Jul 1st, 12:00 AM

Creating Advanced Fire-Spreading models using the CD++ toolkit

Gabriel Wainer

Creating Advanced Fire-Spreading models using the CD++ toolkit

Gabriel Wainer

Dept. of Systems and Computer Engineering. Carleton University. Ottawa, ON. Canada.
gwainer@sce.carleton.ca

Abstract: recent research efforts focused on the creation of fire spreading models using Cellular models. The Cell-DEVS formalism and the CD++ toolkit allow the user to construct their applications using simple and more intuitive model specification, which can execute with high efficiency using a discrete event approach. It was shown that different environmental applications can be easily faced, allowing the study of complex problems through simulation. The use of formal base improves the development, checking and maintaining phases, facilitating the testing and reuse of their components. The utilization of a discrete event formalism such as these ones can provide better precision and performance. We will present the definition of different models, focusing on how to define such applications using Cell-DEVS methodology.

Keywords: DEVS, Cell-DEVS, Rothermel model, semiempirical models.

1. INTRODUCTION

At present, there is increasing interest in monitoring and predicting forest fires, as this phenomenon destroys important resources. Many forest fires models have been developed to study how the fire spreads under different environmental conditions, focusing on the need for fire fighters to have means for providing rapid and relatively accurate information concerning fire position (Muzy et al. [2005]). Although the use of computer simulation permitted to address this problem to an extent unknown with analytical solutions, real-time simulators for fire spread are still tricky to elaborate due to both fire complexity the volume of data that ecological models have.

Here, we show how the Cell-DEVS formalism (Wainer et al. [2002]) can help the fire modeler, as t his formalism is well suited to solve this kind of applications. Cell-DEVS enables defining cell spaces with explicit timing delays. Each cell is defined as an atomic DEVS model (Zeigler et al. [2000]), and a procedure to couple cells is depicted. DEVS has been used recently for fire modeling by different teams (Vasconcelos et al. [1995], Ntaimo et al. [2004], Muzy et al. [2003], Barros et al. [1998]). The result in these efforts showed that discrete event methods in general and DEVS in particular, present several advantages:

- Computational time reduction: for a given accuracy, the number of calculations decrease
- Seamless integration with models defined with other modeling techniques mapped to DEVS
- Simulation of discrete time models: seen as particular cases of discrete event methods
- Hybrid systems modeling: the discrete event paradigm provides the theory to develop a uniform approach to model and simulate systems with continuous and discrete components.

These efforts required complex software solutions developed by specialized teams, and the activity of the fire modeler during the development of the model is thus restricted. We will discuss how we have provided better mechanisms for model definition based on Cell-DEVS, and how environmental experts can use the techniques to create fire spreading models. We want to achieve higher precision and improved resolution in the results obtained when executing these models, while taking the advantage of current expertise of the fire experts (in Wainer et al. [2001] we reported a gain in development times in using this approach for other fields of application, due to the fact that the modelers can focus in defining smaller portions of a problem and in expressing it using simpler equations, which can be solved easier than the complete system, creating a very precise model). The resulting cellular models will be able to run asynchronously and in parallel, thus improving precision and performance.

2. CELL-DEVS AND CD++

The Cell-DEVS formalism was defined as an extension to Cellular Automata (Wolfram [2002]) combined with DEVS for specification of discrete-event models. Cell-DEVS and the CD++ toolkit (Wainer [2002]) permit defining asynchronous cell spaces with explicit constructions for timing definition. The DEVS formalism provides a framework for the construction of hierarchical modular models, allowing for model reuse, reducing development and testing times. In DEVS, basic models

(called **atomic**) are specified as black boxes, and several DEVS models can be integrated together forming a hierarchical structural model (called **coupled**). In Cell-DEVS, each cell in a cellular model is seen as a DEVS atomic model, and a procedure for coupling cells is defined based on the neighborhood relationship. Only the active cells in the cell space are triggered, independently from any activation period. Each cell of a Cell-DEVS model holds a state variable and a computing function, which updates the cell state by using its present state and its neighborhood. Each cell is described as:

$$TDC = < X,Y,?,N,delay,d,d_{int},d_{ext}, t, ?,D> \quad (1)$$

X defines the external inputs, ***Y*** the external outputs of the model. ***?*** is the cell state definition. ***Delay*** defines the kind of delay for the cell, and ***d*** its duration. A **transport** delay can be associated with each cell, which defers the outputs for the cell. A state change will be discarded if it is not steady during an **inertial** delay. Each cell takes the set of inputs to compute its future state using ***t(N)***. The remaining functions drive the cell's behavior: d_{int} for internal transitions, d_{ext} for external transitions, ***?*** for outputs and ***D*** for the state's duration. A Cell-DEVS coupled model is defined by:

$$GCC= < X_{list}, Y_{list}, X,Y,n,\{t_1,\ldots, t_n\},N,C,B,Z > \quad (2)$$

Here, X_{list} and Y_{list} are input/output coupling lists, used to define the model interface. ***X*** and ***Y*** represent the input/output events. The ***n*** value defines the dimension of the cell space, $\{t_1,\ldots, t_n\}$ is the number of cells in each dimension and ***N*** is the neighborhood set. The cell space is defined by ***C***, together with ***B***, the set of border cells, and ***Z*** the translation function.

A modeler simply has to focus on three basic aspects: dimension (size and shape of the cell space), influences and behavior (Figure 1).

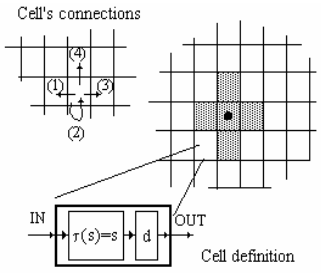

Figure 1. Definition of a Cell-DEVS Model

CD++ implements both DEVS and Cell-DEVS theories. The tool is built as a hierarchy of models, each of them related with a simulation entity. A specification language allows defining the model's coupling. The tool includes an interpreter for a specification language that allows describing Cell-DEVS models, and the models can also be programmed in C++. The behavior specification of a Cell-DEVS atomic model is defined using a set of rules with the form: `POSTCONDITION ASSIGNMENTS DELAY PRECONDITION`. These indicate that when the `PRECONDITION` is satisfied, the state of the cell changes to the designated `POSTCONDITION`, and its output is `DELAY`ed for the specified time. If model's state variables need to be modified, the `ASSIGNMENTS` section can be used (optional). Each 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 (which raise an error condition due to incompleteness of the model). The main operators available to define rules and delays include: Boolean, comparison, arithmetic, neighborhood values, time, conditionals, angle conversion, pseudo-random numbers, error rounding and constants (i.e., gravitation, acceleration, light, Planck, etc.). At present, CD++ is able to execute models in single processor, parallel or real-time mode.

CD++ simulator uses an array of $P_{i=1...n}\ d_i$ to store the states for the cellular model with dimension (d_1, d_2, ..., d_n), and in this case (x_1, x_2,..., x_n) occupies the position $S_{i=1...n}\ x_i$. ($P_{k=1...i-1}\ d_k$). CD++ also permits defining *zones* with differentiated behavior in the cell space. Each zone is defined by a set of cells determined by the cell range {(x_1, x_2, ..., x_n)...(y_1, y_2, ..., y_n)}. Each zone is associated to a different set of rules. Using this capability, different zones into the same cellular model can present different behavior.

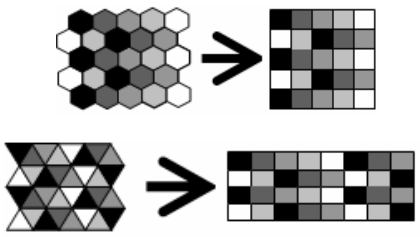

Figure 2. Shift mapping to the square lattice.

In many cases, square topologies are not enough for defining the behavior of advanced cell spaces. Triangular meshes allow covering areas with more varied topology, while permitting every cell to have a limited number of nearby neighbors. Likewise, hexagonal geometries have higher isotropy, that is, the capacity to represent equivalent behavior in every possible direction (which is not the case for square meshes). This is more natural for building the model's rules, and in some cases it is absolutely necessary to simulate the phenomenon upon study. The disadvantage is the difficulty to represent and visualize the model. CD++ Lattice Translator allows defining the cell's behavior based on these topologies, and then they are trans-

lated into square CD++ compatible rules, using the mechanism depicted in Figure 2.

The idea for hexagonal meshes is to use a function that shifts alternate rows in opposite directions, keeping the boundary conditions in the square lattice. The mapping of the triangular lattice is similar: every second cell has a different orientation, and each row of triangles is mapped to one row of square depending on the parity of **x+y**.

3. MODELING FIRE SPREAD

In Muzy et al. [2005], we described a fire model using Cell-DEVS. We will use this example to show the basic rule definition in CD++. The model is based on experimental fires conducted on Pinus pinaster litter (composed of earth and plant matter meshed uniformly with cells of 1 cm².) in a closed room (Balbi et al. [1999]). The energy transferred from the cell to the surrounding air is considered proportional to the difference between the temperature of a cell and the ambient temperature. The heat transferred between a cell and its neighboring cells is represented by a single equivalent diffusion term. The physical model is solved by finite differences:

$$T_{i,j}^{k+1} = aT_{i-1,j}^{k} + aT_{i+1,j}^{k} + bT_{i,j-1}^{k} + bT_{i,j+1}^{k} + cQ\left(\frac{\P S_v}{\P t}\right)_{i,j}^{k+1} + dT_{i,j}^{k} \quad (3)$$

where T_{ij} is the temperature of a grid node. The coefficients a, b, c and d depend on the considered time step and mesh size. Figure 3 represents the specification of this model in CD++.

```
dim : (100,100,2)       border : nowrapped
neighbors   :   (-1,0,0)   (0,-1,0)   (1,0,0)
(0,1,0) (0,0,0) (0,0,-1) (0,0,1)
zone : ti { (0,0,1)..(99,99,1) }
localTransition : FireBehavior

[ti]
rule:{ time/100 } 1 { cellpos(2)=1 AND
         (0,0,-1)>=573 AND (0,0,0) = 1.0 }

[FireBehavior]
rule: {#unburned} 1 {(0,0,0)<300 AND(0,0,0)
!=209 AND (#unburned>(0,0,0) OR time<=20)}
rule: {#burning} 1 {cellpos(2)=0 AND
 ( (0,0,0)>#burning AND (0,0,0)>333 ) OR
  (#burning>(0,0,0) AND (0,0,0)>=573)) AND
    (0,0,0)!=209 } %Burning
rule: {209} 1 {(0,0,0)<=60 AND
 (0,0,0)!=209 AND (0,0,0)>#burning} %Burned

#unburned =(0.98689*(0,0,0) + 0.0031*(
(0,-1,0)+(0,1,0)+(1,0,0)+(-1,0,0))+0.213 )
#burning= (0.98689*(0,0,0)+.0031*(
 (0,-1,0)+(0,1,0)+(1,0,0)+(-1,0,0))+ 2.74 *
    exp(-.19*((time+1)*.01-(0,0,1)))+.213)
```

Figure 3. Fire spread model specification.

We first define the Cell-DEVS coupled model (neighborhood, dimension, etc.). Then, the *ti* rules show how to store ignition times: if a cell in plane 0 starts to burn (*cellpos(2)=1*), we record the current simulation time in plane 1. Then, we show the rules used to compute the cells' temperatures. The macros show the rules corresponding to the temperature calculus: cells can be inactive, unburned, burning and burned. An unburned cell's temperature is lower than 573 degrees. A cell starts burning at 573 degrees and its temperature increases for a while; then it start decreasing as the fuel mass is consumed. When the temperature gets lower than 333 degrees, the cell enters the burned phase (signaled by a constant temperature of 209 degrees). The first rule in Figure 3 applies to unburned cells, whose temperature in the next step will be higher than its current one. The second rule applies to burning cells. The third rule sets the burned flag (temperature=209 degrees) when a burning cell crosses down the 333-degrees threshold, and the fourth rule keeps the burned cells constant.

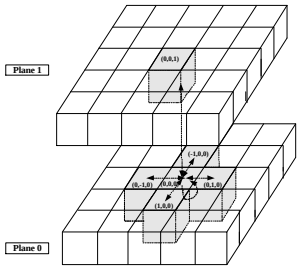

Figure 4. Cell's neighborhood specification

As described in Figure 4, we use two planes to representing the model: the first one stores the cell's temperature, and the second one stores the ignition time. This model accurately reproduces the experimental results obtained in the lab. As we can see in Figure 4, we need n *x* m *x* 2 cells (double the size of the simulated area). This model can be redefined using multiple state variables. In this case, the *temperature* and the *ignition time* are stored in independent a state variables. The model can be redefined as follows (López et al. [2004]).

```
stateVariables: ti       stateValues: 0

[FireBehavior]
rule : { #unburned } 1 { (0,0)!=209
   AND (0,0)<573 AND ( time<=20 OR
       #unburned>(0,0) ) }
rule : { #burning } 1 { (0,0)>333 AND
   ( (0,0)<573 OR $ti != 1.0 ) AND
       (0,0)>#burning }
rule : { #burning }{ $ti := if($ti = 1.0,
time / 100, $ti); } 1
   { (0,0) >= 573 AND #burning >= (0,0) }
rule : { #burning } { $ti := time/100; }
   1 { $ti = 1.0 AND (0,0) >= 573 AND
        #burning < (0,0) }
rule : { 209 } 100 { (0,0) != 209 AND
    (0,0)<=333 AND (0,0) > #burning }
```

Figure 5. Fire spread model specification.

The first step was to add a state variable *ti* to remove one layer of cells and to replace all the references to this layer by references to the state vari-

able. The model does not use multiple planes as in the previous case; therefore, we do not need to check the plane we are using. Simultaneously, the references are always to 2D cells. The *ti* rule records the moment when the cell starts burning. In this case, we use a new state variable instead of an independent plane like in the original model presented in. As these rules are more compact, we can manipulate them to obtain better performance and easier understanding.

This problem can also be solved using multiple ports to replace the extra plane. When we use multiple ports we do not need to store internally the values, but to transmit them through the ports. So, there is not need to set values, but just send them out though the corresponding port. In this case, two ports are declared: *temp* and *ti*. The port *temp* exports the cell's temperature, while the port *ti* exports the ignition time.

```
rule: {~temp:=#burning} 1 { (0,0)~temp>333
   AND ( (0,0)~temp<573 OR (0,0)~ti!=1 )
      AND (0,0)~temp > #burning }
rule: {#burning} 1 {(0,0)>333 AND (0,0)<573
   OR $ti!=1 AND (0,0)>#burning }
rule: {#burning} {$ti:=if($ti=1,time/100,
  $ti)} 1 {(0,0)>=573 AND #burning>=(0,0) }
rule : {#burning} {$ti:=time/100; } 1 {
   $ti=1 AND (0,0)>=573 AND #burning<(0,0)}
```

These two new versions of the model behave exactly the same as the original, but with clear gains in the modeling itself, which permits a user to describe more complex phenomena easily. Likewise, the different optimizations presented allowed us to obtain gains in execution times of up to 40% by just reordering and factoring the rules to execute more efficiently using CD++ evaluation mechanism. The following figure shows a 3D version of the execution results for this model using CD++/Maya (Khan et al. [2005]). These visualizations can be easily expanded to include terrain and climate information, which would be useful for training, online visualization and decision making.

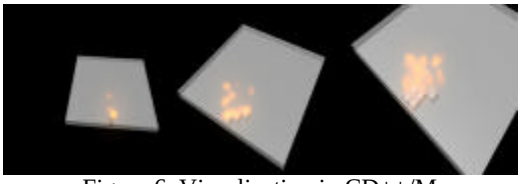

Figure 6. Visualization in CD++/Maya

Another advantage is that the complexity of this physical phenomenon is such that the inclusion of other external influences is difficult to be considered. Cell-DEVS allows including new rules easily, allowing evolvability of the model. For instance, we also defined an advanced Cell-DEVS fire model based on a well known model for fire propagation in forests due to Rothermel [1972], in which we can see how to make use of the explicit time delay functions to improve model definition. This model uses 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). 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 topography and the dimensions of the cellular space to obtain the spread ratio in every direction. Instead of using a time-based approach, the model uses the delay function to compute fire spread. Figure 7 shows the implementation of this model using a hexagonal mesh.

```
dim : (20,20)  delay : inertialneighbors :
(-1,-1) (-1,0) (-1,1) (0,-1)  (0,0)  (0,1)
(1,-1)  (1,0)  (1,1)

[FireBehavior]
rule: {[5]+(15.24/13.680)} {(15.24/13.680)
   * 60000} {[0]=0 and [5]!=? and [5]>0}
rule: {[6]+(15.24/5.10)} {(15.24 /5.106 )
   * 60000} {[0]=0 and [6]!=? and [6]>0}
rule: {[4]+( 15.24/2.950)} {(15.24/2.950)
   * 60000} {[0]=0 and [4]!=? and [4]>0}
rule: {[1]+(15.24/1.630)} {(15.24/ 1.630)
   * 60000} {[0]=0 and [1]!=? and [1]>0}
rule: {[3]+(15.24/1.146)} {( 15.24 / 1.146)
   * 60000} {[0]=0 and [3]!=? and [3]>0}
rule: {[2]+(15.24/1.040)} {( 15.24/ 1.040)
   * 60000} {[0]=0 and [2]!=? and [2]>0}
```

Figure 7. Rothermel's fire forest model.

The rules defining the local computing function are devoted to detect the presence of fire in the eight neighboring cells. For instance, the first rule checks if the current cell is not burning ([0]= 0) and if the SW neighbor has started to burn ([5]>0). If this condition holds, the new value of the cell will be `[5]+(15.24/13.680)`, which is the time the fire will start in the cell. We use a delay of `(15.24/13.680) * 60000` ms after which the present cell state will spread to the neighbors. The remaining rules represent a similar behavior for the remaining neighbors. The results of running this model are shown below.

As we can see, we use a different notation to represent each one of the 6 neighbors ([1…6], in counter clockwise direction starting at 0°). In the hexagonal lattice, the distance between two neighbor cells is the same in every direction, so we use a distance of 15.24m for all of the rules.

Using a triangular lattice, we obtain these rules:

```
[FireBehavior]
rule: {[3]+(4.40/5.106)} {(4.40/5.106)
  * 60000} {[0]=0 and [3]!=? and [3]>0 and
     odd(cellpos(0)+cellpos(1))}
rule: {[1]+(4.40/2.950)} {(4.40/2.950)
   * 60000} {[0]=0 and [1]!=? and [1]>0
     and odd(cellpos(0)+cellpos(1))}
rule: {[2]+(4.40/1.040)} {(4.40/1.040)
  * 60000} {[0]=0 and [2]!=? and [2]>0 and
     odd(cellpos(0)+cellpos(1))}
rule: {[3]+(4.40/8.573)} {(4.40/ 8.573)
   * 60000} {[0]=0 and [3]!=? and [3]>0 and
     even(cellpos(0)+cellpos(1)}
rule: {[2]+(4.40/1.630)} {(4.40/1.630)
   * 60000} {[0]=0 and [2]!=? and [2]>0 and
     even(cellpos(0)+cellpos(1)}
rule: {[1]+( 4.40/1.146)} {(4.40/1.146)
   * 60000} {[0]=0 and [1]!=? and [1]>0 and
     even(cellpos(0)+cellpos(1)}
```

Figure 8. Rules using triangular topology

In this case, there are six rules because we need to do rules for even triangles and odd triangles. These models are translated into a square grid, as showed earlier in Figure 8. CD++ can display these topologies, as showed in the following figure.

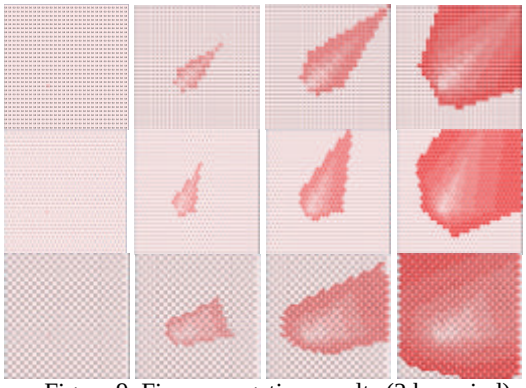

Figure 9. Fire propagation results (2 h. period) (a) square lattice (b) hexagonal lattice (b) triangular lattice

As we can see, the burning time of a cell depends on the spread ratio in the direction of the burning cell. Changes in the propagation here are related to the changes produced by the adjacency properties derived from using different topologies. 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. Cellular Automata model these systems with a similar approach than the Finite Elements method (Fröier et al. [1974]. Instead, with Cell-DEVS, a non burning cell in the direction of the fire spread will be updated in a shorter period of time than one 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.

Fire suppression can be easily implemented. In Figure 10, we show the implementation of a rain front moving to the SE, extinguishing the fire on burning cells. This behavior is implemented in the following rules, which were added to the previous model. 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.

```
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 10. Forest fire. Rules defining rain.

Figure 11 shows the execution of this model using hexagonal topology. The initial behavior is similar to the one seen in Figure 9. We then observe the advance of rain, which cools the fire areas (light gray), and finally we can see how 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.

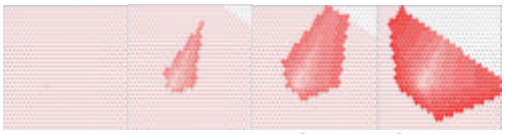

Figure 11. Fire evolution with rain.

The following extension allows analyzing fire suppression by firefighters. 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. 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.

```
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 12. Rules defining firefighter behavior.

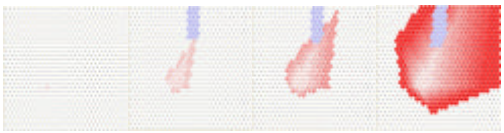

Figure 13. Fire evolution with firefighters.

Figure 13 shows how firefighters spread coolant from N to S, and while fire spreads (as in figure 9), firefighter zones cooled down (light gray), while in some areas fire has been extinguished.

The use of varied topologies improves execution performance. The following figure shows the number of messages involved in executing the model using the different topologies.

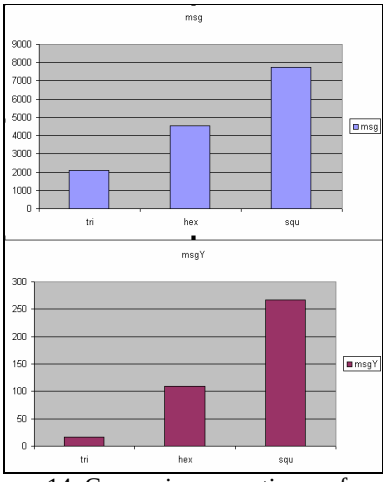

Figure 14. Comparing execution performance.

As we can see, the triangular and hexagonal topologies use a smaller size for the neighborhood, highly reducing the number of messages needed to carry out the simulation, and improving execution.

4. CONCLUSION

We have showed hot to apply the Cell-DEVS formalism for the construction of advanced models in the field fire spreading. Cell–DEVS allows describing physical and natural systems using an n-dimensional cell-based formalism. Complex timing behavior for the cells in the space can be defined using very simple constructions. CD++ simplifies the construction of these models by allowing simple and intuitive model specification. The use of formal base improves the development, checking and maintaining phases, facilitating the testing and reuse of their components. The evolution of propagation models is eased by the hierarchy description of DEVS and the high-level language of Cell-DEVS. Cell-DEVS discretization allows us to activate only the most heated cells of the fire front. The utilization of a discrete event formalism such as this one can provide better precision and performance, while different visualization tools, simulators and an integrated development environment can enhance the modeling experience.

REFERENCES

Balbi, J., Santoni, P. and Dupuy, J. Dynamic modelling of fire spread across a fuel bed. *International Journal of Wasteland Fire*. (9), 275-284, 1999.

Barros, F., Ball, G.L. Fire modelling using dynamic structure cellular automata. III International Conference On Forest Fire Research. 14th Conference on Fire and Forest Meteorology. Luso, Portugal. 1998.

Fröier, M., Nilsson, L. and Samuelsson, A. The rectangular plane stress element by Turner, Pian and Wilson. *International Journal for Numerical Methods in Engineering*. 8(2), 433-437, 1974.

Khan, A., Venhola, W. and Wainer, G.; Jemtrud, M. Advanced DEVS model visualization. Proceedings of IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation 2005. Paris, France. 2005.

López, A. and Wainer, G. Improved Cell-DEVS model definition in CD++. In Proceedings of Sixth International conference on Cellular Automata for Research and Industry. Amsterdam, Netherlands. LNCS Vol. 3305. 2004.

Muzy, A., Innocenti, E., Hill, D., Santucci, J.F. Optimisation of cell spaces simulation for the modelling of fire spreading. Proc. of 36th Annual Simulation Symposium, Orlando, FL. 2003.

Muzy, A., Wainer, G., Innocenti, E., Aiello, A. and Santucci, J.F. Cellular Discrete-event modeling and simulation of fire spreading across a fuel bed. *Simulation: Transactions of the Society for Modeling and Simulation International*. (81), 2. 103-117. 2005.

Ntaimo, L., Khargharia, B., Zeigler, B. and Vasconcelos, M. Forest fire spread and suppression in DEVS. *Simulation, Transactions of the SCS*. (80), 10, 479-500. 2004.

Rothermel, R.C. A Mathematical Model for Predicting Fire Spread in Wasteland Fuels. USDA Forestry Service Research Paper, INT-115. 1972.

Vasconcelos, M., Pereira, J. and Zeigler, B. Simulation of fire growth using discrete event hierarchical modular models. *EARSeL Advances in Remote Sensing*. (4), 3, 54-62. 1995.

Wainer, G. CD++: a toolkit to define discrete-event models. *Software, Practice and Experience*. (32), 3, 1261-1306. 2002.

Wainer, G. and Giambiasi, N. Application of the Cell-DEVS paradigm for cell spaces modeling and simulation. *Simulation* (76), 1. 22-39. 2001.

Wainer, G. and Giambiasi, N. N-dimensional Cell-DEVS. *Discrete Events Systems: Theory and Applications*. (12), 1, 135-157. 2002.

Wolfram, S. *A New Kind of Science*. Wolfram Media. 2002.

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