Active-DEVS: a computational model for the simulation of forest fire propagation*

Eric Innocenti
Alexandre Muzy
Antoine Aïello
Jean-François Santucci
SPE – UMR CNRS 6134
Pascal Paoli University
Campus Grossetti BP 52, 20250
Corti, France
e-mail : ino@univ-corse.fr

David R.C. HILL
ISIMA/LIMOS UMR CNRS 6158
Blaise Pascal University
Campus des Cézeaux BP 10125, 63177
Aubière Cedex, France
e-mail : drch@isima.fr

Abstract - *This paper deals with the design of an efficient object model for propagation phenomena. It is applied to the phenomenological model developed at the University of Corsica, within the context of simulation of vegetation fires. The objective is to simulate large-scale fire propagation, and on the longer term to develop a decision aid tool to guide forest firemen and managers. Based on both cellular automata and discrete Event Specification (DEVS) formalisms, a new kind of model, called Active-DEVS, is specified. Modeling methods based on enhanced cellular automata facilitate both spatial dynamic expression of propagation phenomena, and parallel architectures exploitation. However, such environments usually lack the ability to integrate easy component modifications. The DEVS formalism makes it possible to exploit the cellular models efficiently whatever their dimensions, and to reduce simulation times considerably. A simulation framework is developed to implement and compare Active-DEVS model and classical Discrete Time System Specification (DTSS) models. This framework relies on designs patterns, and thus keeps a modular, elegant and adaptable design.*

Keywords: Fire spread simulation, DEVS formalism, design patterns, object oriented programming.

1 Introduction

This paper describes Active-DEVS an efficient model for large-scale propagation phenomena. The reliability of this approach is shown by simulating an original fire spreading model, designed at the University of Corsica, which is based on physical specifications [1]. The implementation of active-DEVS models uses a combination of recent modeling and discrete event techniques [2]. The execution time is reduced by limiting computations to active components, which is then independent of the domain size. Based on a formal specification, the structure of the model avoids modeling ambiguities and reduces the testing phase. This approach is intended in longer term to be integrated in a software tool to aid decision-making for firemen and fire managers. Exploring a complex phenomenon through simulation necessitates a constant modification of the software architecture as scientists learn more about the phenomenon. Designing a reusable, extensible, and adaptable architecture is a difficult task and design patterns can be used to help achieving this goal [3, 4]. During the development of the simulation famework, profiles of classes and collaborations of objects are used in order to have a modular, elegant and adaptable design. The architecture is presented hereafter and constitutes with the Active-DEVS model the main contribution of this communication. The framework developed aims at structuring reliable simulation systems by allowing optimized simulation models to be composed in a flexible manner. As example, a comparison between the efficiency of the Active-DEVS model of simulation and traditional DTSS model for propagation phenomena is carried out.

The first section is devoted to the presentation of the phenomenological model of propagation used. The second section deals with discrete time computing of modeling and propagation simulation of fire vegetation. The third section underlines the advantages of apprehending simulation using an approach based on discrete events. The fourth section is dedicated to the presentation of the extensible object-oriented framework for simulating fire propagation in a virtual laboratory. The abstract factory pattern can be used to build an extensible object oriented framework for simulation experiments. We overcome the limitation of inheritance, when trying to flexibly compose simulation components. It makes it possible thereafter to compare Active-DEVS model, with the discrete time model.

* 0-7803-8566-7/04/$20.00 © 2004 IEEE

The comparison is carried out in a fifth section. The results obtained are then commented. Finally, the last section concludes on the approach considered and presents the future research prospects.

2 The physical model of propagation

Developing an effective fire spread model remains a challenge for research. To improve the decision aid ability of current fire spread models, physical models are currently used. The physical model we use [1] has been validated against numerous experiments [1, 8, 9, 10, and 12]. The data used in the model comes from real experiments (evolution of a front of flame in a domain of 1 m^2 of pine needles, without slope, nor wind). The model will be tested this year after having experimented fires in much larger areas. For that, we will suppose that the physical mechanisms describing the propagation, identified and modeled on laboratory-scale, remain valid at the scale of the domain. A preliminary numerical study makes it possible to solve the model using the explicit method of finite differences, leading to a system of differential equations. The domain of propagation is then subdivided in elementary components which constitute the ground and the plants, each one being described by the following algebraic equation (1):

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

where T_{ij} represent the temperature of a node of the grid. The coefficients a, b, c and d depend on the time step and the space step considered. The parameters of the model are given starting from experimental statements of temperature obtained according to time. The cellular division of space generates problems in extreme cases. In order to solve them a fixed value is given to edge cells [5]. The numerical results were compared with experimental data from various ignitions and the quality of the predictions is remarkable [1].

