Specification of Discrete Event Models for Fire Spreading

Alexandre Muzy
Eric Innocenti
Antoine Aiello
Jean-François Santucci
Fire Computer Modeling & Simulation
University of Corsica
SPE–UMR CNRS 6134
B.P. 52
Campus Grossetti
20250 Corti. France
a.muzy@univ-corse.fr

Gabriel Wainer
Department of Systems and Computer Engineering
Carleton University
4456 Mackenzie Building
1125 Colonel By Drive
Ottawa, Ontario K1S 5B6, Canada

The fire-spreading phenomenon is highly complex, and existing mathematical models of fire are so complex themselves that any possibility of analytical solution is precluded. Instead, there has been some success when studying fire spread by means of simulation. However, precise and reliable mathematical models are still under development. They require extensive computing resources, being adequate to run in batch mode but making it difficult to meet real-time deadlines. As fire scientists need to learn about the problem domain through experimentation, simulation software needs to be easily modified. The authors used different discrete event modeling techniques to deal with these problems. They have qualitatively compared the Discrete Event System Specification (DEVS) and Cell-DEVS simulation results against controlled laboratory experiments, which allowed them to validate both simulation models of fire spread. They were able to show how these techniques can improve the definition of fire models.

Keywords: Discrete event simulation, DEVS, Cell-DEVS, Cellular Automata, fire spread

1. Introduction

Currently, concerns for the environment have caused an increasing interest in monitoring and predicting ecosystem changes. One of the areas where these activities are crucial is the study of fire spreading. The devastating fires that have occurred over the past few years (Australia, United States, Greece, Corsica, etc.) have stressed the necessity for firefighters to have tools providing rapid and relatively accurate information concerning fire position.

SIMULATION, Vol. 81, Issue 2, February 2005 xxx-xxx

DOI: 10.1177/0037549705052230

Describing fire behavior is extremely complex, and the volume of data that the models have to grasp is particularly large. In fire spreading, the different interacting phenomena that occur (radiation, convection, diffusion, etc.) can be modeled in different ways. Although mathematical models can help us understand the influence of these phenomena on fire propagation, existing models are extremely complex, precluding any possibility of analytical solution. Instead, simulation of fire spread has been shown to be an adequate technique to analyze fire properties. Fire spread simulators numerically exercise the models for the inputs concerned and see how they affect the output measures. Comparing simulation models outputs with experimental data, the model definition can be gradually modified to

improve its definition. Simulation can help us better understand fire behavior, improving existing models.

Real-time simulators can help to predict fire spread on the fly. Existing real-time simulators use simple semi-empirical models [1], which integrate wind and slope empirically. There are more precise models (still under development) using more robust approaches to integrate wind and slope effects. These models take into account the finest mechanisms involved in fire spreading (chemical species, combustion phenomena, and hydrodynamics). They require more computing resources, thus making difficult to meet real-time deadlines [2], resulting in data and computation overloads.

Our long-term goal is to solve these problems and to be able to model fire spread with precision within real-time constraints. We intend to provide a user-friendly, high-performance environment to assist physicists in modeling fire spread and to use it to give real-time advice for firefighters.

We have used an approach based on *reductionism* [3] and *program evolvability* [4]. *Program evolvability* provides the ability to modify a program easily as we learn about the problem domain. To increase productivity, the simulation program corresponding to the fire spread model must be able to take the model modifications into account easily; a fire spread model may need to be dynamically modified according to numerous conditions: climatic (wind and ambient temperature), geographic (slope, non-homogeneous vegetation, and natural barrier), and external (firefighters) conditions.

Reductionism allows one to decrease the complexity of the phenomenon, reducing the behavior of the system to the behavior of its parts. Diffusion processes (oil spills, fire spread, insect infestation, etc.) are usually represented as partial differential equations (PDEs) discretized in the form of finite differences or finite elements. In our case, fire spread was represented using a reaction-diffusion equation [5], and using a reductionistic view, we uniformly meshed the propagation domain, conforming with a cellular model in which each cell is composed of earth and plant matter. This simplifies the problem definition, enabling the efficient execution of simulations in real time. Nevertheless, it is still necessary to investigate more accurate descriptions, using simulation to understand new phenomena.

Our research started by doing a set of experiments, which were used to propose a model of the behavior and later validate the results. Experimental fires were conducted on *Pinus Pinaster* litter, in a closed room without any air motion, at the INRA (Institut National de la Recherche Agronomique) laboratory near Avignon, France [5]. These experiments were performed to observe fire spread for point-ignition fires under no slope and no wind conditions. The experimental apparatus was composed of a 1–square meter aluminum plate protected by sand. A porous fuel bed was used, made up of pure oven-dried pine needles spread as evenly as possible on the total area of the combustion table (to obtain a homogeneous structure). The experiment consisted of igniting a point using alcohol. The resulting spread of the flame across the needles was closely observed with a camera and thermocouples.

Our first model used a cellular automaton (CA) [6] containing continuous state variables (the cells' temperatures) [7]. CAs are idealizations of physical phenomena with space, state, and time discretized. CA components (called *cells*) are geometrically located on an infinite grid and connected in a uniform way. Cells are identical and contain a computational apparatus that is influenced by the cell's *neighborhood* (i.e., those cells geometrically close to the origin cell). From a modeling point of view, physical arguments in disciplines ranging from physics to biology and artificial life support the adjacent neighborhood assumptions [6]. CA may thus be considered as discrete idealizations of PDEs.

In our model, the temperature of each cell was represented by a PDE. Starting with a PDE with independent derivatives in time and space, we discretized the model to obtain a continuous simulation model in cellular form with equal derivative functions. The code for this CA proved to be very efficient in terms of performance, but it did not support program evolvability or detailed system behavior. Furthermore, the simulation cost of CA could be highly reduced if the calculation area (the set of cells whose state is computed at a given time step) amounts to the flame front. None of these needs is supported by CA techniques. Instead, the Cell–Discrete Event System Specification (Cell-DEVS) formalism [9] is especially tailored to solve problems such as this one and can be helpful in achieving our goals. This technique is based on the DEVS formalism [10], which enables modular and hierarchical modeling associated with a discrete event technique. Every cell of a Cell-DEVS is considered an atomic DEVS model with explicit timing delays, and a procedure to create composite models is depicted. To attack the complexity of a system, a simulation model should be able to record the system's timing information and lifetimes for state values of the system. Cell-DEVS provides a sound methodology for describing complex timing behavior without knowing the simulation mechanism of the delays. This enables the definition of complex cellular models, reducing development time related to the programming of timing control.

We will present the benefits of applying Cell-DEVS for complex cellular systems, providing algorithms of both DEVS and Cell-DEVS models of fire spread. For that purpose, we consider program evolvability, algorithmic complexity for the modeler, and computation efficiency of the simulation. First, a brief description of DEVS and Cell-DEVS formalisms is introduced. After, we present some background material on the simulation of mathematical models of fire spread and the model we use. Then, DEVS and Cell-DEVS models of fire spread are described. Finally, we show how comparisons between simulation results and controlled laboratory experiments allowed us to

validate the simulation models, and we discuss advantages and problems of both DEVS and Cell-DEVS models of fire spread.

2. Background

DEVS [10] is a system specification formalism based on concepts of modularity and hierarchical composition. Using a hierarchical approach, DEVS uses different models and submodels coupled in a modular way to specify systems behavior and structure. Modular specifications view a model as possessing input and output ports through which all interactions with the environment are mediated. In the discrete event case, events determine values appearing on such ports. External events are received on input ports and sent on output ports. This allows one to easily combine different simulation models and to connect them with experimental data.

DEVS atomic models give a local description of the dynamic behavior of the studied problem, while the coupled models represent the different interconnections between a set of model elements (other atomic or coupled models).

