# A Non-Modular Modeling and Simulation Approach Based on DEVS for the Forest Fire Spread

M. Hamri
LSIS UMR CNRS 7296
Aix-Marseille university
Marseille, France
Email: amine.hamri@lsis.org

Y. Dahmani
EECE lab
Ibn Khaldoun university
Tiaret, Algeria
Email: dahmani_y@yahoo.fr

***Abstract*—Recently, the modeling and simulation of forest fire spread using discrete event formalisms have been intensively investigated. In this paper, we propose a non-modular approach vs. partial-modular and modular approaches based on Discrete EVent system Specification (DEVS) formalism to reduce exchanged messages between cells and to improve essentially performances of forest fire spread model. Note that the existing DEVS models simulate the forest fire spread for only small scale forests.**

***Index Terms*—forest fire spread M&S; DEVS.**

## I. Introduction

The Modeling and Simulation (M&S) formalisms are used to understand, represent, specify, and analyze the dynamic of systems. Often, these systems are very complex due to the fact that in a small temporal window there is a high number of variables which change values. Consequently, this complexity increases according to the simulation time.

Modeling the system dynamics is a hard task, particularly natural systems in which abstraction of the behaviors should be done to reduce the system complexity. Different methods and techniques were developed in order to improve the formulation of such systems. In the literature, two main categories are distinguished: analytic methods, which are difficult to grasp and M&S ones. Formally, a large variety of behaviors can be formulated mathematically. To learn about systems, we must take into account all involved variables and entities. Although, their dynamics reveals a complex formulation involved by these variables, the corresponding equations are unable to provide accurate results due to the increasing complexity of data.

The M&S is based on an experimental frame. The likeness between experimentation and M&S was the essence of this twinning [1], [2] offering the possibility of predicting the behavior of complex systems. Various approaches were defined to handle the two steps of M&S, depending on time-driven or event-driven.

In the application domain, due to the important damages caused by fire, governments employed the necessary humans and resources to limit these damages and intervention costs of firemen. The scientists on their side have provided efforts to understand and counter at best the forest fires. Consequently, many models were developed to firemen and decision makers to train them and define the efficient strategy.

The mathematical model of Rothermal is one of the viable models for forest fire. It employs a set of continuous variables interrelated (vegetation fuel, wind speed and direction, humidity, etc.), which influence the direction and speed of the forest fire spread. However, the Rothermal model is specific to north America terrains and can not be reused elsewhere.

On the other hand, simulation models are very accessible and easy to use. We find the two categories time-driven and event-driven to M&S of forest fire through the decomposition of forest in the two-dimensional space. The cellular automata were widely used in this field. The spread of the forest fire is based on simple rules executed at each time step. Some of these rules are extracted from the Rothermal model.

In event-driven simulation, Discrete EVent system Specification (DEVS) contributed to this field. The DEVS-Fire model proposed by the authors in [3] combines DEVS formalisms and Rothermal model and the shown results are very interesting. However, despite the use of a heap-based simulation engine to load and simulate only active cells at each simulation cycle and the enhance of neighbor cell ignition process (pre-schedule model vs. on-time schedule one), the corresponding models need an important heap memory and CPU time to execute the behavior of forest fire spread. Consequently, only a small set of active cells are allowed to be simulated, even if the authors note that the model is able to simulate wildfire with large scale.

In order to remedy the lack of such an approach in which each forest cell is modeled with an DEVS atomic model and for each active cell a simulator is invoked to produce the equivalent behavior, we propose to model the whole forest with a unique DEVS atomic model, which describes the fire spread like reality. The paper is organized as follows: Section 2 gives a recall on DEVS M&S and Section 3 discusses our proposal. Section 4 shows the object-oriented design of the new model of forest fire spread. Finally, we conclude on this work and we outline the perspectives.

## II. DEVS Principles

DEVS is one of the popular discrete event formalisms proposed in 70’s by Zeigler [4]. The DEVS M&S framework separates clearly modeling concerns from simulation ones. In fact, DEVS abstract simulator is useful to produce the behaviors of any model that respects the DEVS definitions. On the other hand, DEVS models are reused and coupled among them to make new DEVS models. Many research and practicable works were realized around this formalism thanks to its powerful expressiveness. This formalism has many extensions: GDEVS [5], Cell-DEVS [6], etc. and applications in different fields: forest fire spread, workflows, etc.

