DYNAMIC STRUCTURE CELLULAR AUTOMATA IN A FIRE SPREADING APPLICATION

Alexandre Muzy, Eric Innocenti, Antoine Aïello, Jean-François Santucci, Paul-Antoine Santoni
University of Corsica
SPE – UMR CNRS 6134
B.P. 52, Campus Grossetti, 20250 Corti. FRANCE.
Email: a.muzy@univ-corse.fr

David R.C. Hill
ISIMA/LIMOS UMR CNRS 6158
Blaise Pascal University
Campus des Cézeaux BP 10125, 63177 Aubière Cedex. FRANCE.
Email: drch@isima.fr

Keywords: Systems modeling, Discrete event systems, Multi-formalism, Fire spread, cellular automata.

Abstract: Study of complex propagation phenomena is performed through cellular simulation models. Usually cellular models are specific cellular automata developed by non-computer specialists. We attempt to present here a mathematical specification of a new kind of CA. The latter allows to soundly specify cellular models using a discrete time base, avoiding basic CA limitations (infinite lattice, neighborhood and rules uniformity of the cells, closure of the system to external events, static structure, etc.). Object-oriented techniques and DES are used to achieve this goal. The approach is validated in a fire spreading application.

1 INTRODUCTION

When modeling real systems, scientists cut off pieces of a biggest system that is the world surrounding us. Global understanding of that world necessitates connecting all these pieces (the subsystems), referencing some of them in space. When the whole system is complex, the only way to study its dynamics is the simulation.

Propagation phenomena as fire, swelling, gas propagation (…), are complex systems. Studying these phenomena generally leads to divide the propagation space in cells thus defining a cellular system.

Developed from the General Systems Theory (Mesarovic and Takahara, 1975), the Discrete Event Structure Specification (DEVS) formalism (Zeigler et al., 2000) offers a theoretical framework to map systems specifications into most classes of simulation models (differential equations, asynchronous cellular automata, etc.). For each class of model, one DEVS sub-formalism allows to faithfully specify one simulation model. As specification of complex systems often needs to grasp different kinds of simulation models, connections between the models can be performed using DEVS multi-formalism concepts.

Another DEVS advantage relates to its ability in providing discrete event simulation techniques, thus enabling to concentrate only on components of interest during the simulation. Components are activated, or can send a message to be activated whenever during the simulation thus improving performances and precision of the simulation.

Precise and sound definition of propagation needs to use models from physics as Partial Differential Equations (PDEs). These equations are then discretized leading to discrete time simulation models. These models are generally simulated from scientists by using specific Cellular Automata (CA). As defined in (Wolfram, 1994), standard CA are simple mathematical idealizations of natural systems. They consist of an infinite lattice of discrete identical sites, each site taking on a finite site of, say, integer values. The values of the sites evolve in discrete time steps according to deterministic rules that specify the value of each site in terms of the values of neighboring sites. CA may thus be considered as discrete idealizations of PDEs. CA are models where space, time and states are discrete (Jen, 1990).

However, definition of basic CA is too limited to specify complicated simulations (infinite lattice, neighborhood and rules uniformity of the cells, closure of the system to external events, discrete state of the cells, etc.). Scientists often need to modify CA's structure for simulation purposes (Worsch, 1999).

We extend here basic CA capabilities by using object-oriented techniques and discrete event simulation (Hill, 1996). A mathematical specification of the approach is defined using the Dynamic Structure Discrete Time System Specification (DSDTSS) formalism (Barros, 1997). This formalism allows to dynamically change the structure of discrete time systems during the simulation. These new CA are called the Dynamic Structure Cellular Automata (DSCA).

DSCA have been introduced in (Barros and Mendes, 1997) as a formal approach allowing to dynamically change the network structure of asynchronous CA. Using an asynchronous time base, cells were dynamically instantiated or destroyed during a fire spread simulation.

The scope here is to extend basic CA capabilities using a discrete time base. If previous DSCA were dedicated to discrete event cellular system specification, we extend here the DSCA definition to discrete time cellular system specification.

Table 1 briefly sums up the advantages of the DSCA in relation to basic CA. In DSCA, each cell can contain different behaviors, neighborhoods and state variables. Rules and neighborhoods of the cells can dynamically change during the simulation. Each cell can receive external events. Only active cells are computed during the simulation. Finally, a global transition function allows to specify global behaviors of the DSCA.