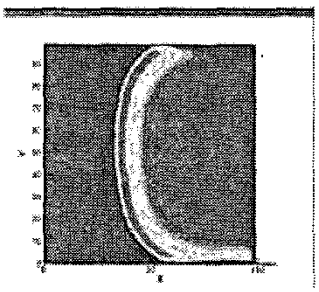

Figure 1. The simulation of the propagation of vegetation fire across a heterogeneous medium.

However, the precision of these models makes them difficult to be simulated under real-time constraints. Moreover, fire spread complexity requires progressively refined model specifications according to the simulation results. The corresponding simulation code has to be modified easily to reduce both implementation and testing phases. Nevertheless, the corresponding simulation code lacked in: (1) easy integration of model modifications as research advances (new ground vegetation, influence of both slope and wind, etc.), (2) meeting real-time deadlines and (3) optimally exploiting the inherent parallelism of cellular models [6]. Hence, an object-oriented framework is developed to get round these difficulties. It integrates simulation components which are designed in order to support the evolutionary nature of their structure and their behavior. It is presented hereafter and constitutes with the Active-DEVS model, one of the main contributions of this article. An example of fire spread simulation obtained using an Active-DEVS model is provided in figure 1.

3 Discrete time fire simulation

Discrete time models based on space oriented approaches are often most intuitive to represent the dynamic systems [7]. They are formalized using algorithms which specify a stepwise execution of the simulation. At an instant 't', the model is in a particular state S and defines how its state variables will change. The next state depends often on the current state S, but it also depends on the influence of its environment. The computational domain is cut into regular cells and each one is associated to a behavior. Each cellular element is influenced by the cells of its neighborhood. At each time step, all the elements of the domain are scanned, and the state of each cell at 't+1' time step, depends on the state of each cells of its neighborhood at the time 't' step. The system is type of cellular automata network.

3.1 Discrete Time System Specification

A discrete time system is a structure (2) :

$$DTSS = \langle X, Y, Q, \delta, \lambda, h \rangle \quad (2)$$

where,

- X , is the set of input ports,
- Y , is the set of output ports,
- Q , is the set of states
- δ , is the state transition function,
- λ , is the output function,
- h , is the simulation step.

3.2 DTSS modeling of fire propagation

The structure of the DTSS model of propagation is inspired by cellular automata, which is described like a multicomponent [2]. It has an input port and an output port. The input port generates the execution of the

phenomenon and the results of the simulation are observed on the output port. The associated simulator must scan the components which model the domain of propagation, in order to update their individual states. This operation results from the execution of the individual functions of transition. Moreover, it must apply the individual functions of output, which define together the output of the multicomponent system. This latter has a global function of state transition δ and a global function of output λ. It is composed of cellular elements $C_{i,j}$, each of them implementing respectively, the local transition function $\delta_{i,j}$ and the local output function $\lambda_{i,j}$. The behavior of the model is described according to the DTSS formalism as depicted in the algorithm 1.

```
δ :
   Ci,j  //Cellular components (propagation
domain)
   Ii,j //set influencing the component Ci,j
   tnext //Simulation Virtual time
   //Domain ignition at t=0
   For all Ci,j of the domain Do
      If not Border Element Then
         //-->recover t° of influencing cells
         UpdateNeighb(Ii,j)//Neighborhood upd.
         Ci,j->λ (Ii,j);  // Output of Ci,j
         Ci,j->δ (tnext);  //Ci,j Transition func.
      End If
   Enf For
   tnext=tnext+h
End Global Transition Function
λ :
  Ci,j   //Cellular components of the domain
  nextT //Temperature of the i,j comp. at t+h

  //-->Domain update by rocking T°next
  For All Ci,j of the domain Do
     //-->Get T° for t+h
     nextT= Ci,j -> getNextTemperature();
     //-->Temperature update
     Ci,j -> setTemperature(nextT);
  End For
End Global Output Function
```

Algorithm 1. DTSS model specification for simulating propagation of vegetation fire.