### A. DEVS Atomic Formalism

According to the literature on DEVS [7], the specification of a discrete event model is a structure, M, given by:

$M = (X, S, Y, \delta_{int}, \delta_{ext}, \lambda, D)$, where $X$ is the set of the external input events, $S$ the set of the sequential states, $Y$ the set of the output events, $\delta_{int}$ is the internal transition function which defines the state changes caused by internal events, $\delta_{ext}$ is the external transition function which specifies the state changes due to external events, $\lambda$ is the output function, and the function $D : S \rightarrow R^{+} \cup \infty$ represents the maximum length or the lifetime of a state. Thus, for a given state $s$, $D(s)$ represents the time during which the model will remain in state $s$ if no external event occurs.

### B. DEVS Coupled Formalism

DEVS promotes modular modeling to reduce the complexity of the system to describe. The DEVS coupled strucutre allows to formalize the modeled system in a set of inter-connected and reused components.

$MC = (X_{MC}, Y_{MC}, D_{MC}, M_{d|d \in D}, EIC, EOC, IC, Select)$, where

- $X_{MC}$: set of external events.
- $Y_{MC}$: set of output events.
- $D_{MC}$: set of components names.
- $M_d$: DEVS model named $d$.
- $EIC$: External Input Coupling relations.
- $EOC$: External Output Coupling relations.
- $IC$: Internal Coupling relations.
- $Select$: defines a priority between simultaneous events intended for different components.

This formalism is proved by the closure under coupling property, which shows that a DEVS coupled model is a DEVS atomic one. However, the formalism is less useful to describe large-scale systems like cellular DEVS, etc. Although, it seems possible at conceptual level, the corresponding computerized model can not be coded or at least simulated on computer due to the fact that the simulation needs a large heap memory. In [8], the authors show that the closure under coupling can be used to change the DEVS coupled model by its equivalent DEVS atomic. Consequently, the heap memory is reduced to the minimum to load the DEVS atomic model and its simulator, which consists of a root-coordinator and a basic simulator. However, the authors note that the transformation from DEVS coupled to atomic is not conducted automatically and intelligent modeling is recommended. This point constitutes our main goal.

### C. DEVS Simulator

The DEVS abstract simulator (see Fig. 1) consists of a root-coordinator, which manages the simulation time, sub-coordinators which dispatch messages according to the specific couplings of the coupled model that attempt to simulate and basic simulators related to atomic models. Each process behaves according to the received messages from parent and child processes.

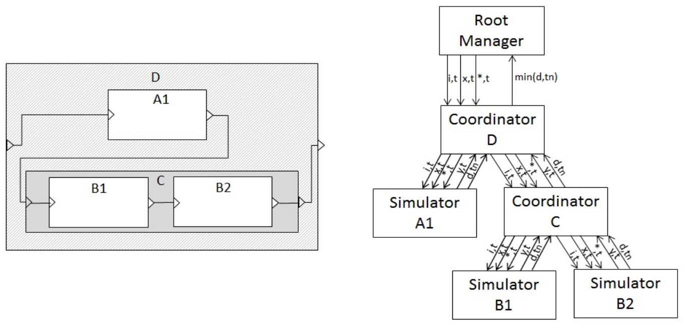


Fig. 1. DEVS abstract hierarchical simulator.

The classic structure of DEVS simulator is a hierarchical one, represented as a tree in which at top level is the root followed by the sub-coordinators created from DEVS coupled structure; then, at low level there are basic simulators related directly to the corresponding DEVS atomic models in order to execute the different functions $\delta_{int}, \delta_{ext}, \lambda$ and $D$. Fig. 1 illustrates this structure and messages transiting from a process to another.

## III. Related works

Many scientific works discuss the forest fire spread and propose models to anticipate the fire direction and calculate its Rate of Spread (ROS). in [9], Weber noted that three categories of such models are developed in the literature: statistical and Markovian models, empirical models like Rothermal and Albini models and Physical models, which consist on reproducing the forest fire spread characteristics with law rules. In the third category, the simulation constitutes an efficient means for M&S of the forest fire behavior. Two paradigms are very popular: Cellular Automata (CA) and Discrete Event Simulation (DES). CA are dynamical models in which the space of cells is a set of states and the time is an integer. The cells are arranged in a two-dimensional space and their shape is often square. Applying CA consists on spreading fire from a burning cell to its neighbors using Moore or von Newman neighborhood. Some works [10] used hexagonal cells to have more realistic simulations. In the field of DES, [3], [11], [12] used DEVS and its extensions Dynamic-Structure DEVS (DS-DEVS), Cellular DEVS, etc. to model and simulate forest fire spread. The main parameter in such models is the ROS, which allows computing delays (times) to ignite neighbor cells according to several cell parameters (fuel, slope, etc.) and weather.