A DEVS atomic model is described as

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

where X is the input events set, S is the state set, Y is the output events set, δ_{int} is the internal transition function, δ_{ext} external transitions are the external transition function, λ is the output transition function, and D is the time advance function.

A DEVS coupled model is defined as

$$CM =< X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, select >,$$

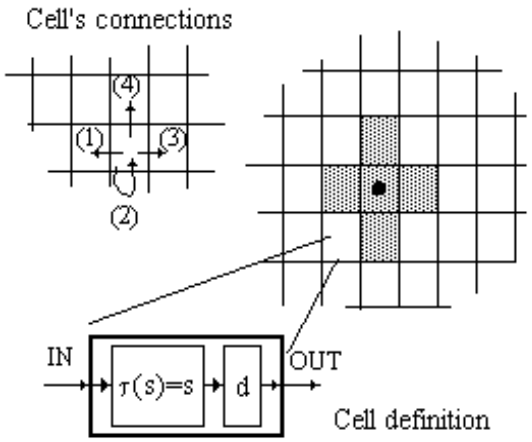
where X is a set of input events, Y is a set of output events, D is an index of the basic DEVS model, $\{M_i\}$ is the set of classic DEVS models, $\{I_i\}$ is the set of influences of model i, $Z_{i,j}$ is the i to j translation function, and *s*elect is the tie-breaking selector.

The Cell-DEVS formalism was built to solve cell-based systems. Using DEVS properties, cellular models can be easily built, improving their execution speed and precision for continuous systems. Their timing definition is simplified using delay functions. Each cell in a Cell-DEVS model is defined as an atomic DEVS model, and they are integrated into a coupled model. Timing delays allow a modeler to define different timing behavior.

The atomic models (the cells) can be described as

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

where X defines the external inputs, Y is the external outputs defining the model's interface, θ is the cell state definition, N is the neighborhood set represented by a list defining the relative position between the neighbors and the origin cell, *d*elay defines the kind of delay for the cell (transportation or inertial) and d its duration, δ_{int} manages internal transitions, δ_{ext} manages external transitions, τ is used for local computations (which uses the state values of the neighborhood to compute the future value of a cell), λ is used for outputs, and D is used for the state's duration. The delay function allows one to differ the transmission of the results. A *transport* delay can be associated with each cell, allowing one to defer the transmission of the cell's changes. To provide transport delays, the external transition function schedules an internal event after the time defined by the delay. When the delay has expired, the state change is transmitted to the influencees (the output function sends the first scheduled output showing a state change, and the internal transition function schedules a new internal event). If there is no other scheduled output, the cell passivates. When *inertial* delays are used, a scheduled output can be preempted if a new input arrives before the scheduled time. This only happens if the state change produced by the new input is different from the one previously scheduled for output. An input will be discarded if it is not steady during the inertial delay of the cell.



Neighborhood list: { (0,-1), (0,0), (0,1), (-1,0) }

Note: -1: left, up and 1: right, down

Figure 1. Informal definition of a Cell-DEVS model and its neighborhood

The atomic cell models can be coupled with others, forming a multicomponent model. These are defined as a space consisting of atomic cells connected by the neighborhood relationship. After, they can be integrated with other Cell-DEVS or DEVS models. An informal description of coupled models is included in Figure 1.

A Cell-DEVS coupled model is defined by

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

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

$$C_{ij} =< X_{ij}, Y_{ij}, S_{ij}, N_{ij}, d_{ij}, \delta_{intij}, \delta_{extij}, \tau_{ij}, \lambda_{ij}, D_{ij} > .$$

The B set defines the cell's space border. If the set is empty, every cell in the space has the same behavior. The space is *wrapped*, meaning that cells in one border are connected with those in the opposite. Otherwise, the border cells will have different behaviors than those of the rest of the model. The interface I is built using X_{list} and Y_{list}, two lists defining the model's outputs/inputs. Finally, the Z function allows one to define the coupling of cells in the model. *Select* is the tie-breaking function.

3. Simulation of Mathematical Models of Fire Spread

In fire spread modeling, the problem consists of calculating the flame front position using the temperature distribution in complex fuel. To achieve real-time simulation, complexity of fire spread and data volume require having simple mathematical models capable of predicting the main behavioral features of fire.

Weber's **[McArthur IN REF.]** [11] classification identified three types of mathematical models for fire propagation according to the methods used in their construction. The first are the *statistical* models [12], which do not attempt to involve physical mechanisms, being merely a statistical description of test fires. The results can be very successful in predicting the outcome of similar fires to the test fires. However, the lack of a physical basis means that the statistical models must be used cautiously outside the test conditions.

The second category of models incorporates *semi-empirical* models [13] based on the principle of energy conservation but does not distinguish between the different mechanisms of heat transfer. At present, most real-time simulators [14-17] are based on Rothermel's stationary model [13]. This is a one-dimensional model empirically integrating wind and slope, in which a second dimension can be obtained using propagation algorithms [18]. Currently, a great deal of effort has been placed in improving the simulation of Rothermel's model. In the CA field, recent studies pinpoint the need for developing new classes of CA for fire-spreading applications [19]. Other applications have proposed to improve CA capabilities for fire spread simulation using the DEVS formalism. Unlike CA, DEVS models can receive external updated information, and the fire perimeter can be updated at any moment due to the continuous time nature of the discrete event specifications [20]. Cell-DEVS formalism allowed for the simplification of Rothermel's model implementation [21], and the Dynamic Structure Cellular Automata (DSCA) [22] allowed for dynamically creating active cells and removing the quiescent ones, saving memory for large cell spaces.

A third category incorporates *physical* models [23], which integrate wind and slope effects in a more robust manner describing the various mechanisms of heat transfer and production. Among the latter, the *multiphase* approach takes into account most mechanisms involved in fire spreading (chemical species, combustion phenomena, and hydrodynamics) [24, 25]. Although the simulation of such models requires a very long calculation time, the multiphase approach can be used to improve or develop simpler models dedicated to fire spread simulators [26, 27]. We developed a strategy based on the reduction of multiphase models, leading to a reduced physical model focusing on the main mechanisms involved in fire spreading [5]. The latter is nonstationary and two-dimensional.

The basic model uses elementary cells of earth and plant matter. Under no wind and no slope conditions, the temperature of each cell is represented by the following PDE:

$$\frac{\partial T}{\partial t} = -k(T - T_a) + K\Delta T - Q\frac{\partial \sigma_v}{\partial t}, \quad \text{(1a)}$$

in the domain

$$\sigma_v = \sigma_{v0} \text{ if } T < T_{ig}, \quad \text{(1b)}$$

$$\sigma_v = \sigma_{v0.} e^{-\alpha(t - t_{ig})} \text{ if } T \geq T_{ig}, \quad \text{(1c)}$$

$$T(x, y, t) = T_a \text{ at the boundary}, \quad \text{(1d)}$$

$$T(x, y, t) \geq T_{ig} \text{ for the burning cells}, \quad \text{(1e)}$$

$$T(x, y, 0) = T_a \text{ for the nonburning cells at } t = 0. \quad \text{(1f)}$$

Here, T_a (27°C) is the ambient temperature, T_{ig} (300°C) is the ignition temperature, t_{ig} (s) is the ignition time, T (°C) is the temperature, K ($m^2\ s^{-1}$) is the thermal diffusivity, Q (m^2 °C/kg) is the reduced combustion enthalpy, Δ is the Laplacian in two-dimensional Cartesian coordinates, α (s^{-1}) is the combustion time constant, σ_v (kg m^{-2}) is the vegetable surface mass, and σ_{v0} (kg m^{-2}) is the initial vegetable surface mass (before the cell combustion). The model parameters are identified from experimental data of temperature versus time. In a previous study, the finite element method (FEM) and the finite difference method (FDM) were used to discretize the model [28]. FDM was chosen because it provided equivalent results, while FEM appeared more complex to implement and involved longer execution time.