3.3 General algorithm of DTSS simulation

The specification of a discrete time model requires the definition of transition rules. From these rules, the model in a current state Q, carries out the events which come from its environment through its input ports, and determines its future state. The state of the model evolves in a discrete way on a temporal basis where time increments by constant step of times h, which are multiple periods of time like second, hour or year. If 'q' is the state of the model at time 't' and 'x' an input at time 't', then the state of the model at time 't+h' is given by $\delta(q,x)$ and the output 'y' at time 't' is given by $\lambda(q)$. Here, δ and λ represent respectively the state and output transition function of the discrete time model, specified before.

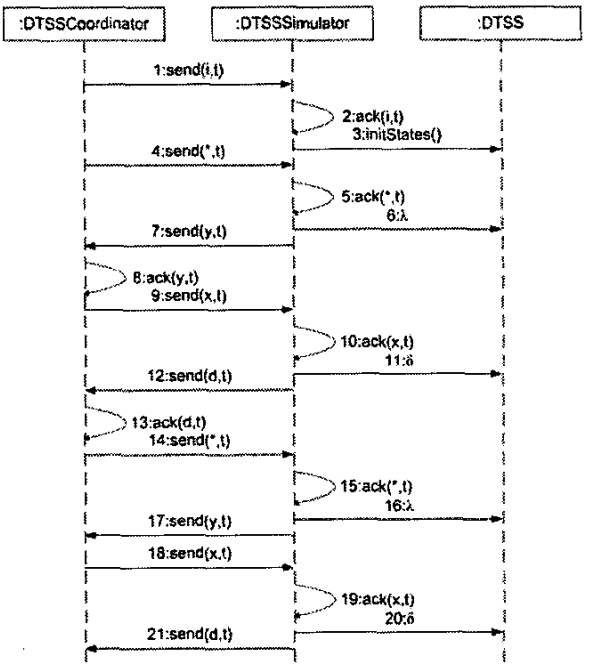

Figure 2. UML sequence diagram of the discrete time simulation for simulating the propagation of vegetation fire.

The global simulator is a tree structure of control; it forms the tree of the simulation. This tree is responsible for the exploitation of the model attached. The leaves of this tree constitute the processors, i.e. the simulators specific to the different components of the model. Simulation is carried out here, thanks to the use of processors like the DTSS coordinators and the DTSS simulators.

When the simulator receives a '* message', all the ouput values of the components are calculated, and the general ouput of the system is calculated by the parent coordinator, following the reception of a 'y message'. The reception of a 'x message' by the simulator, involves the computation of the new states, as well as the storage of the elementary states 'q'$_d$'. For that the elementary transition functions δ_d of each component are requested. The current simulation time is then increased to one unit and the state variables of the different components are updated, from the values stored in the temporary state variables. The simulation algorithm is depicted in figure 2, for a more precise description please refer to [2].

The DTSS Coordinator associated to the DTSS simulator has the role of initiating the stepwise execution of the whole simulation cycles. It achieves this task while sending '*messages', then 'x messages' to the subordinate simulator at each time step.

4 Discrete events fire simulation

This model schedules events on a restricted set of cells (components). At each time step, a list of active cells is defined. It results from the application of the rule answering to the criterion stated by Zeigler [2]: « A cell will not change state at the next state transition time, if none of its neighboring cells changed state at the current state transition time. The event-based simulation procedure follows from this logic... ». Then the list is updated by seeking the cells that will change state at the next time step. The computations are thus reduced to a handful of active cells.

4.1 Active-DEVS System Specification

To reduce execution time, the simulation relies on discrete events techniques. The latter is applied to each individual component of the model, which define the Active-DEVS system structure (3) :

$$Active\text{-}DEVS = \langle X, Y, S, \delta_{ext}, \delta_{int}, ta, \lambda, D \rangle \quad (3)$$

where,

X, is the set of input ports,

Y, is the set of output ports,

S, is the set of states which characterizes the Active-DEVS model,

δ_{ext} , is the external transition function, which specifies the behavior of the Active-DEVS model, when an external event occurs;

δ_{int} , is the internal transition function, which specifies the new state of the Active-DEVS model, after the time ta() has elapsed;

ta , is the time advance function, which schedules internal transitions;

λ , is the output function, which is carried out before the internal transition function and generates the response of the Active-DEVS model;

D, is the list of the behavioral elementary components of the Active-DEVS model.