As an example, DEVS-Fire [3] allows to model and simulate the spread and suppression of fire. It consists of three models: Rothermal model to compute the ROS according to natural parameters (wind, terrain, etc.), firefighting model to manage and execute the planned strategy to suppress fire and the forest fire spread model, which ignites unburned cells and deletes burned ones or saved ones by firemen. The last model is a DEVS coupled one, which describes the state of the forest. It decomposes the forest into equal limited cells (areas) according to two-dimensional space. Each cell is a DEVS atomic model, which has eight neighbors (except those situated on the boundaries) and all together form the forest. Each cell influences its neighbors with events sent out and received via ports, so an external modular coupling is involved. In order to enhance DEVS-Fire performances, DS-DEVS is used to create and delete cells dynamically. This adjustment is motivated by the fact that only a few number of cells are active during the simulation of fire spread. Consequently, for each simulation cycle useless messages like igniting passive cells (burned and noninflammable) cells are ignored and a reduced heap memory is allocated only for active active cells; passive cells will not be loaded. This main advantage is loosen when an important number of cells are active for a simulation cycle (for more details see [13], [14]).

The implementation of the DEVS cellular space of DEVS-Fire following the previous description is known under the modular approach in [15] and OnTime_Schedule model in [3]. The alternative implementation to the modular approach is the partial-modular one [15] (pre_Schedule model [3]). In this implementation, the number of simulation cycles to ignite neighbors of a burning cells is not eight cycles but only one due to the fact that the burning cell sends out the delays for which neighbors should ignite and not the event ignite. Consequently, less messages are exchanged between cells and simulation execution time is reduced.

However, these implementations are heavy by designing forest cells with identical DEVS atomic models to get on memory the cell state and to ignite neighbors on time or by pre_scheduling. In the next section, we propose a new model for the forest fire spread based on DEVS.

## IV. New DEVS Model for the Forest Fire Spread

Designing each forest cell in DEVS-Fire with an atomic model has the advantage that the approach remains modular. Consequently, cells could be coupled with other DEVS models. However, the use of external couplings via DEVS coupled leads to a large structure of simulation for cells which have the same behaviors. Although, the model is based on a dynamic structure in which only active cells are simulated; the model is limited to simulate a small set of active cells at the same time in order not to overload the heap memory and not to spend time on creating and deleting active and passive cells respectively in case of a large set of active cells.

### *A. Our Proposal*

In our modeling of the forest fire spread, we model the whole forest decomposed into cells with a unique DEVS atomic model. Each cell is a state variable of this model, holds a DEVS behavior and communicates with its neighbors through an internal couplings. In fact, the communication between cells is done directly inside the model, except events which influence the outside of model. We can model the natural parameters that influence the forest fire spread with other DEVS models coupled with the forest cell model or sensors which keep values from the real world or estimated by the user to update the simulation parameters.

The literature distinguishes three classes of parameters which set the ROS: vegetation type (caloric content, density, etc.); fuel properties (vegetation) and environmental parameters (wind speed, humidity and slope, etc.). The flaming fire evolves mainly according to the direction of the wind, its velocity and the relative humidity. The present model uses two relevant baseline parameters: wind velocity and relative humidity. The humidity influences the wild-land fire behavior by increasing the risk factor. Low relative humidity is an indicator of high fire danger. A dry and powerful wind, associated with a dry ground, enormously increase the fire spread.

Firstly, we identify two main categories of cells:

1) Nonflammable cell: can be a road, a surface of water or just an empty surface. It is designed with a static model.
2) Inflammable cell: is each area from the forest that is sensible to fire. According to its natural parameters and when the event ignite occurs in the corresponding area, the cell transients from the initial state unburned to burning to represent consuming fire; after some duration estimated from natural parameters, the cell goes to state ember to show the fact that small glowing piece of coal or wood in dying fire. Then, the cell reaches the state burned, which is the final combustion process. At this stage, the nonvolatile products and residue were formed when matter is burned (see Fig. 2).

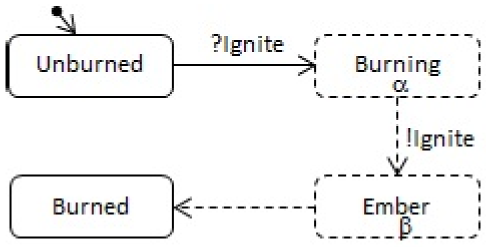


Fig. 2. DEVS behavior for inflammable cell.