The study domain was meshed uniformly with cells of 1 cm^2 and a time step of 0.01 sec. The physical model was solved by the FDM, leading to the following algebraic equation:

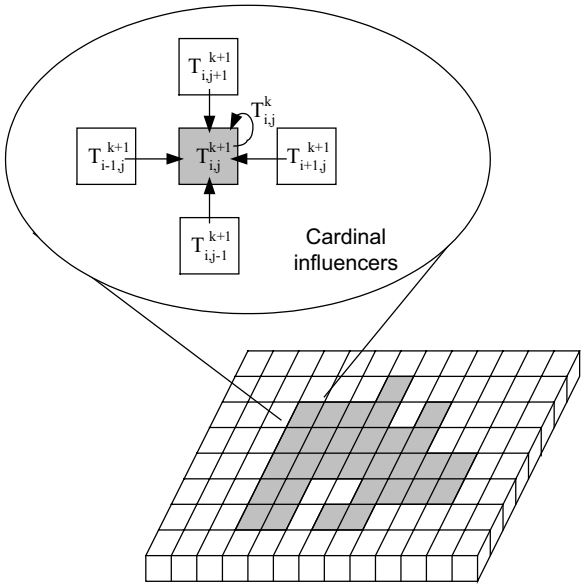


Figure 2. Cellular model of fire spread

$$T_{i,j}^{k+1} = a(T_{i-1,j}^{k} + T_{i+1,j}^{k}) + b(T_{i,j-1}^{k} + T_{i,j+1}^{k}) + cQ\left(\frac{\partial \sigma_v}{\partial t}\right)_{i,j}^{k+1} + dT_{i,j}^{k}. \quad (2)$$

Here, T_{ij} is the grid node temperature. The coefficients a, b, c, and d depend on the time step and mesh size considered. As illustrated in Figure 2, the propagation domain thus consists of a cellular model in which each future cell's temperature is calculated using the current cell's temperature and temperature in the cardinal neighbors.

4. DEVS Modeling of Fire Spread

The numerical solution of the physical problem presented in equation (2) needs the meshing of the spread domain. First, we split up the problem in a mesh of cells and implemented each cell as a DEVS model [29], as illustrated in Figure 3. In this case, each atomic model (*C* element) corresponded to each cell of the propagation domain. The whole components were interconnected by means of a coupled model *M*.

A single atomic model, the *generator* element, was linked to each *C* element, initiating fire spread by specifying ignition zones. The generator's input port, *in.G*, was used to determine ignition location and type of ignition (punctual or linear). A punctual ignition consists of igniting a point of the domain, thus resulting in a circular fire spread. A linear ignition consists of igniting different points lined up, thus resulting in a linear propagation of a fire front. The generator's output ports, *out0.G* to *out8.G*, were linked to each *C* element to send ignition temperatures. With this separation between the model and the experimental frame, we could study the same model under different conditions just by changing the generator model.

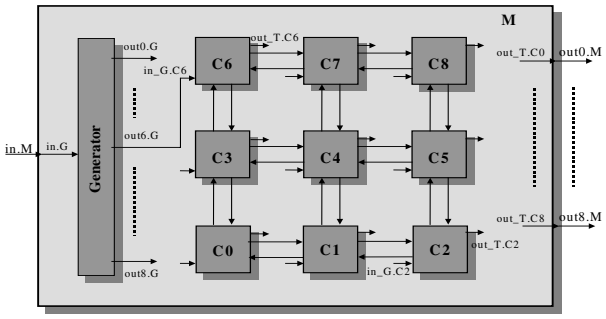


Figure 3. DEVS model of fire spread

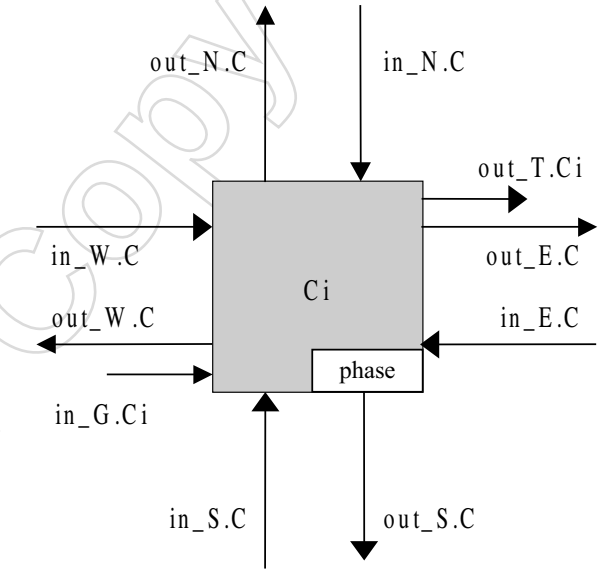


Figure 4. *C* element description

As detailed in Figure 4, the *C* elements use different ports to interact with the cell's neighbors and external models:

- an input port *in_G.Ci* (where *i* corresponds to the number of the cell) for ignition,
- an output port *out_T.Ci* to obtain the temperature value of the element,
- four input ports (*in_N.C*, *in_S.C*, *in_E.C*, and *in_W.C*) and four output ports (*out_N.C*, *out _S.C*, *out _E.C*, and *out*

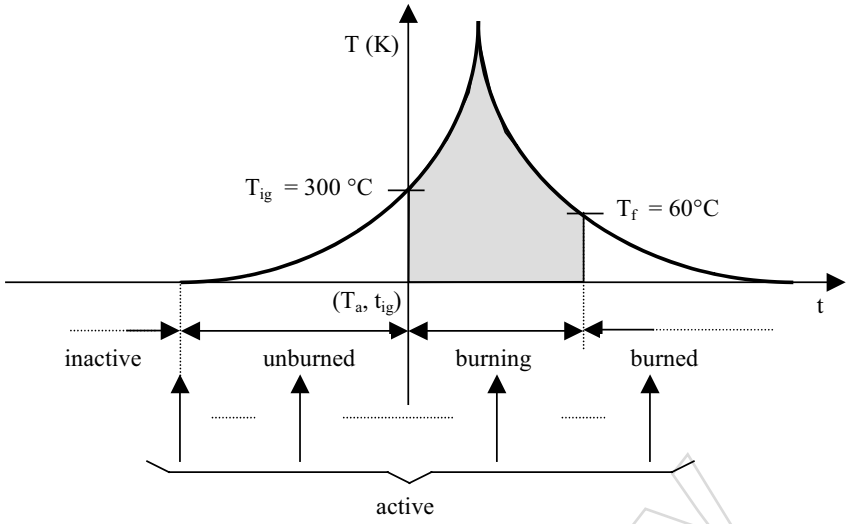


Figure 5. Simplified temperature curve of a cell in the propagation domain

_*W.C*) corresponding to the cardinal neighborhood. These ports are used for information exchanges between the cell and its neighbors.

Figure 5 depicts a simplified temperature curve of a cell in the propagation domain. Transitions from a state to a next state with a different applicable model signify a qualitative change in the dynamic behavior [10]. Such qualitative states are called *phases*. Different phases corresponding to time advances and the cell's behavior can be defined from Figure 5. Hence, the *C* elements represented phases *inactive*, *active*, *unburned*, *burning*, and *burned*, with corresponding time advance functions *ta* of *infinity*, *0*, *1*, *1*, and *infinity*.