The dynamic interpretation of the behavior of the Active-DEVS model is related to the atomic DEVS models described in [11]. Furthermore, the whole methodology allows several Active-DEVS models to be simulated in parallel. Their synchronizations can be simply managed thanks to the time advance function ta(). Thus, Active-DEVS models represent the complex behavior of a part of the domain, described structurally on the basis of a structure inspired from the cellular automata, and restrict computations to the active cellular elements.

4.2 Active-DEVS modeling of fire propagation

An Active-DEVS model provides a global description of the dynamic behavior of the different components which represents the simulation domain. It has an input port and an output put, through which it interacts with the external environment of the system. Its working is close to the working of the atomic components which are described by the DEVS formalism; also, it allows the deployment of chains of discrete events simulations and facilitates the parallel simulations of several propagation domains [8]. When an external event occurs on the input port of an Active-DEVS model, this latter reacts within the external transition function state δ_{ext}, which implements the response of the model. This response occurs following internal events which result to the execution of the internal transition function state δ_{int}. The behavior of the model is translated according to the Active-DEVS specification as depicted in the algorithm 2.

δ_{ext} :

```
// Model activation (if in passive mode)
If phase = passive Then
   phase ← active
   s ← 1
   t_last ● t_next
   t_next = t_last + ta()
   e ← 0
Else
   Display « Synchronisation error ! »
End if
End External Transition Function
```

δ_{int} :

```
SetOfActiveCells
nextT // temp. of the component i,j at t+h
C_i,j   // Cellular component of domain
I_k     // Set influencing C_i,j

// If the model is active, it evolves
If phase = active Then
   For All C_i,j of SetOfActiveCells Do
      If C_i,j->getTemperature()> T° Ambient
         and C_i,j->InFront=true
      Then
         C_i,j.->InFront ← False
         For All I_k of C_i,j
            If not border component
               and not active Then
               I_k ● « active »
               I_k ● « front component »
               SetOfActiveCells -> add(I_k)
                End If
```

```
            End For
         End If
      End For
      For All C_i,j in SetOfActiveCells Do
         If C_i,j is active
         Then
            //-->Get T° at t+h
            next←C_i,j->getNextTemperature();
            //-->Update temperature
            C_i,j->setTemperature(nextT);
         End If
      End For
      phase ← « passive »
      s ← 1
      e ← 0
      t_next ← t_next + ta()
   Else
      Display «  Synchronisation Error ! »
   End If
End Internal Transition Function

λ :
   Send  a  '(y,t)-message'  to  the  parent
simulator
End Output Function

ta() :
   Send h
End Advance Time Function
```

Algorithm 2. Active-DEVS model specification for the simulation of vegetation fire propagation.

The Active-DEVS model uses a list which references the cells of the fire front. It describes two levels of abstraction. At the highest level, it represents the overall fire propagation, within the domain. At the lower level, it describes the local propagation phenomena. Computation is restricted to a handle of components called 'active components'. Each component is clearly identified in the propagation domain, and their management is based on the list of references. The simulation of the front fire, then, is carried out with the help of iterators, which move along the list. This can be implemented with the C++ STL (Standard Template Library).

4.3 General algorithm of Active-DEVS simulation

The Active-DEVS model allows the restriction of scans only to the cellular components which are programmed by the next event. The list of the active components reference front cells at an instant 't', in order to apply the transition rules which permit to update the propagation domain. All the specified actions are deduced from the virtual time 't' and are executed simultaneously at virtual time 't+1'. The global simulation algorithm is schematized figure 3. At each time step, each front component is tested in order to update the list of active elements. An addition, it is always preceded by the activation of the property frontCell of the domain components, in order to avoid the double references. The additions are done at the end of the list, in order to not disturb the loop that it is initiated on this latter.

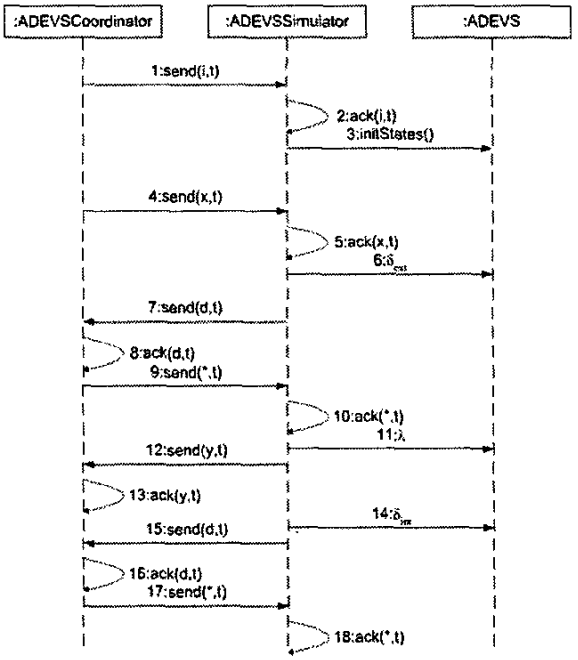

Figure 3. UML sequence diagram of the discrete event simulation of a vegetation fire propagation.