Note that the difficulty of this model is the estimation of $\alpha$ and $\beta$ the lifetime of burning and ember states respectively. In our case, they are functions of wind velocity, humidity, and fuel area.

Other categories could be described, such as areas with risks (houses, plants, etc.) and introduced them into our model. Therefore, we should first describe the behavior of such areas with DEVS and then, we plug them inside the forest cell model.

### *B. Formal Model of the Forest Fire Spread*

The model of the forest fire spread that we propose is a DEVS atomic model, which collapses all cells inside one model to model forests with large scale and to enhance simulation performances. It is described as follows:

$ForestFireSpread = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, D)$
$X = \{(ignite, list), (wind, velocity), (humidity, value)\}$
$list$: the list of cells to ignite initially.
$Y = \emptyset$. There is no output event to send out.
$S = Cell \times Wind \times Humidity \times Fuel$

$Cell$ is the set of cell areas represented in the two-dimensional space. Each cell is identified by its position (line, column) and typed according to two categories nonflammable and inflammable mentioned above. Each cell keeps its current state and according to external and internal events state changes will occur to update the state of the concerning cell. Note that each cell has eight neighbors except those situated on the boundaries. $Wind$ is a global parameter. We assume that the wind is uniform for all cells. It can be supposed as a constant or change over time. $Wind$ is described with two parameters $direction(degree)$ and $speed(km/h)$. We choose this description to avoid the linguistic description for wind direction (from north, south, etc.) and to get an exact value. $Humidity$ is also a global parameter that we suppose uniform for all cells. $Fuel$ is specific to each cell. It is used to compute the times of burning and ember of the corresponding cell.

The functions $\delta_{ext}()$, $\delta_{int}()$ and $D()$ are shown in Fig. 3.

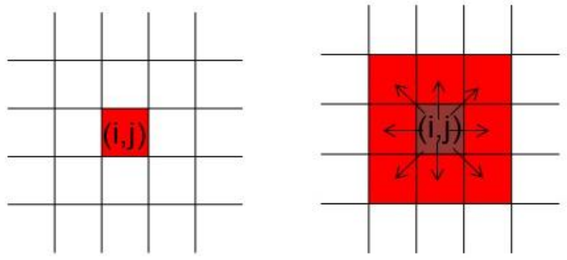
```
$\delta_{\mathbf{ext}}(\mathbf{s}, \mathbf{e}, \mathbf{x})$
        c, c´, c$_a$ : Cell
        if (x = ignite){
          for each c $\in$ ignite$_{list}$
            s$_c$ = $\delta_{ext}$ (s$_c$, e, ignite)
        }
        if(x = Wind)
          update Wind
        if (x = Humidity)
          update Humidity
        recompute the lifetime for each active cell $c$

$\delta_{\mathbf{int}}(\mathbf{s})$
        for each $c \in Cell${
          if (lifetime(c) = lifetime(Cell)){
            if (burning($s_c$))
              for each c´ $\in$ neighbor(c)
                s$_{c'}$ = $\delta_{ext}$(s$_{c'}$, lifetime(Cell), ignite)
          $s_c = \delta_{int}(s_c)$
          }
          else
             lifetime(c) = lifetime(c) - lifetime(Cell)
        }

$\mathbf{lifetime}(\mathbf{s})$
        return min {lifetime(c) | $c \in Cell$}
```

Fig. 3. DEVS atomic model of forest fire spread

In this model, we assume there is a direct communication between neighbor cells to exchange the events $ignite$. With a modular modeling, when a cell ignites its neighbors (see Fig. 4), it makes an internal state change to compute the output through the function $\lambda()$, which sends out the event $ignite$ for all neighbors. Then, the execution of $\delta_{ext}()$ of each neighbor puts fire in those cells. In our non-modular modeling, when a cell tries to ignite its neighbors through the autonomous state change, the internal function $\delta_{int}()$ calls to $\delta_{ext}()$ of each neighbor cell to ignite it. Such a communication optimizes the M&S structure and decreases the number of exchanged messages between cells.



(a) time = t $u.t$ (units of time) (b) time = t + $\alpha$ $u.t$

Fig. 4. Spreading of fire in the two dimensional plan using Moore neighborhood.

*1) Identification of Ignitable Neighbor Cells:* knowing that the wind direction impacts the spread fire and effectively not all neighbor cells are ignited when some cells burn, we develop a specific function to compute the potential neighbor cells which will be ignited. In fact, from aerial view, the spread fire takes the form of an ellipse according to the wind direction. Consequently, a burning cell does not necessarily ignite its eight neighbors necessarily but only some of them for a point of time. We can remark a neighbor situated in the same wind direction will be ignited unavoidably. However, a neighbor in the opposite wind direction will have less chance to be ignited. So, we can admit that the fire spread in neighbor cells follows a sigmoid curve according to the neighbor position and wind velocity (see Fig. 5).