	Basic CA	DSCA
Time	discrete	discrete
Space	discrete	discrete
State	discrete	continuous
Closure to external events	-	+
Different variables per cell	-	+
Rules uniformity*	-	+
Neighborhood uniformity*	-	+
Activity tracking*	-	+
Global function	-	+

**during the simulation*

Table 1: DSCA extensions

DSCA definition is validated against a fire spreading application. Recent forest fires in Europe (Portugal, France and Corsica) and in United-States (California) unfortunately pinpoint the necessity of increasing research efforts in this domain. Fires are economical, ecological and human catastrophes. Especially as we know that present rising of wild land surfaces and climate warming will increase forest fires.

Modeling such a huge and complex phenomenon obviously leads to simulation performance over loadings and design problems. Simulation model reusability has to face to the complicated aspect of both model implementations and model modifications. Despite a large number of cells, simulation has to respect real time deadlines to predict actual fire propagation. Hence, this kind of simulation application provides a powerful validation to our work.

This study is organized as follows. First some formalisms background is provided. Then two sections present the DSCA modeling and simulation principles. After, simulation results of a fire spreading application are provided. Finally, we conclude and make some prospects.

2 BACKGROUND

A formalism is a mathematical description of a system allowing to guide a modeler in the specification task. The more a formalism fits to a system class the more simple and accurate it will be.

Efficiently modeling complex systems need to define subsystems using different formalisms. Connections between the different formalisms can then be achieved through a multi-formalism to perform the whole system specification.

In this study, subsystems are specified using DEVS, DTSS (Discrete Time System Specification) and DSDTSS formalisms. Connections between the different models are achieved using a Multi-formalism Network (*MFN*). A structure description of each model is provided here after.
A DEVS atomic model is a structure:

$$DEVS = (X, Y, Q, q_0, \delta_{int}, \delta_{ext}, \lambda, t_a)$$

where X is the input events set, Q is the set of state, q_0 is the initial state, Y is the output events set, δ_{int}: $Q \rightarrow Q$ is the internal transition function, δ_{ext}: $Q \times X \rightarrow Q$ is the external transition function, $\lambda : Q \rightarrow Y$ the output function, t_a is the time advance function.

DSDTSS basic models are DTSS atomic model:

$$DTSS = (X, Y, Q, q_0, \delta, \lambda, h)$$

where X, Y are the input and output sets, Q is the set of state, q_0 is the initial state, $\delta : Q \times X \rightarrow Q$ is the state transition function, $\lambda : Q \rightarrow Y$ is the output function (considering a Moore machine) and h is a constant time advance.

At a periodic rate, this model checks its inputs and, based on its state information, produces an output and changes its internal state.

The network of simple DTSS models is referred to as a Dynamic Structure Discrete Time Network (DSDTN). We introduce here input and output sets to allow connections with the network. Formally, a DSDTN is a 4-tuple:

$$DSDTN = (X_{DSDTN}, Y_{DSDTN}, \chi, M_\chi)$$

where X_{DSDTN} is the network input values set, Y_{DSDTN} is the network input values set, χ is the name of the DSDTN executive, M_χ is the model of the executive χ.

The model of the executive is a modified *DTSS* defined by the 8-tuple:

$$M_\chi = (X_\chi, Q_\chi, q_{0,\chi}, Y_\chi, \gamma, \Sigma^*, \delta_\chi, \lambda_\chi)$$

where $\gamma : Q_\chi \rightarrow \Sigma^*$ is the structure function, and Σ^* is the set of network structures. The transition function δ_χ computes the executive state q_χ. The network executive structure Σ, at the state $q_\chi \in Q_\chi$ is given by $\Sigma = \gamma(q_\chi) = (D, \{M_i\}, \{I_i\}, \{Z_{i,j}\})$, for all $i \in D$, $M_i = (X_i, Q_i, q_{0,i}, Y_i, \delta_i, \lambda_i)$, where D is the set of model references, I_i is the set of influencers of model i, and $Z_{i,j}$ is the i to j translation function.

Because the network coupling information is located in the state of the executive, transition functions can change this state and, in consequence, change the structure of the network. Changes in structure include changes in model interconnections, changes in system definition, and the addition or deletion of system models.