As long as a cell does not receive a temperature different from the ambient temperature T_a, it does not propagate its temperature, and it remains in the *inactive* phase. This allows the restriction of the calculation domain around the flame front, thus reducing calculation time. The intermediate active phase allows activating the cell when receiving the temperature of a neighbor. When a cell receives temperatures different from T_a, it changes to the *unburned* phase. The physical description of section 3 assumes that above a threshold ignition temperature T_{ig}, the combustion occurs (see equation (1c)). Hence, when the cell's temperature becomes higher than T_{ig}, the cell passes from *unburned* to *burning*. Under a T_f temperature, the combustion is considered as finished, and the cell passes from *burning* to *burned*, deliberately neglecting the end of the real curve to save simulation time.

Figure 6 depicts the *C* element's algorithm corresponding to the cell's behavior. We define the input/output ports on lines 0 and 2. Line 1 sets out the state variables used. The behavior of the δ_{ext} and δ_{int} transition functions, the output function λ, and the time advance function T_a is then described from lines 3 to 95.

The external transition function (depicted from lines 3-16) stores the neighboring temperatures received (lines 7-16) or the initialization temperature of the cell (lines 4-6). This is made by testing the input ports (lines 4 and 7).

The output function (lines 17-43) returns the cell's state. If the cell is burned, the function will return the ambient temperature T_a. Otherwise, the function will check if the cell's temperature changed and will send it to the neighbors. In any other case, it returns the cell's temperature to the neighbor that has sent its temperature (if the latter is different from T_a). When the function has received the temperature values from every neighbor, it calculates the cell's value according to the current phase.

The internal transition function (lines 44-77) affects the cell's phase after the output function execution. At initialization, if the cell ignites (lines 46-53), the phase corresponding to the ignition temperature will be affected. Otherwise, a new phase is activated, the cell's temperature is updated, and the simulation time is incremented (lines 54-77).

Finally, the time advance function T_a sets the output timing delays (lines 78-95). The model was simulated using the abstract simulator principles introduced in Wainer [30]. Every atomic model is connected to a *simulator* and every coupled model to a *coordinator*. The coordinator manages the simulation process of the lower level. This organization results in the definition of a simulation tree such as the one depicted in Figure 7. The *Root* processor writes and reads data received from the simulators and activates the coordinator *Coo*. The coordinator synchronizes the message exchanges between the simulators *SG*, *S*0, *S*2, . . ., *S*8, which handle the atomic models *Generator*, *C*0, *C*1, . . ., *C*8.

```
0.  X: in_N.C, in_S.C, in_E.C, in_W.C, in_G.C
1.  S: phase, old_phase, T(t-1),T(t), T(t+1), t_ig, T_ig, T_f, T(t)_N, T(t)_E, T(t)_S, T(t)_W
2.  Y: out_N.C, out_S.C, out_E.C, out_W.C, out_T.C

3.  δext(C):
4.  If (port==in_G.Ci)
5.     T(t)←in_G.Ci.value
6.     send a d-message containing T(t) at t
7.  Else
8.     old_phase←phase
9.     phase←active
10.    If (old_phase!=burned)
11.       store neighboring cell temperature
12.       send a d-message at t containing the port received
13.    Else
14.       send a d-message at t containing the port received
15.    Endif
16. Endif

17. λ(C):
18. If (*-message empty)
19.    If (phase==burned)
20.       send a y-message containing T_a to the root
21.    Else
22.       If (T(t)!=T(t-1))
23.          send a y-message containing T(t) to the neighboring cells
24.       Endif
25. Else
26.    phase←old_phase
27.    Switch(phase)
28.       Case 'inactive':
29.          send a y-message, at t=0,containing T(0) to the neighboring cells and to the root
30.       EndCase
31.       Case 'unburned','burning':
32.          If (the cell received a neighboring cell temperature different from T_a)
33.             send a y-message containing T(t) to the neighboring cells
34.             If (all neighboring cells temperature are known)
35.                calculate the cell's temperature
36.                send a y-message, containing T(t+1) to the root
37.             Endif
38.          Endif
39.       EndCase
40.       Case 'burned':
41.          send a y-message containing T_a to the neighboring cell, which has sent its temperature
42.       EndCase
43.    End Switch
```

Figure 6. Algorithm of the *C* components [29] *(continued on next page)*

The simulation evolves through message passing. When a simulator receives an *x*-message, it executes the external transition function, which returns a *d*-message. This message is transformed into a *-message, which leads to the execution of the output function, which returns a *y*-message. The internal transition function is then executed, returning a *d*-message.

The simulation evolution can be analyzed by considering both the *C* component's algorithm of Figure 6 and message exchanges between processors schematized in Figures 9 and 10. Processors allow the sharing of messages between the models (generator and cells), thus activating their behavioral functions. The function activation process can be followed by considering both Figures 9 and 10, while the functions of the algorithm are detailed in Figure 6.

Figure 8 illustrates the point ignition applied at the initialization ($t = 0$) on the plate center, thanks to a temperature gradient. This gradient allows one not to create a thermal shock for the mathematical model. As described in Figure 9, this is simulated by an incoming *x*-message on the generator (2), with the coordinates and the type of ignition (linear or punctual). Then, the generator sends immediately *n* *y*-messages (one message per ignited cell) containing the ignition temperatures to the coordinator (5). The coordinator then transforms the *y*-messages into *x*-messages and addresses them to the simulators *Ci* involved in the ignition (6).

The *x*-messages carry out the external transition functions δ_{ext} of the *Ci* elements associated with the simulators. Each δ_{ext} function stores the cell's temperature and sends

```
44. δint(C):
45. If (the temperature of the cell has not been calculated)   // initialization phase
46.    If (phase==inactive and T(t)!=Ta)
47.       If (T(t)<Tig)
48.          phase←unburned
49.       Else
50.          phase←burning
51.       Endif
52.    Endif
53. Else
54.    Switch(phase):
55.        Case 'unburned':
56.           If (T(t+1)≥Tig)
57.              tig←t
58.              phase←burning
59.           Endif
60.           T(t-1)←T(t)
61.           T(t)←T(t+1)
62.           T(t+1)←0
63.           send an empty d-message at t+1
64.        EndCase
65.        Case 'burning':
66.           If (T(t+1)≤Tf)
67.              phase←burned
68.              T(t)←Ta
69.              send an empty d-message at t
70.           Else
71.              T(t-1)←T(t)
72.              T(t)←T(t+1)
73.              T(t+1)←0
74.              send an empty d-message at t+1
75.           Endif
76.        EndCase
77. End Switch

78. Ta(C):
79. Switch(phase) :
80.    Case 'inactive':
81.       ta←∞
82.    EndCase
83.    Case 'active':
84.       ta←0
85.    EndCase
86.    Case 'unburned':
87.       ta←1
88.    EndCase
89.    Case 'burning':
90.       ta←1
91.    EndCase
92.    Case 'burned':
93.       ta←∞
94.    EndCase
95. End Switch
```

Figure 6. *(continued from previous page)*

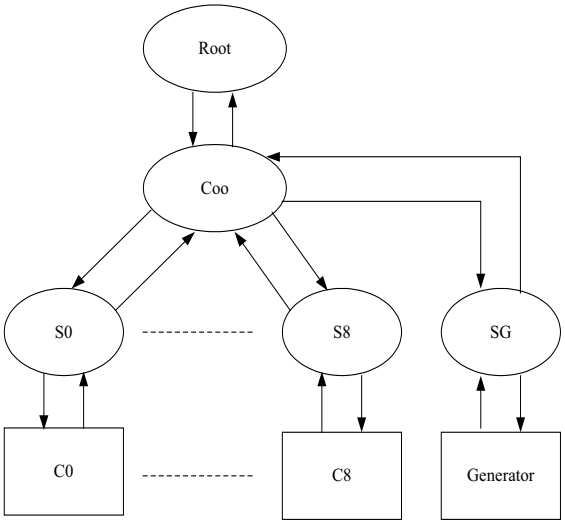


Figure 7. DEVS simulator associated with the fire spread model

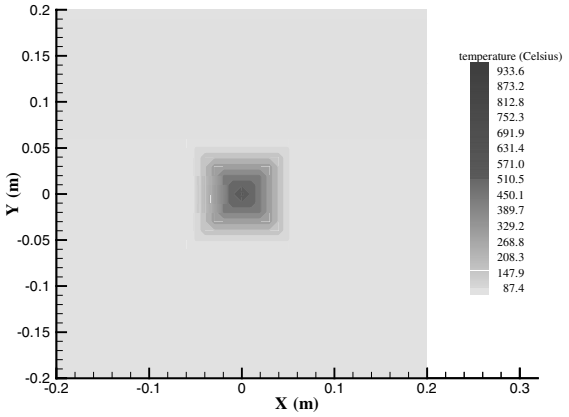


Figure 8. Initial temperature gradient for a punctual ignition

a d-message to the coordinator (7), which transforms it in a *-message. The *-messages finally induce the execution of the output functions λ (8).

The λ functions send four y-messages (9) containing the cell's temperature to the four neighbors. Each neighbor will send, by turn, four y-messages to its own neighbors until they propagate a temperature equal to T_a. In that case, the requested cells will not respond. The internal transition function δ_{int} finally assigns the phase corresponding to the ignition temperature of the cells.

The algorithm to pass at time $t + 1$ is depicted in Figure 10. When a cell receives the fourth temperature (1), λ calculates the cell's temperature and sends a y-message to the root (4). The δ_{int} function is then executed. This one assigns the phase corresponding to the cell's temperature and sends a d-message to pass at time $t + 1$ (6). Above $t = 1$, a cell will be activated if at least one of its cardinal neighbors has a temperature greater than T_a.

Finally, when a cell is burned, it passes in the burned phase and then sends its temperature (T_a) to the root. If a neighbor sends it a y-message, the burned cell returns the temperature T_a.

Although the DEVS model of fire spread allowed program evolvability, the computer implementation proved to be complex. Important model modifications can take a long time because of the configuration of the message interfaces. The debugging phase proved to be complex due to both timing delay verifications and code complexity. Finally, the whole simulation model proved difficult to understand and was modified by a noncomputer science specialist. To solve these problems, we redefined the model using the Cell-DEVS formalism.

5. Cell-DEVS Modeling of Fire Spread

The CD++ environment [31] provides a means to implement DEVS and Cell-DEVS models. To implement Cell-DEVS models, a user simply specifies the cell domain dimensions, the cell's neighborhood, and the cell's behavior by defining simple, logical rules using a built-in specification language. It has the format {*result*} *delay* {*condition*}. The semantics of the sentences is that, if the *condition* is true, the cell will take the *result* value and will send it through output ports after a *delay* time. If the condition is not valid, the next rule is evaluated (according to the order in which they were defined), repeating this process until a rule is satisfied. The most common operators are included: Boolean (*AND*, *OR*, *NOT*, *XOR*, *IMP*, and *EQV*), comparison (=, !=, <, >, <=, and >=), and arithmetic (+, –, *, and /). In addition, different types of functions are available: trigonometric, roots, power, rounding and truncation, module, logarithm, absolute value, minimum, maximum, greatest common denominator (GCD), and least common denominator (LCM). Other existing functions allow one to check if a number is an integer, even, odd, or prime. Space *zones*, defined by a cell range, can be associated with a set of rules different from the rest of the cell space.

With these considerations in mind, we redefined our fire model using Cell-DEVS [32]. As described in Figure 11, we used two planes representing different phenomena. The first one describes fire spread, and each cell computes temperature, whereas the second plane stores the ignition time t_{ig} of each cell (see equation (1c)). Every cell in this plane detects the corresponding cell of the propagation plane when it starts burning.

Figure 12 represents the model specification in CD++. Lines 0 and 1 of this specification declare the components of the top-level coupled model. Then, we include a definition for the Cell-DEVS coupled model representing the

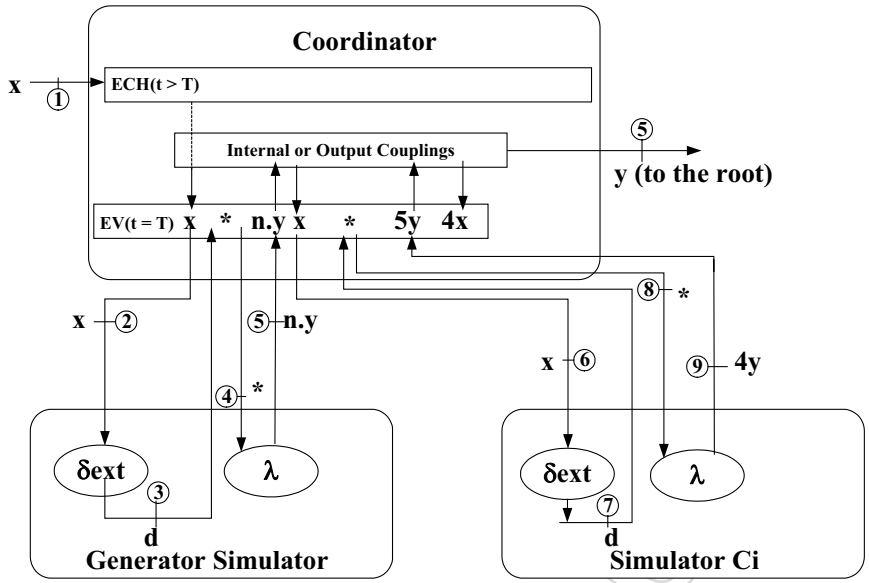


Figure 9. Message exchanges for the *C* element ignition

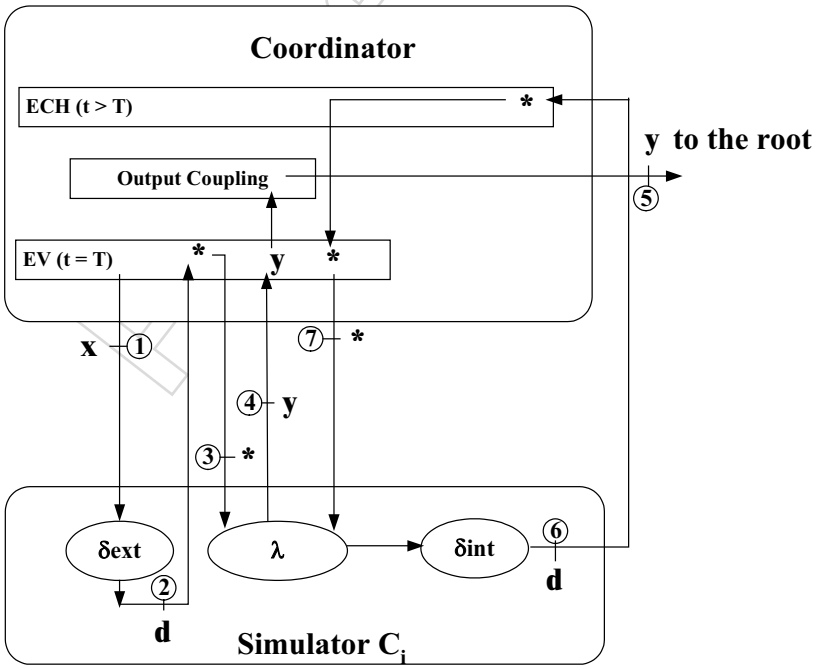


Figure 10. Message exchanges for time increment

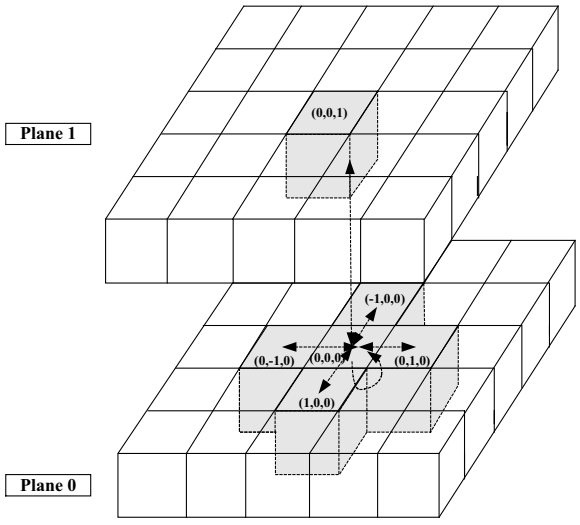


Figure 11. Cell's neighborhood specification

fire spread model by including the parameters mentioned earlier (neighborhood, dimension, type of delay, etc.).

In the propagation plane, we defined a border zone (*cst*) corresponding to the model behavior in the boundaries of the cell space (see equation (1d)). These cells stay at the ambient temperature (lines 15 and 16). A second zone (*ti*) represents the area of plane 1, which will use a different set of rules, defined on line 17. These rules show how we store the ignition times: if the corresponding cell in plane 0 starts to burn, we will record the current simulation time in the cell.

The rules used to compute the cells' temperatures start on line 20. Macros were used to make the model specification more readable. The two macros (lines 25-30) contain the rules corresponding to the temperature calculus when the cell is in the *unburned* or *burning* phase. The first rule used to compute the cell temperatures is presented on line 21. The latter corresponds to the unburned phase. If the computed cell temperature is higher than the present value, the cell will take the computed value. As cells only contain one state variable, this checking is necessary to know which part of the temperature curve we are computing. The same occurs during the transient period (simulation time smaller than 20), when the cell's state is neither burning nor burned. Based on the same principles, rules on lines 22 and 23 correspond to the burning and burned phases. The last rule on line 24 is used for cells that are away from the fire front or *burned*. These cells remain unchanged, and they are not reactivated.

Cell-DEVS simplifies the construction of the fire spread model, allowing simple and intuitive model specification. The Cell-DEVS rules facilitated the debugging phase and reduced development time. Complex model modifications can now be easily and quickly integrated with the current model of fire spread, even by a noncomputer science specialist.

6. Validation of the Simulation Model Outputs

As mentioned earlier, our research started by doing a set of experiments, which were used to study the problem. The results of these experiments were also used to validate the simulated results. We built different applications in each phase of the project. We started by implementing a CA model written in C [33]. Then, the DEVS model of section 4 was implemented on the JDEVS environment [34]. Finally, CD++ was used to implement the Cell-DEVS fire spread model. Our goal was to validate the DEVS and Cell-DEVS models with experimental data and to compare them with an already validated CA model.

Different experimental and simulated results have been compared: rate of spread of the fire front, temperature growth of a cell, and fire spread over the domain. The different qualitative comparisons allow one to validate the mathematical model and the associated simulation models.

An overview of the simulation of a point ignition is provided in Figure 13. Outputs of the CA, DEVS, and Cell-DEVS models consisted of temperature grids at given time steps. Fire front evolution has been achieved by linking isothermal points at T_{ig} = 300°C at each time step. The white squares in Figure 1 represent the front positions obtained with the CA model that were already validated against experimental data [33]. Both DEVS and Cell-DEVS simulation models' outputs are equivalent and have been illustrated by the color gradation corresponding to the cells' temperatures.

At time t = 30 sec, we observe that the circular wave fronts are the same. At t = 50 sec, a difference appears between the fire front widths of the CA and both DEVS and Cell-DEVS models. This is due to the end combustion assumption (T_f = 60°C) described in section 4. Hence, the temperature of the flame front obtained with the DEVS and Cell-DEVS models decreases more quickly, reducing the active zone (corresponding to the cells calculating their temperatures).

Over the whole propagation domain, the active zones of the DEVS and Cell-DEVS simulation models correspond to the red gradations. While a CA will calculate the temperature of every cell of the domain (even the inactive ones in yellow), the DEVS and Cell-DEVS models allow one to fit the active zone by specifying significant events in front of the fire front and threshold conditions at the end of the fire front. For the DEVS model, the significant event tests can be observed on line 32 of the algorithm presented in Figure 6, and the threshold condition can be observed on line 66. CD++ directly implements the principle of significant event detection for the simulation of Cell-DEVS models. The threshold condition can be observed on lines 22 and 23 of Figure 12.

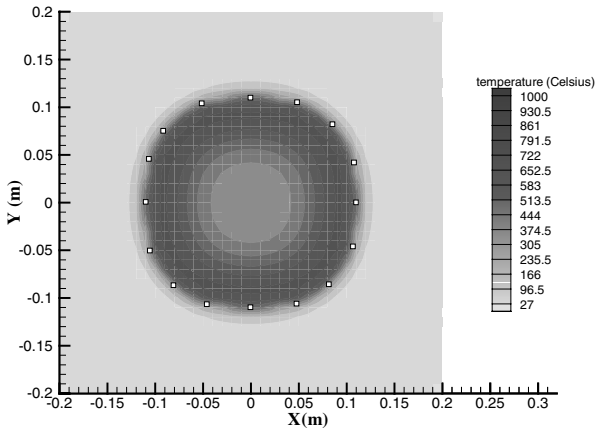
```
0.  [top]
1.  components : ForestFire
2.  [ForestFire]
3.  type : cell
4.  dim : (100,100,2)
5.  delay : transport
6.  border : nowrapped
7.  neighbors : ForestFire(-1,0,0) ForestFire(0,-1,0) ForestFire(1,0,0)
8.  neighbors : ForestFire(0,1,0)  ForestFire(0,0,0)  ForestFire(0,0,-1) ForestFire(0,0,1)
9.  zone : cst { (0,0,0)..(0,99,0) }
10. zone : cst { (1,99,0)..(99,99,0) }
11. zone : cst { (99,0,0)..(99,98,0) }
12. zone : cst { (1,0,0)..(98,0,0) }
13. zone : ti { (0,0,1)..(99,99,1) }
14. localTransition : FireBehavior

15. [cst]

    %Constant border cells
16. rule : { (0,0,0) } 1 { t }
17. [ti]
18. rule : { time * 0.01 } 1 { (0,0,0) = 1.0 AND (0,0,-1) >= 300  }
19. rule : { (0,0,0) } 1 { t }

20. [FireBehavior]
    %Unburned
21. rule : { #macro(unburned) } 1 { (0,0,0) < 300 AND (0,0,0) != 26 AND ( #macro(unburned) > (0,0,0)
    OR time <= 20 )  }
    %Burning
22. rule : { #macro(burning) } 1 { (0,0,0) != 26 AND ( ( (0,0,0) > #macro(burning) AND (0,0,0) > 60)
    OR (#macro(burning) > (0,0,0) AND (0,0,0) >= 300) ) }
    %Burned
23. rule : { 26 } 1 { (0,0,0) <= 60 AND (0,0,0) != 26 AND (0,0,0) > #macro(burning) }
    %Stay Burned or constant
24. rule : { (0,0,0) } 1 { t }

%Macros definition
25. #BeginMacro(unburned)
26. ( 0.98689 * (0,0,0) + 0.0031 * ( (0,-1,0) + (0,1,0) + (1,0,0) + (-1,0,0) ) + 0.213 )
27. #EndMacro

28. #BeginMacro(burning)
29. ( 0.98689 * (0,0,0) + 0.0031 * ( (0,-1,0) + (0,1,0) + (1,0,0) + (-1,0,0) ) + 2.74 * exp(-0.19 * (
    (time + 1) * 0.01 - (0,0,1) ) ) + 0.213 )
30. #EndMacro
```

Figure 12. Fire spread model specification: Fire.ma

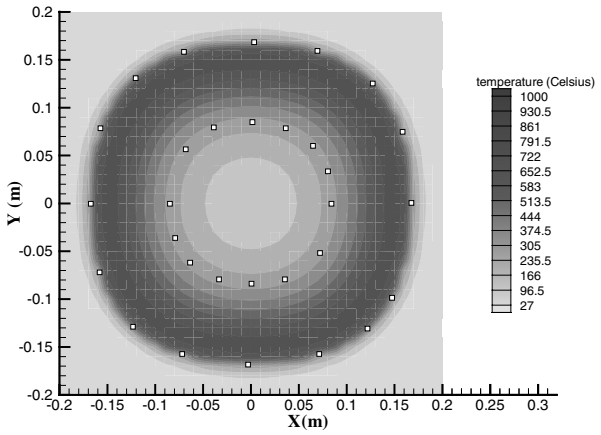
The effect of the threshold assumption can also be observed in Figure 14, which shows the simulated and observed temperature curves of a cell of the domain. The end of the simulated curve is cut down, but this no longer has any influence on fire propagation [7]. The preheating of the unburned zone depends on the fire front. The burned area has no influence on the burning one. The end of the fire front can thus be neglected to improve simulation performances.

Rothermel's stationary model considers a fire rate of spread (ROS) as constant even in the ignition phase, which is not a correct representation of actual propagation. Figure 15 depicts the ROS of the fire front obtained with our mathematical model of fire spread. It can be seen that our model considers the acceleration phase, proving a better accuracy of our model in the ignition phase.

Thanks to experimental data, we ensure that the DEVS and Cell-DEVS models are capable of reproducing, with sufficient accuracy, laboratory fire propagations under no wind and no slope conditions. Modularity, hierarchical modeling, and abstract simulator principles of DEVS provide program evolvability. Discrete events allow one to activate only the active cells of the propagation domain (the cells whose temperature is different from T_a), and as shown in Zeigler, Praehofer, and Kim [9], this results in improved execution speeds when compared with CA.



(a)



(b)

Figure 13. Comparison of the circular front positions of the CA, DEVS, and Cell-DEVS simulations at time (a) $t = 30$ sec and (b) $t = 50$ sec

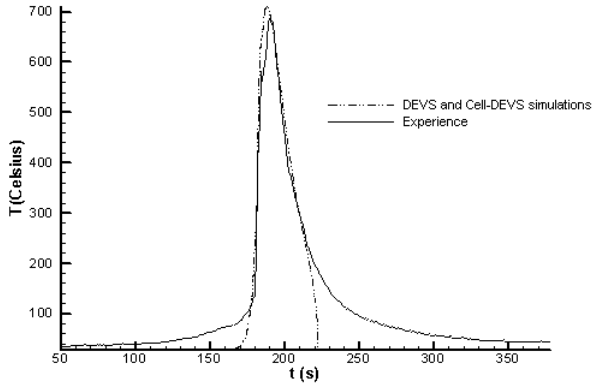


Figure 14. Comparison of the DEVS and Cell-DEVS simulation outputs with the experimentation for the temperature curve of a cell of the domain

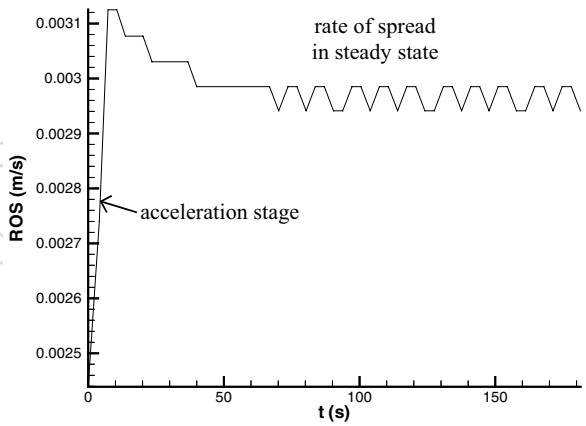


Figure 15. Predicted rate of spread of the fire front obtained with our mathematical fire spread model

Explicit timing delays embedded in Cell- DEVS models and the high-level language included in CD++ simplified the DEVS model implementation.

7. Conclusion

Based on a fundamental mathematical model, our simulations describe fire spreading precisely in each phase of the propagation from ignition to propagation. This allowed us to validate our fire spread model by using different simulation results (rate of spread of the fire front, temperature growth of a cell, and fire spread over the domain).

DEVS provided us with facilities for program evolvability. The developed code can easily be modified to include new aspects such as wind and slope [35]. This can simply be achieved by modifying the cell's output functions. Another advantage in using DEVS is that discrete events allow one to amount the calculation area to the flame front, thus optimizing model simulation for a large propagation domain.

Cell-DEVS improved the definition of the fire spread model. Development time related with the programming of timing control has been reduced, and the high-level specification language provided by CD++ allowed us to simplify the model specification. In fact, DEVS and Cell-DEVS model executions should be equivalent because Cell-DEVS uses DEVS simulation mechanisms. However, Cell-DEVS provides formal and implemented methods to automate and facilitate cellular modeling.

Digital simulation of continuous phenomena, such as reaction-diffusion, requires discretization. Classical numerical methods such as Euler, Runge-Kutta, Adams, and

so on are based on the discretization of time [36]. This approximation procedure results in a discrete time simulation model.

These models are so fine-grained that it is difficult to efficiently simulate them with modular and event-oriented approaches. Synchronization of numerous active cells overrules performance improvements of these asynchronous approaches for synchronous applications. Hence, the decision to use modular and discrete event approaches for cell space simulation is based on *how sparse* it is and *how often* cells need to be updated.

Quantized DEVS theory [37, 38] offers an alternative for classical discrete time algorithms. Using the quantization of state variables allows one to obtain a discrete event approximation of a continuous system. A significant event detector, called a *quantizer*, monitors its input and uses a logical condition to decide when a significant event change occurs. This results in reducing the number of messages. The problem, then, consists of a trade-off between accuracy and speed of computation. Interesting perspectives are currently explored for PDE simulations, reducing the time to reach the solution provided by conventional numerical methods [39**[*PLS. PROVIDE REF.*]**].

8. References

[1] Albright, D., and B. N. Meisner. 1999. Classification of fire simulation systems. *Fire Management Notes* 59 (1): 5-12.

[2] Porterie, B., D. Morvan, J. C. Loraud, and M. Larini. 2000. Fire spread through fuel beds: Modelling of wind aided fires and induced hydrodynamics. *Physics of Fluids* 12 (7): 1762-1872.

[3] Campbell, D. T. 1974. Downward causation. In *Hierarchically organized biological systems: Studies in the philosophy of biology*, edited by F. J. Ayala and T. Dobzhansky, 179-86. New York: Macmillan.

[4] Zeigler, B. P. 1976. *Theory of modeling and simulation*. New York: John Wiley.

[5] Balbi, J. H., P. A. Santoni, and J. L. Dupuy. 1999. Dynamic modelling of fire spread across a fuel bed. *International Journal of Wildland Fire* 9:275-84.

[6] Wolfram, S. 1994. *Cellular automata and complexity: Collected papers*. Reading, MA: Addison-Wesley.

[7] Santoni, P. A. 1998. Elaboration of an evolving calculation domain for the resolution of a fire spread model. *Numerical Heat Transfer, Part A* 33:279-98.