5 The simulation framework

Virtual laboratories for fire simulation experiments are built using an extensible object-oriented framework. Profiles of classes and collaborations of objects of the simulation framework have been considered to provide a modular, elegant and adaptable design. The framework aims at structuring reliable simulation systems by allowing optimized simulation models to be composed in a flexible manner. In fact, the capacity to adapt the architecture to the different physical models implies to take into account the possible evolution of coordinators, simulators and models. In our case, using a model implies both the use of a coordinator and a simulator. Coordinators and simulators must be created differently according to the specified model. It is thus necessary to be able to build the same tree structure, but on the basis of different implementations. The issue here is to make possible for the root coordinator to choose a model, a coordinator and a simulator, then to build the object architecture of the simulation, using a simple interface. A possible solution relies on the abstract factory patterns, which is used to get round the limitations of inheritance as far as the complexity of programming. The structure obtained facilitates the

creation of the simulation models, as well as their related simulators and coordinators [3]. The prototype is written in C++, and is fully operational for large-scale experiments. The interfaces are developed using the factory pattern, instanciations of related models and processors are submitted to sub-classes. Three hierarchies of parallel classes are inter-connected: Model class, Simulator class and Coordinator class, as described in figure 4.

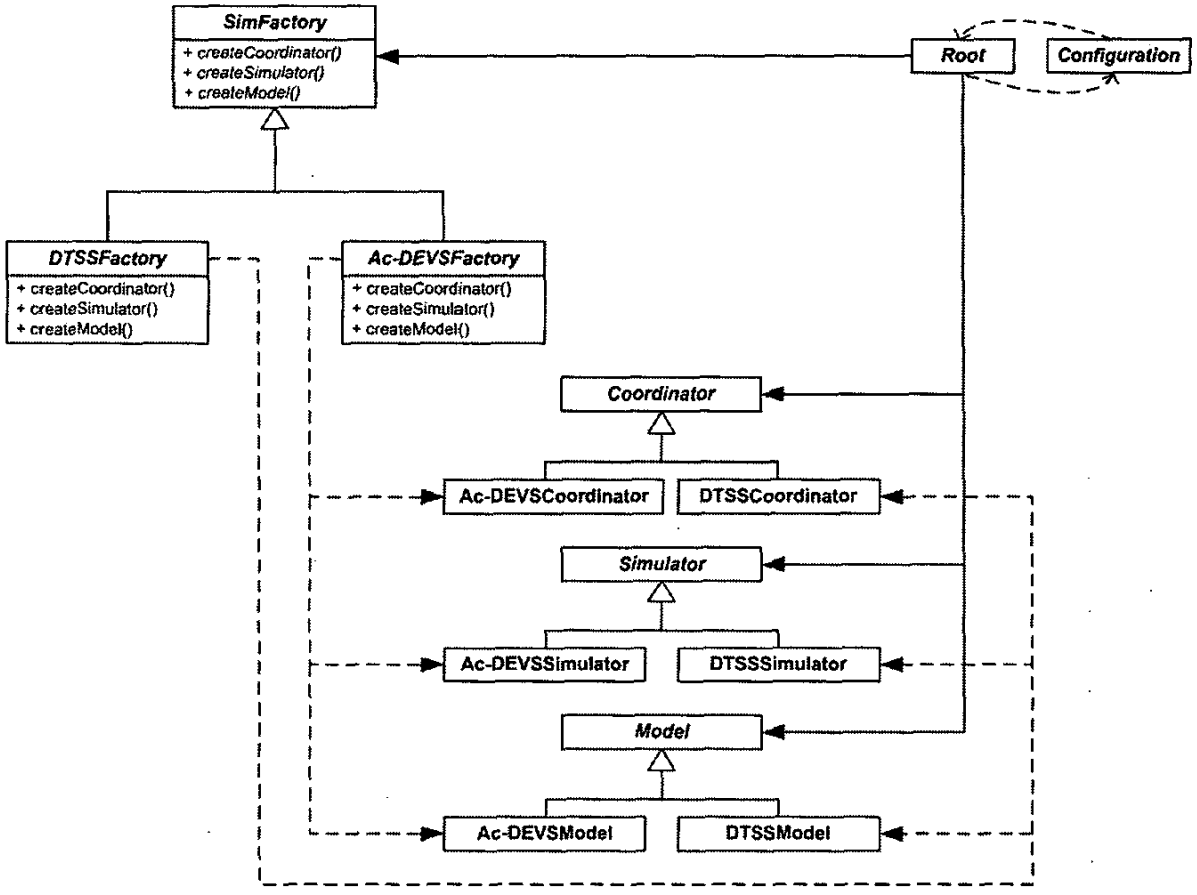

Figure 4. The abstract factory pattern in the simulation framework.

The architecture is independent from the way of creating, composing and representing, the models, the simulators and the coordinators. The root coordinator builds the simulation tree using as starting point the different families of models and algorithms available. A library of simulation tools is then made up. The parent root coordinator does not know the implementation classes of the families of models and processors (simulators and coordinators). The permutation between the latter becomes simple, and the consistency inside the models and of the processors is reinforced. In that case, during the addition or the modification of the simulation encapsulated tools, only the SimFactory objects are modified.

6 Experimentation

