub-tree **Result** is only recognized for compatibility reasons.

```
PortSendSet :  /* Empty */  | PortSend PortSendSet;
```
The **list of outputs** to neighbor ports can be **empty** or an **output operation** followed by a **list of outputs**.

```
PortSend :  SEND '(' PORTNAME ',' RealExp ')' ';' | PORTNAME OP_ASSIGN RealExp ';' ;
```
An **output operation** can be either a **send** function with a **port name** and a **value** as arguments, or a **neighbor port assignment**. Both cases must be finished by a semi-colon (;). Except for **RealExp**, all the components of the rule are basic token provided by the lexical analyzer.

```
RealExp :  IdRef ...
```

```
IdRef : CellRef OptPortName ...
```
This rule recognizes the references to identifiers. In particular references to cells (**CellRef**). The cells can be followed by a reference to the port (**OptPortName**), whose value we are interested in obtaining.

```
OptPortName :  /* Empty */  | PORTNAME ;
```
A reference to the port is just a token representing its name. However, this rule is optional.

```
AssignSet :  /* Empty */  | Assign AssignSet ;
```
This rule is a component of the rule *Rule*. It represents a list of assignments to state variables. It can be **empty** or an **assignment operation** followed by a list of **assignments to state variables**.

```
Assign : STVAR_NAME OP_ASSIGN RealExp ';' ;
```
This rule recognizes the operation that assigns a value (**RealExp**) to a state variable reference. Both **STVAR_NAME** and **OP_ASSIGN** are basic tokens recognized by the lexical analyzer.

**GABRIEL A. WAINER** (SMSCS, SMIEEE) received the M.Sc. (1993) and Ph.D. degrees (1998, *with highest honors*) at the University of Buenos Aires (UBA), Argentina, and Université d'Aix-Marseille III, France. After being Assistant Professor at the Computer Science Department of UBA, in July 2000 he joined the Department of Systems and Computer Engineering at Carleton University, where he is now Full Professor. He has been a visiting scholar at ACIMS (The University of Arizona); LSIS (CNRS), and INRIA (Sophia-Antipolis), France. He has been invited professor at the UCM, UPC (Spain), Université Paul Cézanne, Université de Nice (France). He is the author of three books and over 260 research papers; he edited nine other books, and helped organizing over 120 conferences, including being one of the founders of SIMUTools, SimAUD and the Symposium of Theory of Modeling and Simulation. Prof. Wainer is the Vice-President Publications, and was a member of the Board of Directors of the SCS. He is Special Issues Editor of SIMULATION, member of the Editorial Board of IEEE Computing in Science and Engineering, Wireless Networks (Elsevier), Journal of Defense Modeling and Simulation, and International Journal of Simulation and Process Modelling (Inderscience). He is the head of the Advanced Real-Time Simulation lab, located at Carleton University's Centre for advanced Simulation and Visualization (V-Sim). He has been the recipient of various awards, including the IBM Eclipse Innovation Award, SCS Leadership Award, and various Best Paper awards. He has been awarded Carleton University's Research Achievement Award (2005-2006), the First Bernard P. Zeigler DEVS Modeling and Simulation Award, and the SCS Outstanding Professional Award (2011). Further information can be found at <http://www.sce.carleton.ca/faculty/wainer>.