[8] Wainer, G., and N. Giambiasi. 2001. Application of the Cell-DEVS paradigm for cell spaces modeling and simulation. *SIMULATION* 76 (1): 22-39.**[*NOT CITED IN TEXT. PLS. ADD CITE*]**

[9] Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. *Theory of modeling and simulation*. 2nd ed. New York: Academic Press.

[10] Weber, R. O. 1990. Modelling fire spread through fuel beds. *Progress in Energy and Combustion Science* 17:67-82.

[11] McArthur, A. G. 1966. Weather and grassland fire behaviour. Australian Forest and Timber Bureau Leaflet No. 100, ***CITY?***.

[12] Rothermel, R. C. 1972. A mathematical model for predicting fire spread in wildland fuels. Forest Service Research Paper INT-115, U.S. Department of Agriculture, Washington, D.C.

[13] Veach, M. S., P. D. Coddington, and G. C. Fox. 1994. BURN: A simulation of forest fire propagation. Project Report for the Northeast Parallel Architectures Center, Research Experiences for Undergraduates Programme. Accessed from http://citeseer.nj.nec.com/119 736.html

[14] Finney, M. A. 1995. *FARSITE: Fire area simulator: Version 1.0—Users guide and technical documentation*. Missoula, MT: Systems for Environmental Management.

[15] Andrews, P. L., and C. D. Bevins. 1998. Update and expansion of the BEHAVE fire behavior prediction system. Technical report. Accessed from http://www.firelab.org/fbp/fbppubs/fbppdf/pandrews/update.pdf

[16] Coleman, J. R., and A. L. Sullivan. 1996. A real-time computer application for the prediction of fire spread across the Australian landscape. *SIMULATION* 67 (4): 230-40.

[17] Richards, G. D. 1990. An elliptical growth model of forest fire fronts and its numerical solution. *International Journal for Numerical Method Engineering* 30:1163-79.

[18] Berjak, S. G., and J. W. Hearne. 2002. An improved cellular automaton model for simulating fire in a spatially heterogeneous savanna system. *Ecological Modelling* 148:133-51.

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

[20] Ameghino, J., A. Troccoli, and G. Wainer. 2001. Models of complex physical systems using Cell-DEVS. In *Proceedings of Annual Simulation Symposium*, Seattle, WA.

[21] Barros, F., and G. L. Ball. 1998. Fire modeling using dynamic structure cellular automata. In *14th Conference on Fire and Forest Meteorology*, Luso, Portugal.

[22] Albini, F. A. 1985. A model for fire spread in wildland fuels by radiation. *Combustion Science and Technology* 42:229-58.

[23] Grishin, A. M. 1997. *Mathematical modeling of forest fires and new methods of fighting them*. Tomsk, Russia: Publishing House of Tomsk State University.

[24] Larini, M., F. Giroux, B. Porterie, and J. C. Loraud. 1997. A multiphase formulation for fire propagation in heterogeneous combustible media. *International Journal of Heat Mass Transactions* 41:881-97.

[25] Giroux, F. 1997. Contribution to fire spreading modelling: Multiphase approach of forest fires, development of a propellant fire in a semi-confined medium. Ph.D. diss., University of Provence, IUSTI, Aix-Marseille-I, France (in French).

[26] Dupuy, J. L., and M. Larini. 2001. Fire spread through a porous forest fuel bed: A radiative and convective model including fire-induced flow effects. *International of Wildland Fire* 9:155-72.

[27] Santoni, P. A. 1997. Propagation de feux de forêt, modélisation dynamique et résolution numérique, validation sur des feux de litière. Ph.D. diss., University of Corsica.

[28] Muzy, A., T. Marcelli, A. Aiello, P. A. Santoni, J. F. Santucci, and J. H. Balbi. 2001. An object oriented environment applied to a semi-physical model of fire spread across a fuel bed. In *ESS 2001–DEVS Workshop*, Marseille, France.

[29] Zeigler, B. P. 1984. *Multifaceted modeling and discrete event simulation*. London: Academic Press.

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

[31] Muzy, A., G. Wainer, E. Innocenti, A. Aiello, and J. F. Santucci. 2002. Comparing simulation methods for fire spreading across a fuel bed. In *Proceedings of AIS 2002—Simulation and Planning in High Autonomy Systems*, Lisbon, Portugal, pp. 219-24.

[32] Santoni, P. A., and J. H. Balbi. 1997. Numerical study of a two-dimensional reaction-diffusion model of fire spread across a fuel bed. *International Journal of Heat Mass Transfer* ***VOL:000-000***.

[33] Filippi, J. B., F. Chiari, and P. Bisgambiglia. 2002. Using JDEVS for the modeling and simulation of natural complex systems. In *Proceedings of AIS 2002—Simulation and Planning in High Autonomy Systems*, Lisbon, Portugal, pp. 317-22.

[34] Marcelli, T., A. Aiello, P. A. Santoni, and J. H. Balbi. 1999. An object oriented environment applied to the fire spread across a fuel bed under local wind condition. Paper presented at the Advanced Technology Workshop, ***MONTH?***, University of Corsica, France.

[35] Ainsworth, A. 1995. *Theory and numerics of ordinary and partial equations*. New York: Clarendon.

[36] Zeigler, B. P. 1998. DEVS theory of quantized systems. Advanced simulation technology thrust DARPA contract.
[37] Kofman, E. 2002. A second-order approximation for DEVS simulation of continuous systems. *SIMULATION* 78 (1): 76-90.
[38] Muzy, A., E. Innocenti, G. Wainer, A. Aiello, and J. F. Santucci. 2002. Cell-DEVS quantization techniques in a fire spreading application. In *Proceedings of the Winter Simulation Conference 2002*, San Diego, pp. 542-9.

Alexandre Muzy *received his M.Sc. from the University of Corsica, France, in 2001. He is currently pursuing a Ph.D. degree in the CNRS research laboratory UMR 6134 of the University of Corsica. His current research interests relate to the theory of modeling and simulation of complex systems, DEVS and Cell-DEVS formalisms, and the optimization of discrete event simulation for environmental problems.*

Eric Innocenti *received his M.Sc. from the University of Corsica, France, in 1998. He has worked as software developer for the past 3 years. He is currently pursuing a Ph.D. degree in the CNRS research laboratory UMR 6134 of the University of Corsica. His current research interests relate to the theory of modeling and simulation of complex systems, DEVS and Cell-DEVS formalisms, and parallel and distributed processing.*

Antoine Aiello *received a Ph.D. degree (1997) from the University of Corsica, France. He is an assistant professor at the University of Corsica. His current research interests relate to the modeling and simulation of natural complex systems and the development of multimedia architectures. He is a member of the CNRS research laboratory UMR 6134 of the University of Corsica.*

Jean-François Santucci *obtained a Ph.D. degree in March 1989 at the Université d'Aix-Marseille. He has been a professor in computer sciences since 1996 at the University of Corsica. His research interests are modeling and simulation of complex systems and high-level digital testing. He has published about 100 papers in international conferences and journals and has been responsible for several European and international industrial contracts. Since 1998, he has been the adjunct director of the CNRS Research Laboratory UMR CNRS 6134, University of Corsica.*

Gabriel Wainer *received M.Sc. (1993) and Ph.D. degrees (1998, with highest honors) at the Universidad de Buenos Aires, Argentina, and Université d'Aix-Marseille III, France. He is an assistant professor in systems and computer engineering at Carleton University (Ottawa, Ontario, Canada). He was an assistant professor in the Computer Sciences Department at the Universidad de Buenos Aires, Argentina, and a visiting research scholar at the Arizona Center of Integrated Modeling and Simulation (ACIMS, University of Arizona) and LSIS, CNRS, France. He has published more than 70 articles in the field of operating systems, real-time systems, and discrete event simulation. He is the author of a book on real-time systems and another on discrete event simulation.*