Discrete time simulation is not effective in the case of large-scale simulations of fire spreading. Indeed, the simulation times obtained are higher than those obtained using the discrete events model. Moreover, the increase of the size of the simulation domains corroborates this report. The Active-DEVS model developed guarantees the best possible simulation times, limiting computations to the active components of the simulation. Validations of future large-scale experiments will start from this point.

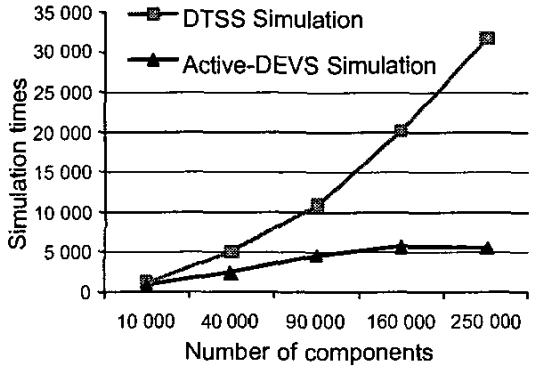

Figure 5. Comparison of execution time s for a real propagation lasting 1000 seconds.

7 Conclusion

In this document, the Active-DEVS model inspired from the DEVS formalism is presented. It simplifies the modeling of propagation phenomena. Its efficiency is compared to the traditional discrete time model, which it is described by the DTSS formalism. The Active-DEVS model is much more efficient than the classical DTSS model, however its performance is not yet good enough for a real time use, but we are currently working on performance evaluations and code optimization [12]. The object oriented framework we proposed provides the basic components for the construction of discrete event cellular simulations. It is designed to be flexible, extensible and it integrates the Active-DEVS and DTSS models. It is adapted to the issue of our work, has a weak coupling and a strong global cohesion, and it also helps in developing new model specifications. The framework is designed to ease programmers in building reliable simulation systems and is based on design patterns as architecture elements in addition to basic classes. With this framework the simulation system is composed of processors and propagation models, and we have the ability to designate each component and to experiment for various specifications. Moreover, the discrete techniques of simulation confer reduced simulation times, compared to the traditional propagation models [10]. We plan now to improve the framework's capabilities and to model new physical experiments. This will be achieved using the Active-DEVS model and its supporting framework as a foundation for a parallel and distributed simulation environment, which is currently under development [8].

References

[1] P.A. Santoni, J.H. Balbi, et J.L. Dupuy. "Dynamic modelling of upslope fire growth", *International Journal of Wildland Fire*, Vol 9, No. 4, pp. 285-292, 1998.

[2] Bernard P. Zeigler, Herbert Praehofer, Tag Gon Kim. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, Academic Press, 2000.

[3] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley. 1994.

[4] Alan Shalloway, James R. Trott. *Design patterns par la pratique*. Technologies objet. Eyrolles. 2002.

[5] Patrick Coquillard, David R.C. Hill. *Modélisation et Simulation d'Ecosystèmes. