Modelling a Sand Pile Application Using Cell-DEVS

Hesham Saadawi **Gabriel Wainer**

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

Keywords: cellular automata, cellular models, Cell-DEVS, DEVS models.

Abstract

Cell-DEVS formalism extends cellular automata modeling. In this paper, we present a simple model for landslides using a model of a sand pile. The model uses Cell-DEVS formalism and CD++ tool to model and simulate landslides. Using this simple model, we present how Cell-DEVS formalism can be used to model systems that involve material accumulation and flow. Material flow and accumulation need special modeling consideration with Cell-DEVS to preserve mass conservations law. In modeling such systems, not only model updating rules would be important for correct behavior, but also choosing the model characteristics as type of delay would affect that behavior.

1. INTRODUCTION.

In recent years, computer simulation has played a key role in the analysis of complex natural systems. The cellular Automata (CA) formalism has recently gained popularity to describe complex physical systems [1]. CA is defined as infinite n-dimensional lattices of cells whose values are updated according to a local rule. This is done simultaneously and synchronously using the current state of the cell and the state of a finite set of nearby cells (known as the neighborhood). Unfortunately, CA has showed to have different problems to model physical systems: they usually require large amounts of computational time, mainly due to its synchronous nature.

The Cell-DEVS formalism [2] solved these problems by using the DEVS (Discrete EVents Systems specifications) formalism [3] to define a cell space where each cell is defined as a DEVS model. This technique permits to build discrete-event cell spaces, improving their definition by making the timing specification more explicit. Besides this, discretizing the model into a bidimensional grid poses constraints on the precision that can be achieved by the model. Finite element analysis, instead, is able to provide higher precision due to the characteristics of the technique.

Here we are going to show how to use Cell-DEVS to a sand pile model. Sand pile models provide means to analyze earthquake phenomena, as major earthquakes often cause landslides. In [4] an effort for modelling such an event using cellular automata was presented, and we will show how a model of this nature can be modeled using Cell-DEVS.

2. THE DEVS FORMALISM

Cell-DEVS defines cell spaces in which each cell is defined as a DEVS model. The DEVS formalism was originally proposed to model discrete events systems. A DEVS model is built using a set of behavioral models called **Atomic**, which can be combined to form **Coupled** ones. A DEVS atomic model is defined as:

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

Input external events are considered to be received in input ports (***X***). When an event arrives, the model executes the external transition function d_{ext} to produce a state change. Each state has an associated duration **D**. When this lifetime is consumed, the internal transition function d_{int} is activated to produce internal state changes. The internal state **S** can be used to provide model outputs, which are sent through the output ports (***Y***). They are sent by the output function l, which executes before the internal transition.

A DEVS coupled model is defined as:

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

Each coupled model consists of a set of **D** basic models $\mathbf{M_i}$ connected through input/output ports. The list of influencees $\mathbf{I_i}$ of a given model is used to determine the models to which outputs must be sent. These sets are used to build the translation function $\mathbf{Z_{ij}}$,

which is in charge of translating outputs of a model into inputs for the others. An index of influencees is created for each model (I_i). For every *j* in the index, outputs of model M_i are connected to inputs in model M_j.

In Cell-DEVS, each cell of a cellular model is defined as an atomic DEVS model. Cell-DEVS atomic models are specified as:
TDC = < *X*, *Y*, S, N, delay, d, δ_{int}, δ_{ext}, τ, λ, D >.

Each cell will use the **N** inputs to compute the future state **S** using the function t. The new value of the cell is transmitted to the neighbors after the consumption of the delay function. A **transport** delay allows us to model a variable commuting time for each cell with anticipatory semantics (every scheduled event is executed). Using **inertial** delays, the semantics is preemptive: some scheduled events are not executed due to a small interval between two input events. Therefore, the outputs of a cell are not transmitted instantaneously, but after the consumption of the delay. **Delay** defines the kind of delay for the cell, and **d** its duration. This behavior is defined by the d_{int}, d_{ext}, l and **D** functions, as in other DEVS models.

Once the atomic cell model is defined, a number of cells they can be put together to form a coupled model, built as an array of atomic cells. A Cell-DEVS coupled model is defined by:

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

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

CD++ [5] is a tool that allows a user to implement DEVS models. The tool is built as a hierarchy of models, each of them related with a simulation entity. Atomic models can be programmed in C++. A specification language allows defining the model's coupling, including the initial values and external events. The tool also enables a user to build Cell-DEVS models. The language provides a set of primitives to define the size of the cell-space, the type of borders, a cell's interface with other DEVS models and a cell's behavior. In the following sections we will show how to use Cell-DEVS to model a sand pile model.

3. A SAND PILE MODEL

This model represents a sand pile on a table area that is divided to a grid of cells. Sand particles are added to the pile continuously at the middle cell at discrete time intervals. Each cell would start redistributing its content to its four non-diagonally neighboring cells, whenever it contains four or more sand particles. In case of four particles, the cell would be empty after redistribution. For any number above four particles, the cell would only distribute four sand particles to its neighbors and keep the rest in it. The model can be described as:

1. The model is initially loaded with some sand particles in some of its cells, or it can be empty.
2. A particle would arrive at a random time intervals to the cell at the center of the model.
3. Every cell has a maximum capacity of four particles. When a cell reaches this capacity, a redistribution operation would begin that results in emptying this cell and adding one particle to each non-diagonal neighboring cell.
4. The redistribution operation can trigger another redistributions among neighboring cells, which in turn can do the same for their neighbors.
5. Boundary cells would lose particles to the environment at sides were there are no neighboring cells.
6. The redistributions would represent avalanches in the model. Severity of an avalanche can be measured either by number of cells participating in a redistribute operation, or by number of particles lost from border cells.

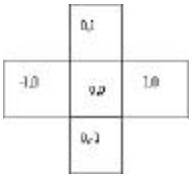

Figure 1: Neighborhood cells.

A Cell-DEVS model will represent the sand pile by dividing the model area in a grid of 10 by 10 cells. The neighborhood cells are defined as the diagram shows in Figure 1. The formal specification for the corresponding Cell-DEVS model is defined as follows:

M=<I,X,Y,Xlist,Ylist,?, N,{f,c}, C, B, Z, select>

I=<P^X,P^y>,with P^X={(N(5,5), binary) }, P^y={F};
X=1; Y= F; Xlist={IN}; Ylist= F
?=5; N={(0,0),(0,1),(1,0),(0,-1),(-1,0)}
f=10; c =10; C={Cij/ie[1,f], je[1,c]}
B={ (Cij | i = 1,10; je[1,10]) , (Cij | j = 1,10; ie[2,9]}
Z:

P_{ij}^{Y1}? $P_{i,j+1}^{X1}$	$P_{i,j-1}^{Y1}$? P_{ij}^{X1}
P_{ij}^{Y2}? $P_{i+1,j}^{X2}$	$P_{i-1,j}^{Y2}$? P_{ij}^{X2}
P_{ij}^{Y3}? $P_{i,j-1}^{X3}$	$P_{i,j+1}^{Y3}$? P_{ij}^{X3}
Pij^{Y4}? $P_{i-1,j}^{X4}$	$P_{i+1,j}^{Y4}$? P_{ij}^{X4}
Pij^{Y4}? $P_{i,j}^{X4}$	$P_{i,j}^{Y4}$? P_{ij}^{X4}

Select={(0,1), (1,0), (0,-1),(-1,0), (0,0)}

This model can be defined in the CD++ tool as shown in Figure 2.

```
[top]
components : sandpile
particleGenerator@Generator
link : out@particleGenerator in@sandpile
out :  out
link : out@particleGenerator out

[sandpile]
type : cell
dim : (10, 10)
delay : inertial     border : nowrapped
neighbors : (0,1)   (1,0)  (0,-1) (-1,0)
(0,0)
in : in
link : in in@sandpile(5,5)
localtransition : sandpile-rule
portInTransition : in@sandpile(5,5)
NewParticle-rule

[sandpile-rule]
rule : { statecount(4)+statecount(5)+
statecount(6)+statecount(7)- 1 } 100 { (0,0) =
4}
rule : { statecount(4)+ statecount(5)+
statecount(6)+statecount(7)+(0,0)} 100 {(0,0)<
4}
rule : { (0,0)+statecount(4)+statecount(5)+
statecount(6)+statecount(7)-1-4} 100 { (0,0) >
4 }
rule : {(0,0)} 100 { t }

[NewParticle-rule]
rule : { statecount(4)+ statecount(5)+
statecount(6)+statecount(7)+(0,0) + 1} 2
{(0,0)<4}
rule : { statecount(4)+ statecount(5)+
statecount(6)+statecount(7)+(0,0)} 2
{(0,0)>=4}
rule : {(0,0) + 1 } 2 {t}

[particleGenerator]
distribution : exponential
mean : 2      initial : 1   increment : 0
```

Figure 2. CD++ tool model definition file

For each cell, it can take one of the following values (states). These values and their meaning are described in Table 1.

Value	Meaning
0	An empty cell
1	A cell filled with one sand particle
2	A cell filled with two sand particles
3	A cell filled with three sand particles
4	A cell filled with four sand particles – It starts to distribute all 4 particles to neighbors in next time step.
5	A cell filled with five sand particles – It distributes four particles and keeps one in next time step.
6	A cell filled with six sand particles - It distributes Four particles and keeps two in next time step.
7	A cell filled with seven sand particles – It distributes four particles and keeps three in next time step.

Table 1. Cell's state description.

The first rule in *sandpile* states that for a cell with exactly four particles in it, the cell would take a new value equal to the sum of neighboring cells that are distributing, minus one to count to itself in statecount(4). As we only have 4 neighbors, the new value assigned would be between 0 (in case no neighboring cell is giving me any particle) to 4 (in case the cell has distributed its particles to neighbors, and taking one particle from each neighbor that has 4 or more particles). This can be shown in Figure 3. In this case, the cell is to distribute to 4 neighboring cells, each with less than 4 particles.

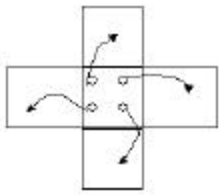

Figure 3. Rule 1 at (0,0), Rule 2 at all others

Figure 4 shows the case of a cell distributing to neighbors, and three of its neighboring cells are having 4 or more particles. Hence, it will get one particle from each overflowing cell (with 4 or more particles in it).

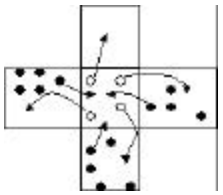

Figure 4. Rule 1 at (0,0) and (1,0); Rule 2 at (0,1); Rule 3 at (-1,0) and (0,-1)

The second rule of *sandpile* tells that if the cell has less than 4 particles, then it will not redistribute to its neighbors. The cell will only receive one particle from each overflowing neighbor (which has 4 or more particles), and will add received particles to its current value.

The following rule states that if the cell contains more than 4 particles, then it will redistribute only 4 particles to the neighbors, and retain the rest. Add to that any particles distributed from over flown neighbors. We subtract 1 to count for self-state as it would be counted (self value is more than 4), and we subtract 4 to count for 4 particles distributed to neighbors.

Finally, we define a default rule to keep the cell contents of particles if none of the previous rules is held.

The *NewParticle* rule is used for arrived sand articles generated by the DEVS generator model. If contents are less than 4 particles, for any arrival from the generator, the cell will increment its contents by 1 particle. Add to this any particles coming from overflowing

neighbors. Likewise, if the cell's contents are 4 or more particles, the cell will add 1 to its contents, plus any particles from overflowing neighbors. We subtract one from final result to count for the self state (I contain more than 4 particles) in the "statecount()".

4. TESTING THE MODEL

We test the simulation results of our model in order to verify that the desired behavior is obtained. We selected different initial values to test different possible situations and rules. First, we checked that the behavior of the model cells was identical to the required behavior in the way distribution is handled. Second, we tested that no sand particles were lost or created for the whole grid after every redistribution, i.e. the total number of sand particles on the grid remained constant after redistribution.

In our first test case, we initialized some cells with 4 or 5 sand particles, keeping the others empty. The input from the Sand Generator was disabled as we intended to test only the Sand Pile Cell-DEVS model. The results of this test can be seen in Figure 5. We can see that all cells with initial value of 4 have distributed their contents to neighboring cells and contain zero particles (Rule 1). Only the cells which initially contained 4 or more particles and a neighbor with 4 or more particles (distributing cells) would get a particle from each distributing neighboring cell (as (1,2), (1,3)). A cell like (2,6) would get a particle from each distributing neighbor (2 in this test). In addition, for all cells with content of less than 4 particles (Rule 2), their content has increased with one particle arriving from each distributing neighbor. The results are correct as per our model specifications. Total number of sand particles on the grid before and after redistributions is the same of 21 particles.

```
Line : 244 - Time:
00:00:00:000
     0123456789
    +----------+
   0|          |
   1|  45      |
   2|     4 4  |
   3|          |
   4|   4      |
   5|          |
   6|          |
   7|          |
   8|          |
   9|          |
    +----------+
```

```
Line : 2124 - Time:
00:00:00:100
     0123456789
    +----------+
   0|  11      |
   1| 11211 1  |
   2|  111 2 1 |
   3|   1 1 1  |
   4|  1 1     |
   5|   1      |
   6|          |
   7|          |
   8|          |
   9|          |
    +----------+
```

Figure 5. Test case 1 results.

Our second test included checking the Generator DEVS model, and execution results of this test can be seen in Figure 6. At each time interval, the Sand Generator would generate an output with value of 1 to represent a new sand particle. Time intervals between generated outputs can be chosen to be one of many probability distributions. In this example, we used an exponential distribution with mean of two time units.

```
00:00:00:000 out 1
00:00:00:082 out 1
00:00:01:274 out 1
00:00:01:628 out 1
00:00:01:683 out 1
00:00:08:724 out 1
00:00:10:657 out 1
00:00:12:120 out 1
00:00:15:903 out 1
00:00:16:234 out 1
00:00:17:199 out 1
00:00:18:745 out 1
00:00:19:115 out 1
00:00:20:089 out 1
00:00:24:063 out 1
00:00:25:517 out 1
00:00:25:663 out 1
00:00:27:692 out 1
00:00:33:429 out 1
00:00:37:892 out 1
00:00:38:674 out 1
00:00:38:761 out 1
00:00:39:702 out 1
00:00:42:120 out 1
00:00:45:428 out 1
00:00:46:175 out 1
00:00:47:403 out 1
00:00:49:485 out 1
00:00:50:344 out 1
00:00:53:125 out 1
00:00:55:747 out 1
. . .
00:09:55:741 out 1
00:09:59:545 out 1
00:09:59:645 out 1
00:09:59:875 out 1
```

Figure 6. Sand Generator test results.

In Figure 7: test case 2 results., we show the results of a test in which the Generator is disabled, as we want to test rule 1 (for cells containing 4 particles) and rule 3 (for cells containing more than 4 particles). In this test, cell (1,3) takes value of 5 particles along with other cells containing some other values. We see in next time step all cells having 4 or more are distributing. Cell (1,3) has distributed 4 particles and kept one, plus 4 particles from distributing neighbors (Rule 3). In next time step, cell (1,3) redistributes all its 4 particles and stays empty (Rule 1). The model starts with 42 particles and ends with the same number of particles after all redistributions, thus it conforms to the specifications.

```
Line : 251
Time:00:00:000
     0123456789
    +----------+
   0|          |
   1| 5454     |
   2|   5 4 4  |
   3|          |
   4|   4 7    |
   5|          |
   6|          |
   7|          |
   8|          |
   9|          |
    +----------+
```

```
Line : 414 -
Time:00:00:100
     0123456789
    +----------+
   0| 1111     |
   1|122412 1  |
   2| 1223 2 1 |
   3|   2 2 1  |
   4|  1 231   |
   5|   1 1    |
   6|          |
   7|          |
   8|          |
   9|          |
    +----------+
```

```
Line : 2202 -
Time:00:00:200
     0123456789
    +----------+
   0| 1121     |
   1|123 22 1  |
   2| 1233 2 1 |
   3|   2 2 1  |
   4|  1 231   |
   5|   1 1    |
   6|          |
   7|          |
   8|          |
   9|          |
    +----------+
```

Figure 7: test case 2 results.

In the third test case, we tested the model with 11 filled cells each with 4 particles or more. The cells are all internal on the grid so that no sand particles would escape the grid from border cells. As shown in Figure 8, the cells are distributing their content to neighbors until a steady state is reached. In this figure, the first three time steps are shown along with the final step in model execution. We begin the test with 56 sand particles on the grid, and finish the test with the same number after redistribution. This result confirms the model behavior as specified. The input from the Generator is disabled as we test only Sand Pile Cell-DEVS model.

```
Line : 246 - Time:
00:00:00:000
     0123456789
    +----------+
   0|          |
   1|       43 |
   2|    674 4 |
   3|    676   |
   4|    45    |
   5|          |
   6|          |
   7|          |
   8|          |
   9|          |
    +----------+
```

```
Line : 401 - Time:
00:00:00:100
     0123456789
    +----------+
   0|        1 |
   1|    112 5 |
   2|   14623 1 |
   3|   157411 |
   4|   1232   |
   5|    11    |
   6|          |
   7|          |
   8|          |
   9|          |
    +----------+
```

```
Line : 749 - Time:
00:00:00:400
     0123456789
    +----------+
   0|       11 |
   1|    233211 |
   2|   2432 21 |
   3|   321531 |
   4|   214 1  |
   5|    221   |
   6|          |
   7|          |
   8|          |
   9|          |
    +----------+
```

```
Line : 3034 - Time:
00:00:01:200
     0123456789
    +----------+
   0|    11111 |
   1|   2131311 |
   2| 111 3221 |
   3| 1123 12  |
   4|   33132  |
   5|    231   |
   6|          |
   7|          |
   8|          |
   9|          |
    +----------+
```

Figure 8. Test case 3 results.

5. SAND PILE MODEL EXECUTION RESULTS

The complete model is composed of a Generator (a DEVS model) and *SandPile* sub-models as shown in Figure 9. The Generator generates sand particles in discrete time intervals and delivers them to the *SandPile* model. These time intervals for arriving sand particles are simulated by exponential distribution. In each interval, one or more particles can be added to the *SandPile* model. In our example run, only one sand particle is generated at each time interval.

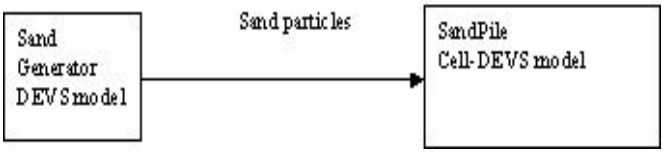

Figure 9. Sand Generator and SandPile models connected.

The sand particle is put into center cell (5,5). The cell (5,5) would accumulate sand particles to the overflowing limit of 4, and then start redistributing its contents to neighbors. After neighboring cells get 4 particles, they start redistributing to others. An avalanche as defined per [4], can be measured by number of sand particles escaping from border cells or by number of redistributing cells in one avalanche. More work on how to gather statistics information can be done on the model. Some of the execution results are shown in Figure 10. In order to trace those results, times of sand particles arriving from the Sand Generator as shown in Figure 6 should be consulted with Figure 10. Whenever a particle arrives from the generator, it appears in middle cell (5,5) after a delay of 2 time units, as seen in 00:00:00:002 time step. Cells would react to changes in their neighbors after 100 time units as specified in our model. This can explain cell (5,5) reaching value of 5 at time step 00:00:01:685 before redistributing as according to the rules, redistribution happens after 100 units, while arrival of a new particle shows in 2 time units.

```
Line : 221 -
Time:00:00:000
     0123456789
    +----------+
   0|          |
   1|          |
   2|          |
   3|          |
   4|          |
   5|          |
   6|          |
   7|          |
   8|          |
   9|          |
    +----------+
```

```
Line : 248 - Time:
00:00:002
     0123456789
    +----------+
   0|          |
   1|          |
   2|          |
   3|          |
   4|          |
   5|     1    |
   6|          |
   7|          |
   8|          |
   9|          |
    +----------+
```

```
Line : 275 -
Time: 00:00:084
     0123456789
    +----------+
   0|          |
   1|          |
   2|          |
   3|          |
   4|          |
   5|     2    |
   6|          |
   7|          |
   8|          |
   9|          |
    +----------+
```

```
Line : 349 - Time:
00:01:685
     0123456789
    +----------+
   0|          |
   1|          |
   2|          |
   3|          |
   4|          |
```

```
Line : 388 - Time:
00:01:730
     0123456789
    +----------+
   0|          |
   1|          |
   2|          |
   3|          |
   4|     1    |
```

```
Line : 408 -
Time: 00:01:785
     0123456789
    +----------+
   0|          |
   1|          |
   2|          |
   3|          |
   4|     1    |
```

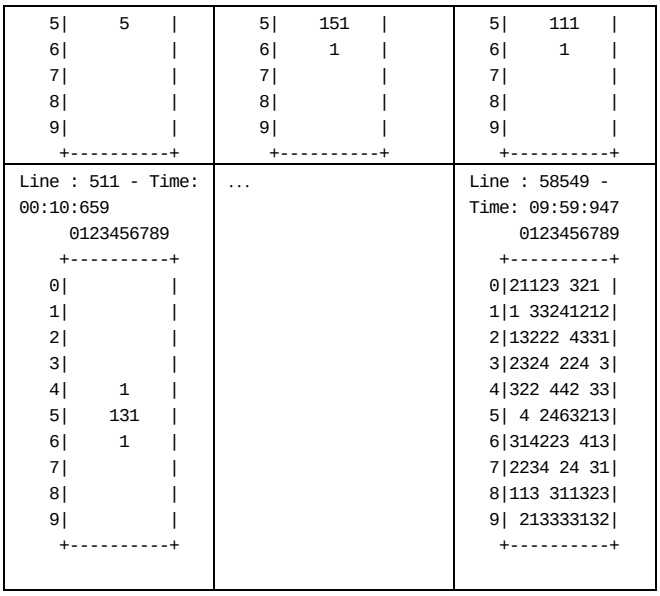

Figure 10: Some Execution results of SandPile model.

6. CONCLUSION

In this paper, we have seen how to model natural landslides using Cell-DEVS formalism. As all material flow between cells is done through changes in cell value, care needed to be taken to guarantee consistency of the model results through the of law mass conservation.

The model behavior is found to be best represented when using Inertial delay. This is because some rules that accumulate sand particles into a cell as Rules 2 and 3, need to be executed only once in one time step. Executing these rules more than once would increment cell's value unnecessarily. Inertial delay is found to solve this problem because if a cell is notified several times in one time step due to changing neighbors, its value is evaluated and incremented only once for all notifications as they all fall in the same time step. In transport delay however, evaluations of cell's value would be made sequentially without preempting previous value, thus accumulating a false value in the cell.

The above problem may also be solved if the Cell-DEVS model allows us to notify neighbors at only some pre-defined states (like a threshold states). All other intermediate states that are not important to neighboring cells can be kept without notifying them. This would be the case when a cell changes its value from 0 to 1,2, or 3. As all these changes do not involve redistribution to other cells, hence, we don't need to notify neighbors with these changes. This would not only enhance efficiency of execution, but would also prevent such errors as described in the previous point.

This model as implemented would need some mechanism to take statistics for getting simulation results. Such a method may be to attach each cell's output to a counter component to count if the cell is distributing and hence an avalanche has started. This counter component can count all cells participating in an avalanche to estimate its severity. Alternatively, the counter could be attached to border cells to count sand particles escaping the board as a measure of avalanche severity. Both ways would need a considerable effort for defining the connections between the counter and the cells especially when the model grows in size.

REFERENCES

[1] WOLFRAM, S. "A new kind of science". Wolfram Media, Inc. 2002.

[2] WAINER, G.; GIAMBIASI, N. "N-Dimensional Cell-DEVS". In *Discrete Events Systems: Theory and Applications*, Kluwer. Vol. 12, No. 1. January 2002. pp. 135-157.

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

[4] MALAMUD, B.; TURCOTTE, D. "Cellular-Automata Models Applied to Natural Hazards", *Earth System Science*, May/June 2000, P. 42-51.

[5] WAINER, G. "CD++: a toolkit to define discrete-event models". G. Wainer. *Software, Practice and Experience*. Wiley. Vol. 32, No.3. pp. 1261-1306. November 2002.