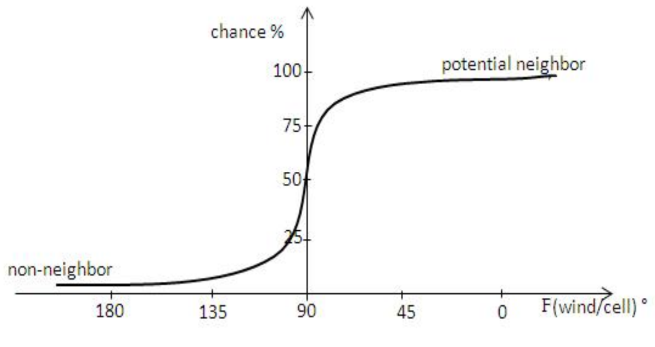


Fig. 5. Neighboring curve according to wind direction and cell position.

*2) Forest Mapping:* In our approach, the fire is spread on the forest map that the user chooses or defines graphically. This map defines the cell positions and fuels. Each cell has a fuel constant that the user attributes. A zero fuel for a cell means that the cell is nonflammable. However, a non-zero fuel for a cell means that the cell is inflammable; consequently, the corresponding cell behaves according to the DEVS model in Fig. 6.

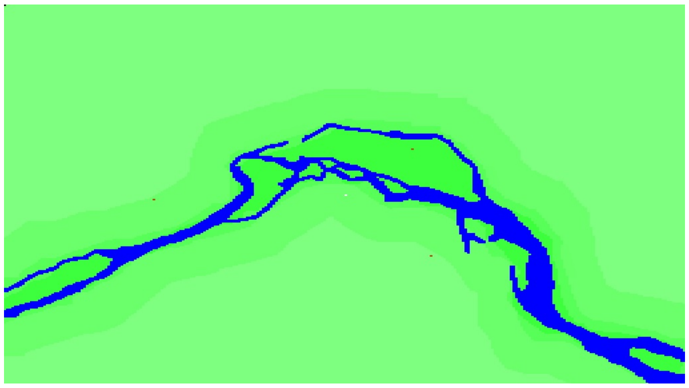


Fig. 6. Forest map with a river.

On this map, the user can identify cells with high risks to compute the reaching time to such cells, using the simulation of the forest fire spread.

## C. Simulation of the Proposed Model

In DEVS, the simulation mechanism is well formalized. The structure is made from the model to simulate. So, according to the size of the model to simulate, a computation of the needed heap memory could be estimated. In order to simulate our forest fire spread model, only a basic simulator and a connected root manager are needed to make simulations (see Fig. 7); so, a small size of heap memory is useful to simulate the proposed model.

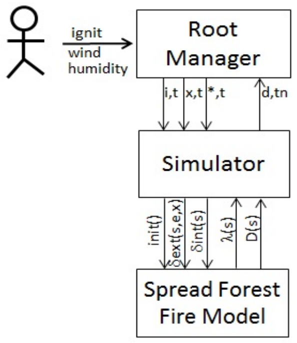


Fig. 7. DEVS simulation structure for non-modular (atomic) model.

The root can interact with user data to update the data of the model to simulate. The user can pause the simulation and then modify any parameter of the model: map data, wind direction and speed or put the fire in inflammable cells. Then, the user re-runs the simulation at the paused time with the new data.

## V. Design and Implementation of the Proposed Model

We design the proposed forest fire spread model in the object-oriented paradigm (see Fig. 8). Each cell is designed with a class which holds the corresponding behavior. The different cells are designed into classes `NFlammable` and `Inflammable` to implement the two cell categories non-flammable and inflammable, respectively. The uniform variables $Wind$ and $Humidity$ are sharable among all cell classes.

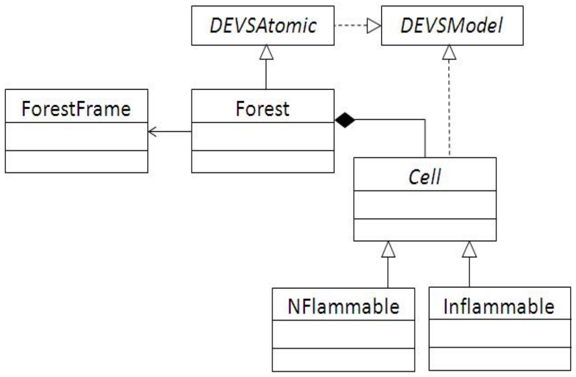


Fig. 8. Class diagram of the forest fire spread model.

This static structure could be replaced by a dynamic one in which we define two lists to hold active and passive cells separately. The active cell list holds cells for which the current state is active and the passive cell list holds cells for which the current state is passive. Thus, a cell migrates from a list to the other according to the cell lifetime. At the end, these lists avoid to visit all cells, i.e., to browse all the cell space to execute the internal state change $\delta_{int}(s)$ and computes the lifetime of current state of the model only from the active cells. Consequently, the run-time of the function $lifetime(s)$ will be reduced.

This design is extensible, it can integrate new cell categories with specific behaviors. The designer extends the class `Cell` and describes the DEVS behavior of the new cell class by implementing the interface `DEVSModel`. The class `Forest` instanciates cell objects according to the saved map description. The `Forest` object is a DEVS atomic model that will interact with the simulator to produce behavior. Note that the structure of this object is static, i.e., cell instances created first will remain until the end of the simulation. In addition, the class `Forest` notifies the graphical user class `ForestFrame` about the state changes occurring in cell objects in order to visualize the spread of fire on the chosen map.

The simulation core jDEVSPattern is developed according to the design proposed in [16] using DEVS simulation algorithms [7] and Java language. Often, the use of standard DEVS simulators certified by the community are well-suitable like DEVSJava, CD++, etc., for which many applications were modeled and simulated with success. However, we privilege this simulator for the following reasons:

1) the design of the simulator is based on software engineering design patterns. So, the software simulation reuses existing, approved and well-known solutions.
2) the validation of the simulator through the simulation of complex systems.

Fig. 9 shows the simulation of fire spread on the chosen map seen on Fig. 6 at the end stage.

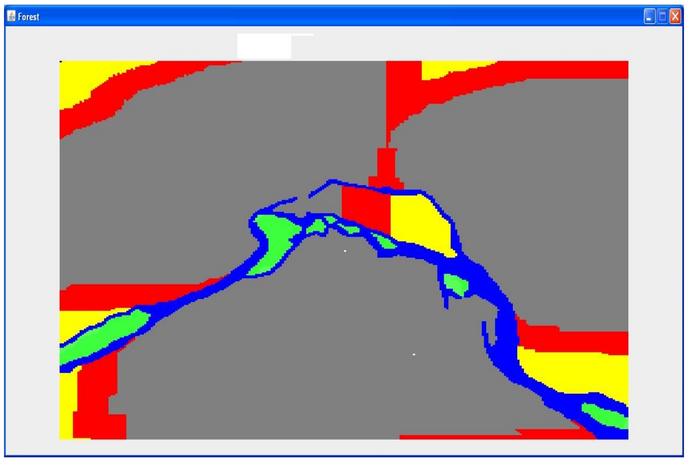


Fig. 9. Simulation of forest fire spread for a specific scenario.

In the above figure, we can note that zones enclosed by the river are still unburned due to the fact that the fire can not cross water area.

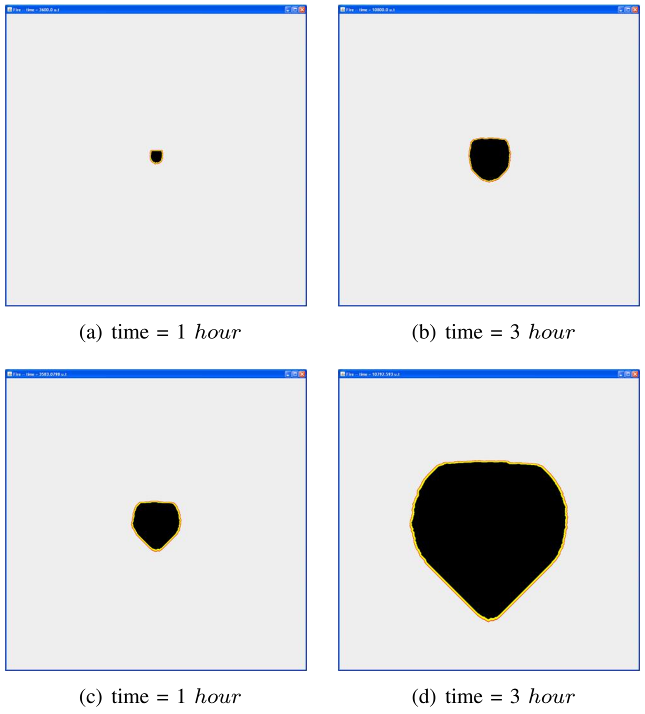
## VI. Simulation results and comparative study