Formally, a multiformalism network is defined by the 7-tuple:

$$MFN = (X_{MFN}, Y_{MFN}, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, select)$$

where $X_{MFN}=X^{discr}\times X^{cont}$ is the network input values set, X^{discr} and X^{cont} are discrete and continuous input sets, $Y_{MFN}=Y^{discr}\times Y^{cont}$ is the network input values set, Y^{discr} and Y^{cont} are discrete and continuous output sets, D is the set of model references,

For each $i \in D$,

M_i is are DEVS, DEVN, DTSN, DTSS, DESS, DEV&DESS or other MFN models. As DSDTSS proved to be closed under coupling, M_i can also be dynamic structure models or networks,

I_i is the set of influencers of model i,
$Z_{i,j}$ is the i to j translation function,
select is the tie-breaking function.

3 DSCA MODELLING

Models composing a DSCA are specified here using the previous models' definition. As described in the modeling part of Figure 1, external events are simulated using a DEVS atomic model: the *Generator*. The latter can asynchronously generate data information to the DSCA during the simulation. The cell space is embedded in a DSDTN. Each cell is defined as a DTSS model. Using its transition function, the DSCA executive model (containing every cells) achieves changes in structure directly accessing to the cells' attributes. A mathematical description of each model is provided here after.

We define the *MFN* by the structure:

$$MFN = (X_{MFN}, Y_{MFN}, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, select)$$

where $D=\{G, DSDTN\}$, $M_G = (X_G, Q_G, q_{0,G}, Y_G, \delta_G, \lambda_G, \tau_G)$, $M_{DSDTN}=DSDTN$, $I_G=\{\}$, $I_{DSDTN}=\{G\}$, and $Z_{DSDTN,MFN}: Y_{DSDTN} \rightarrow Y_{MFN}$, $Z_{G,DSDTN}: Y_G \rightarrow X_{DSDTN}$.

We define the *DSDTN* by the structure:

$$DSDTN = (X_{DSDTN}, Y_{DSDTN}, DSCA, M_{DSCA})$$

where $\Sigma = \gamma(q_{0,\chi}) = (D, \{M_i\}, \{I_i\}, \{Z_{i,j}\})$, *where* $D=\{(i,j) \ / \ (i,j) \in \Im^2\}$, $M_{DSCA} = (X_{DSCA}, Q_{DSCA}, q_{0,DSCA}, Y_{DSCA}, \delta_{DSCA}, \lambda_{DSCA})$, $I_{DSCA}=\{DSDTN\}$, $I_{cell}=\{cell^n, DSDTN\}$. Where $I_{cell} = \{I_{kl} \ / \ k \in [0,m],\ l \in [0,n]\}$ is the neighbourhood set (or the set of influencers) of the cell as defined in (Wainer and Giambiasi, 2001). It is a list of pairs defining the relative position between the neighbours and the origin cell. $I_{kl} = \{(i_p,j_p) \ / \ \forall \ p \in I_{cell},\ p \in [1,\eta_{kl}],\ i_p,j_p \in Z\ ;\ |k-i_p| \geq 0 \ \wedge\ |l-j_p| \geq 0 \ \wedge\ \eta_{kl} \in I_{cell}\}$, and $\eta \in I_{cell}$ is the neighborhood size.

$Z_{cell,DSCA}$: $Y_{cell} \rightarrow Y_{DSCA}$, $Z_{DSCA,DSDTN}$: $Y_{DSCA} \rightarrow Y_{DSDTN}$, $Z_{DSDTN,DSCA}$: $X_{DSDTN} \rightarrow X_{DSCA}$, $Z_{DSCA,cell}$: $X_{DSCA} \rightarrow X_{cell}$.

For the implementation, ports are defined:

$Py_G=Px_{DSDTN}=Px_{DSCA}=\{data\}$
$Py_{DSCA}=Py_{DSDTN}=Py_{MFN}=\{state\}$
$Px_{cell}=Py_{cell}=\{(i,j)\}$

We specify each cell as a special case of DTSS model:

$$cell=(X_{cell}, Q_{cell}, q_{0,cell}, Y_{cell}, \delta_{cell}, \lambda_{cell})$$

where X_{cell} is an arbitrary set of input values, Y_{cell} is an arbitrary set of output values, $q_{0,cell}$ is the initial state of the cell and

$q \in Q_{cell}$ is given by:

$q=((i,j), state, N, phase)$,
$(i,j) \in \Im^2$, is the position of the cell,
state is the state of the cell,
$N = \{N_{kl} \ / \ k \in [0,m],\ l \in [0,n]\}$. N is a list of states N_{kl} of the neighboring cells of coordinates (k,l),
phase = {passive, active} corresponds to the name of the corresponding dynamic behavior. For numerous adjacent active cells, the *active* phase can be decomposed in '*testing*' and '*nonTesting*' phases. The use of these phase is detailed in section *5*.

$\delta_{cell} : Q_{cell}\times X_{cell} \rightarrow Q_{cell}$
$\lambda_{cell} : Q_{cell} \rightarrow Y_{cell}$

4 DSCA SIMULATION

As depicted in Figure 2, implementation of discrete event models consists in dividing model's transition function δ_d according to event types ev_n of a set of possible event types S_d. The transition function then depends on the event types the model receives.

```
Model d
     Sd = {ev1, ev2, … , evn}
     δd (Sd)
          case Sd
          ev1 : call event-routine1
          ev2 : call event-routine2…
          evn : call event-routinen
```

Figure 1: Discrete event model implementation (Zeigler et al., 2000)

The DSCA receives data from the *Generator*. These data represent external influences of the DSCA. During the simulation, information is embedded in messages and transits through the *data* and *state* ports. Messages have fields *[Message type, Time, Source processor, Destination port, Content]*, where *Content* is a vector of triplets *[Event type, Value, Coordinate port]*. When a DSCA receives a message on its *data* port, it scans the *Content* vector and according to the *Coordinate port*, sends the *[Event type, Value]* pairs to the concerned cells. A vector of pairs *[Event type, Value*] can be sent to one cell's port. Then, according to the *Event type* a cell receives, it will update the concerned attributes, executing the concerned transition function.

The simulation tree hierarchy is described in the simulation part of Figure 2. Except for the *Root* and *DTSS interface,* all nodes of the tree are processors attached to models. The processors manage with message exchanges and execution of model functions. Each processor is automatically generated when the simulation starts.

The *Coordinator* pilots the *MFN* model, the simulator *SimG* pilots the *Generator*, the *DSDTN* model and the *DSDTN* models are piloted by the Aggregated Network Simulator (*AN Simulator)*. Algorithms of the DSCA simulators can be found in (Muzy et al., 2003), whereas basic DEVS simulators and DTSS interface can be found in (Zeigler et al., 2000).

The Root processor supervises the whole simulation loop. It updates the simulation time and activates messages at each time step. For the Coordinator, the DTSS interface makes the Aggregated Network simulator seen as a DEVS atomic simulator. This is done by storing all messages arriving at the same time step and then by calculating the new state and output of the DSCA when receiving an internal transition message. The simulation tree thus respects the DEVS bus principle. That means that whatever DEVS model can be appended to the simulation tree.

5 Activity tracking

Using discrete event cellular models, activity tracking can be easily achieved (Nutaro et al., 2003). Active cells send significant events to be reactivated or to activate neighbors at next time step. However, pure discrete event models proved to be inefficient for discrete time system simulation (Muzy et al., 2002). Interface configurations and message management produce simulation overhead, especially for numerous active components.

For discrete time systems, we know that each component will be activated at each time step. Moreover, in CA cells' state directly depends on its neighbors' state. To optimize simulation, messages between the cells have to be canceled and simulation time advance has to be discrete. However, a new algorithm has to be defined to track active cells.