### A. Simulation of the forest fire spread model

In order to experiment the proposed model for the forest fire spread and analyze the fire evolution, we execute two simulations with the same kind of vegetation and different wind speed. The wind direction is supposed blowing from north to south (0°). The forest square is 25 $km^2$ designed with cell space 1000x1000 $cells$ and cell size $5m$x$5m$. The simulations are conducted on a personal computer with the following characteristics: Windows XP professional, Intel Core(TM) 2 Duo CPU 3.0 GHz and 1.97 GB of RAM. The lifetime of states burning and ember are estimated from a well-known rule of firemen from the Mediterranean forests. They estimate the ROS as 3% from wind speed. Consequently, for the two expected scenarios in which the wind speeds are 3 $km/h$ and 10 $km/h$, we deduce the following values of ROS: 0.025 $m/s$ and 0.083 $m/s$, respectively. Note that for the two scenarios we ignite the cell situated in the middle (cell(500,500)).

In Fig. 10, we show the spread of forest fire at two stages (1 hour and 3 hour) after event $ignite$. We can see clearly in scenario 1 that the fire spreads slowly in case of a calm wind; comparing to the scenario 2 in which the fire spreads quickly causing an important burned area greater than in scenario 1. In addition, the shape of fire in scenario 1 is a circle, lightly flat from north side due to the fact that the wind blocks the fire spread in north direction. In scenario 2, the shape of fire has an ellipse one, with an oriented cone to south. In fact, this cone is driven by the wind direction and its form (pointed or large) depends on the wind speed.

The propagation rules of fire, that we use in the proposed model, are different from Rothermal rules, and developed by



(a) time = 1 $hour$ (b) time = 3 $hour$

(c) time = 1 $hour$ (d) time = 3 $hour$

Fig. 10. Fire spread on GUI `ForestFrame`.

using our own skills. Unfortunately, it is difficult to validate the forest fire spread model due to the lack of real data. An issue, that we will explore, consists in using fuzzy logic to well estimate the ROS according to firemen knowledge, and to make easier the validation process.

### B. Performance analysis

To analyze the time execution (second $s$) and resource allocation (mega-byte $MB$) of our model, we re-conduct the scenario 1 in which the simulation continues until there is no cell to ignite and all burning cells change state to unburned using different cell spaces 100x100, 200x200, 500x500, 1000x1000 and 2000x2000 $cells$. The cell (0, 0) is ignited initially.

TABLE I
Execution time of non-, partial- and modular models

| **Exec. Time (s)** | **100x100** | **200x200** | **500x500** | **1000x1000** | **2000x2000** |
|---|---|---|---|---|---|
| **non-modular** | 0.1 | 0.3 | 4.2 | 26.1 | 194.4 |
| **partial-modular** | 36.2 | 232.2 | 288.2 | 282.8 | - |
| **modular** | 154.6 | 1088.2 | 1320.5 | 1337.6 | - |

From Table I, we see that simulation of cell space size less than 1000x1000 $cells$, the execution time takes some few milliseconds. The enhanced simulation structure and the optimized model avoid sending out and receiving messages through ports and exchanged between the simulators and the coordinator to ignite the neighbor cells. This fact, leads to a minimum time CPU to execute the fire spread behavior. However, for the cell space size more than 1500x1500 $cells$, the execution time of the corresponding simulation is important (few seconds).

Comparing the execution times of partial-modular and modular approaches that were carried out on a laptop Toshiba with Intel Celeron 1.6 GHz CPU, 1.2 GB of heap memory and Windows XP operating system using DEVSJava version 3.0 and end simulation at tn = 11 $hours$ (the given results are extracted from [15]), our approach is still very efficient, although our simulations are carried out on personal computer well-equipped (Intel Core(TM) 2 Duo CPU 3.0). In fact, the design of our model, which is based on a non-modular approach and in which cells are modeled with state variables, makes the main difference with the approaches that design cells with atomic models.

The simulation of our model uses a small size of heap memory, which increases essentially according to the size of cell space (cell space with 1000x1000 $cells$ consumes 44 $MB$). A dynamic structure could be used to design the cell space instead of a static one by keeping temporary in memory only active cells, in order to optimize the computations and to manage efficiently the heap memory.

## VII. Conclusion

In this paper, we privileged the non-modular approach to model the forest fire spread using DEVS instead of partial-modular and modular ones; so, we avoid in our modeling the classical rule that consists on designing each cell with an atomic model and we decide to incorporate it as a state variable of the forest fire spread model. Note that this model is designed using the object-oriented paradigm, which allows to design this state variable with an object instance.

The simulation of the proposed model gives correct results by analyzing the simple conducted scenarios. On the other hand, simulation performances (execution time and heap memory) are more advantageous than those given by the modular and partial-modular implementations of DEVS-Fire. In the near future, we will compare our non-modular model with other forest fire spread models and real fire cases in order to validate the given results. The main difficulty will consist in reproducing real fires for which it is difficult to collect accurate data (fuels, landscape, wind direction, and wind speed) [17].

Currently, we are working to introduce fuzzy logic rules of terrain (slope and aspect), weather and vegetation to improve the computation of the ROS parameter which influences mainly the behavior of fire and to solve the problem of accurate data for modeling the input data of the simulation. This will give more significant results and insights to firemen.

## References

[1] L. V. Bertalanffy, General System Theory. Dunod edition, 1973.

[2] P. A. Fishwick, Simulation model design and execution: building digital worlds. Prentice hall, 1995.

[3] X. Hu, Y. Sun, and L. Ntaimo, "Devs-fire: Design and application of formal discrete event wildfire spread and suppression models," Simulation: Transactions of the Society for Modeling and Simulation International, vol. 88, no. 3, March 2012, pp. 259–279.

[4] B. P. Zeigler, Theory of Modeling and Simulation. Wiley&Son, 1976.

[5] N. Giambiasi, B. Escudé, and S. Ghosh, "Gdevs: A generalized discrete event specification for accurate modeling of dynamic systems," Simulation: Transactions of the Society for Modeling and Simulation International, vol. 17, no. 3, September 2000, pp. 120–134.

[6] G. Wainer and N. Giambiasi, "Application of the cell-devs paradigm for cell spaces modelling and simulation," Simulation: Transactions of the Society for Modeling and Simulation International, vol. 76, no. 1, January 2001, pp. 22–39.

[7] B. P. Zeigler, H. Praehofer, and T. G. Kim, Theory of Modeling and Simulation. Academic Press, 2000.

[8] F. A. Shiginah and B. P. Zeigler, "A new cell space devs specification: Revewing the parallel devs formalism seeking fast cell space simulations," Simulation Modelling Practice and Theory, vol. 19, no. 5, May 2011, pp. 1267–1279.

[9] R. O. Weber, "Modelling fire spread through fuel beds," Progress in Energy and Combustion Science, vol. 17, no. 1, 1991, pp. 67–82.

[10] L. Hernandez Encinas, S. Hoya White, A. Martin del Rey, and G. Rodriguez Sanchez, "Modelling forest fire spread using hexagonal cellular automata," Applied Mathematical Modelling, vol. 31, no. 6, June 2007, pp. 1213–1227.

[11] A. Muzy, J. J. Nutaro, B. P. Zeigler, and P. Coquillard, "Modeling and simulation of fire spreading through the activity tracking paradigm," Ecological Modelling, vol. 219, no. 1–2, November 2008, pp. 212–225.

[12] B. Nader, J. B. Filippi, and P. A. Bisgambiglia, "An experimental frame for the simulation of forest fire spread," in Proceedings of the 2011 Winter Simulation Conference, December 2011, pp. 1010–1022.

[13] Y. Sun and X. Hu, "Performance measurement of devs dynamic structure on forest fire spread simulation," in Proceedings of the AI, Simulation and Planning in High Autonomy Systems (AIS 2007), February 2007, pp. 75–80.

[14] ——, "Performance measurement of dynamic structure devs for large-scale cellular space models," Simulation: Transactions of the Society for Modeling and Simulation International, vol. 85, no. 5, May 2009, pp. 335–351.

[15] ——, "Partial-modular devs for improving performance of cellular space wildfire spread simulation," in Proceedings of the 2008 Winter Simulation Conference, December 2008, pp. 1038–1046.

[16] M. Hamri and L. Baati, "On using design patterns for devs modeling and simulation tools," in Proceedings of the 2010 Spring Simulation Multiconference-Symposium Theory of Modeling Simulation-DEVS Integrative MS symposium, April 2010, doi:10.1145/1878537.1878664.

[17] F. Gu, X. Hu, and L. Ntaimo, "Towards validation of devs-fire wildfire simulation model," in Proceedings of the 2008 Spring simulation multiconference, April 2008, pp. 355–361.