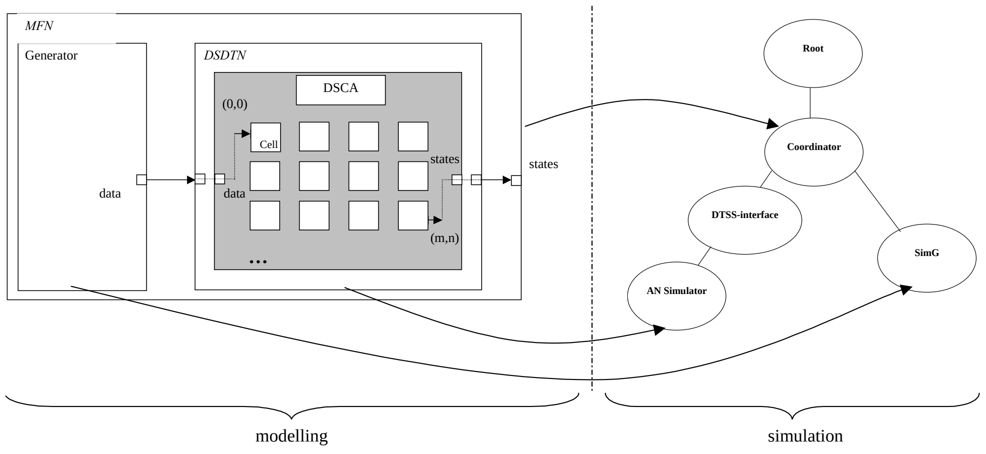


Figure 2: DSCA modeling and simulation

To focus simulation on active cells, we use the basic principles exposed in (Zeigler et al., 2000) to predict whether a cell will possibly change state or will definitely be left unchanged in a next global state transition: *a cell will not change state if none of its neighboring cells changed state at the current state transition time*.

Nevertheless, to obtain optimum performance the entire cells cannot be tested. Thereby, an algorithm, which consists in testing only the neighborhood of the active bordering cells of a propagation domain, has been defined for this type of phenomena.

To be well designed, a simulation model should be structured so that all information relevant to a particular design can be found in the same place. This principle enhances models modularity and reusability making easier further modifications.

Pure discrete event cells are all containing a micro algorithm, which allows to focus the whole simulation loop on active cells. We pinpointed above the inefficiency of such an implementation for discrete time simulation. An intuitive and efficient way to achieve activity tracking in discrete time simulation is specifying this particular design at one place. As depicted in Figure 3, the activity tracking algorithm is located in the global transition function of the DSCA, in charge of the structure evolution of the cell space.

Cells are in phases '*testing'* when located at the edge of the propagation, '*nonTesting'* when not tested at each state transition and '*quiescent'* when inactive.

A propagation example is sketched in Figure 4 for cardinal and adjacent neighborhoods. In our algorithm, only the bordering cells test their neighborhood, this allows to reduce the number of testing cells.

The result of the *spreadTest(i,j,nextState)* function of Figure 3, depends on the state of the tested cells. If this state fulfils a certain condition defined by the user, the cell becomes '*nonTesting*' and new tested neighboring are added to the set of active cells. The transition function receives x_χ messages from the Generator corresponding to external events. The x_χ messages contain the coordinates of the cells influenced by the external event. If the coordinates are located in the domain calculation, the cells' state is changed. Otherwise, the new cells are added to the propagation domain.

```
//'Q' is for the quiescent phase,
//'T' for the testing one and 'N' for
//the nonTesting one

Transition Function(xχ)
 For each cell(i,j) Do
  If(cellPhase(i,j)=='Q') Then
   removeCell(i,j)
  endIf

  If(cellPhase(i,j)=='T') Then
   If (cellNearToBorder(i,j)) Then
    setSpreadStateCell(i,j,'N')
   Else
    If(spreadTest(i,j,nextState))Then
     setCellPhase(i,j, 'N')
     addQuiescentNeighboringCells(i,j)
    endIf
   endIf
  endIf
 endFor

 if(xχ message is not empty)
  if(cells in the propagation domain)
   changeCellStates()
  else
   addNewCells()
  endIf
 endIf

endTransitionFunction()
```

Figure 3: Transition function of the DSCA

For efficiency reasons, the simulation engine we developed has been implemented in C++ and dynamic allocation has been suppressed for some classes. Indeed, for significant numbers of object instantiation/deletion dynamic allocation is inefficient and we have designed a specialized static allocation (Stroustrup, 2000). A pre-dimensioning via large static arrays can be easily achieved thanks to current modern computer memory capabilities.

The state of the executive model is a matrix of cellular objects. References of the active cells are stored in a vector container. A start-pointer and an end-pointer are delimiting the current calculation domain on the vector. Thus initial active cells that are completely burned during a simulation run can be dynamically ignored in the main loop. At each time step, by modifying the pointers position new tested cells can be added to the calculation domain and cells that return in a quiescent state are removed from the former.

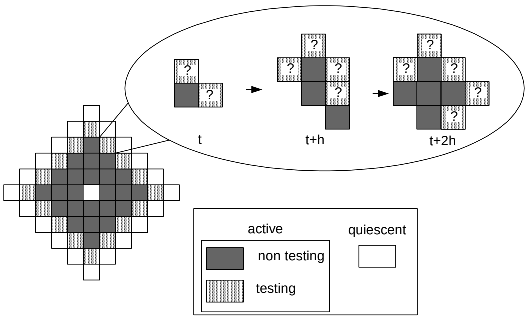


Figure 4: Calculation domain evolution

6 FIRE SPREADING APPLICATION

The simulation engine we used has been proved to achieve real-time simulation (Muzy et al., 2003). We used a mathematical fire spread model already validated and presented in (Balbi et al., 1998). In this model, a Partial Differential Equation (PDE) represents the temperature of each cell. A CA is obtained after discretizing the PDE. Using the finite difference method leads to the following algebraic equation:

$$T_{i,j}^{k+1} = a(T_{i-1,j}^{k} + T_{i+1,j}^{k}) + b(T_{i,j-1}^{k} + T_{i,j+1}^{k}) + \sigma_{v,0} e^{-\alpha(t-t_{ig})} + dT_{i,j}^{k} \qquad (1)$$

where T_{ij} is the grid node temperature. The coefficients *a*, *b*, *c* and *d* depend on the time step and mesh size considered, *t* is the real time, t_{ig} the real ignition time (the time the cell is ignited) and $\sigma_{v,0}$ is the initial combustible mass**.**

Figure 5 depicts a simplified temperature curve of a cell in the domain. We consider that above a threshold temperature T_{ig} the combustion occurs and below a T_f temperature the combustion is finished.

The end of the real curve is purposely neglected to save simulation time. Four states corresponding to each cell's behavior are defined from these assumptions. A cell has states '*unburned'*, *'heating'*, '*onFire'* and '*burned'*.

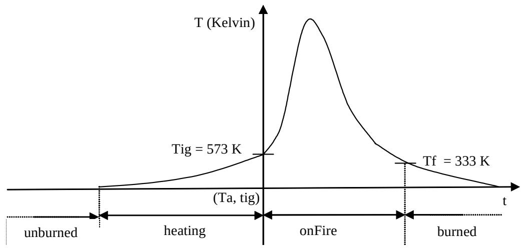


Figure 5: Simplified temperature curve of cells' behavior

Figure 6 depicts a fire spreading in a Corsican valley, generated using the OpenGL graphics library. Looking at this picture we easily understand that simulation has to focus only on a small part of the whole land. Actually, areas of activity just correspond to the fire front, and the cells in front of the latter, that is the '*heating*' and '*onFire'* cells.

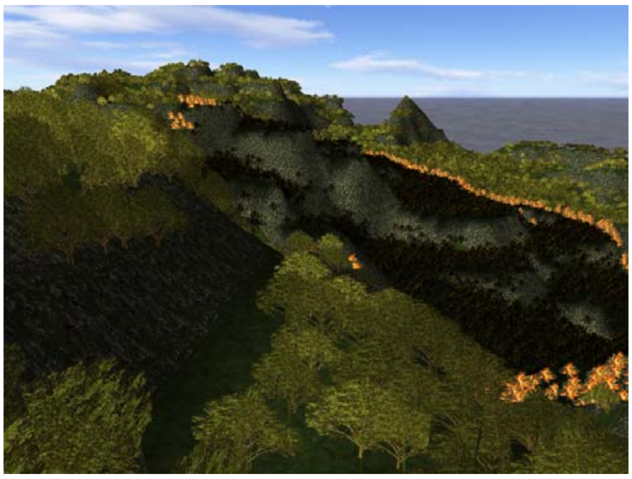


Figure 6: 3D Visualization of fire spreading

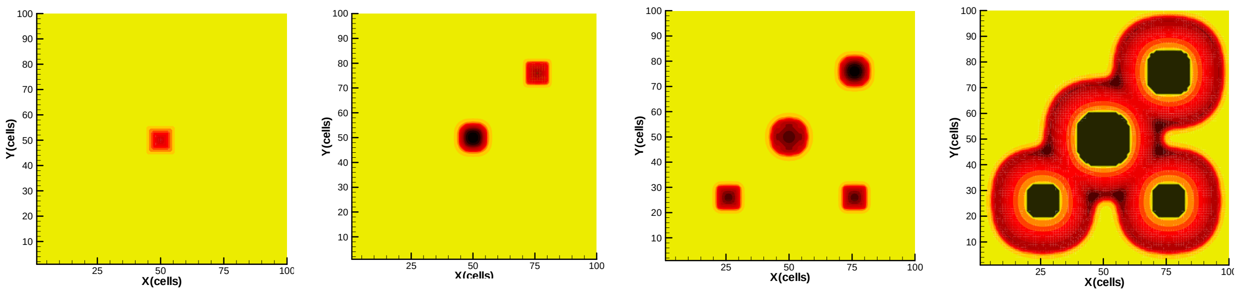


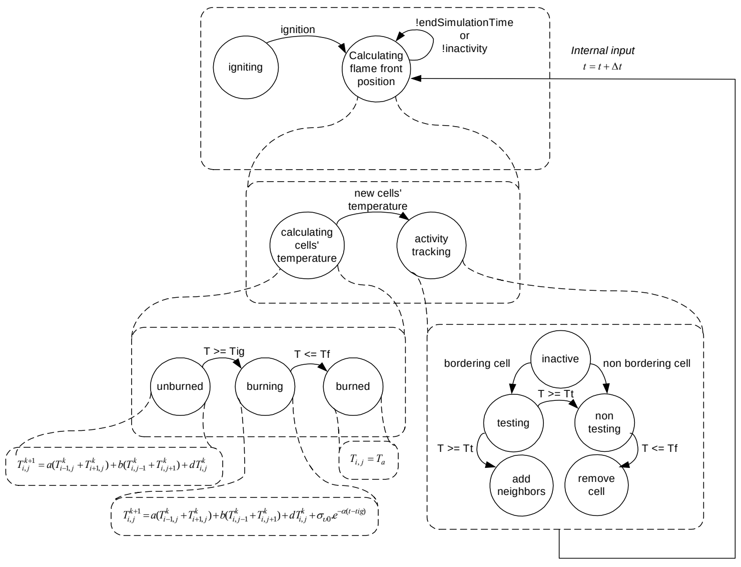
Figure 7: Fire ignitions and propagation

During fire spreading, flying brands ignite new part of lands away from fire. This is an important cause of fire spreading. However, tracking activity of flying brands is difficult. Firstly, because flying brands occur whenever during the simulation. Secondly, because they are applied away from the calculation domain, thus needing to dynamically create new calculation allocations.

Figure 7 represents an explicit case of multi-ignitions during the simulation. Simulation starts with one ignition on the center of the propagation domain. Then, at time *t=7s*, a second ignition occurs on the top right corner of the propagation domain. Finally, at *t=12s*, two new ignitions occur on the right and left bottom of the propagation domain. Last picture shows the multiple fire fronts positions at *t=70s*.

Figure 8 describes the DSCA state transitions in a fire spread simulation. First the simulation starts with the first ignition, which is simulated by an output external event of the Generator of Figure 1. Then the main simulation loop calculating the fire front position is activated. The latter consists in calculating the cells' temperature. After, according to the calculated temperatures, the calculation domain is updated. For each cell of the calculation domain, the temperature is calculated using equation *(1)*, according to the state of the cells. The calculation domain is updated using the algorithm of described in Figures 3 and 4. Phase transitions depend on the cells' temperature.

At the initialization, only one calculation domain corresponding to the one described in Figure 4 is generated. Bordering cells of the calculation domain are in a '*testing*' phase and non-bordering cells in a '*nonTesting*' one. Remaining cells are '*quiescent'*. If the temperature of a '*testing*' cell fulfills a certain threshold temperature T_t, the testing cell will pass in the '*nonTesting*' phase and neighboring '*testing*' cells will be added to the calculation domain. In the fire spread case, this threshold can be fixed slightly over the ambient temperature.



ignition
igniting
Calculating flame front position
!endSimulationTime or !inactivity
Internal input
t = t + Δt
new cells' temperature
calculating cells' temperature
activity tracking
T >= Tig
T <= Tf
unburned
burning
burned
bordering cell
inactive
non bordering cell
T >= Tt
testing
non testing
T >= Tt
T <= Tf
add neighbors
remove cell
$T_{i,j}^{k+1} = a(T_{i-1,j}^k + T_{i+1,j}^k) + b(T_{i,j-1}^k + T_{i,j+1}^k) + dT_{i,j}^k$
$T_{i,j} = T_a$
$T_{i,j}^{k+1} = a(T_{i-1,j}^k + T_{i+1,j}^k) + b(T_{i,j-1}^k + T_{i,j+1}^k) + dT_{i,j}^k + \sigma_{\upsilon 0}.e^{-\alpha(t-tig)}$

Figure 8: Transition state schema

During the simulation, the Generator simulates the flying brands by sending external events. When the DSCA receives the events and updates the calculation domain.
The resulting activity tracking is showed in Figure 9. We can notice that active cells corresponds to the fire front lines, not to the burned and non-heated areas.

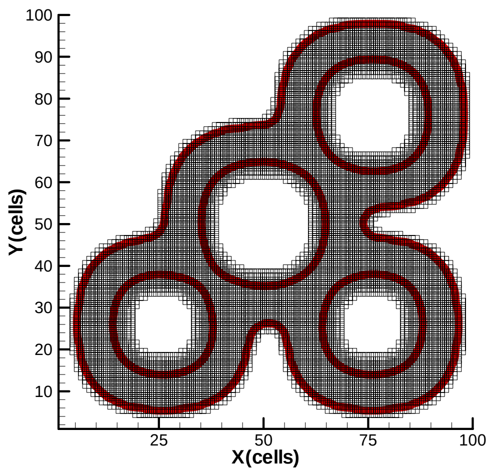


Figure 9: Activity tracking

5 CONCLUSION

Considering the previous discrete event DSCA (Barros and Mendes, 1997), new well-designed and complementary ones have been defined. These two methodologies allow to faithfully guide modelers for modeling and simulating discrete-event and discrete-time cellular simulation models.

DSCA allow to simulate a large range of complicated cellular models. Complex phenomena can be simulated thanks to basic CA simplicity. We hope that more complex phenomena will be able to be simulated thanks to DSCA. To be well understood and widely applied, DSCA definition has to be clear enough. Clearness and simplification of DSCA specification will remain our objective.

Another objective will be to improve DSCA specification using new experimentations. To achieve this goal complexity of fire spread is an infinite challenge for DSCA simulation. We plan now to extend DSCA specification to the simulation of implicit models.

Another important validation of our approach concerns network structure changes. Here again, a fire spread model taking into account wind effects (Morandini et al., 2000) will allow us to validate DSCA network structure changes. This model actually needs to dynamically change the neighborhood of burning cells according to the fire front shape.

ACKMOWLEDGMENTS

We would like to thank our research assistant Mathieu Joubert, for his technical support and for his C++ implementation of the 3D visualization tool using OpenGL.

REFERENCES

Balbi, J. H., Santoni, P. A. and Dupuy, J. L. (1998) *Int. J. Wildland Fire*, 275-284.

Barros, F. J. (1997) *ACM Transactions on Modelling and Computer Simulation,* **7,** 501-515.

Barros, F. J. and Mendes, M. T. (1997) *Simulation Practice and Theory,* **5,** 185-197.

Hill, D. R. C. (1996) *Object-oriented analysis and simulation*.

Jen, E. (1990) *Physica,* **45,** 3-18.

Mesarovic, M. D. and Takahara, Y. (1975) *General Systems Theory: A mathematical foundation*, New York.

Morandini, F., Santoni, P. A. and Balbi, J. H. (2000) In *Third International Seminar on FIRE AND EXPLOSION HAZARDS*.

Muzy, A., Innocenti, E., Barros, F., Aïello, A. and Santucci, J. F. (2003) In *SCSC*Montréal.

Muzy, A., Wainer, G., Innocenti, E., Aiello, A. and Santucci, J. F. (2002) In *AIS 2002 - Simulation and planning in high autonomy systems conference*Lisbon, Portugal, pp. 219-224.

Nutaro, J., Zeigler, B. P., Jammalamadaka, R. and Akrekar, S. (2003) In *Intrnational conference for computational science*Melbourne, Australia.

Stroustrup, B. (2000) *The C++ Programming Language*.

Wainer, G. and Giambiasi, N. (2001) *Simulation,* **76,** 22-39.

Wolfram, S. (1994) *Cellular automata and complexity: Collected papers*.

Worsch, T. (1999) *Future Generation Computer Systems,* **16,** 157-170.

Zeigler, B. P., Praehofer, H. and Kim, T. G. (2000) *Theory of modelling and simulation*.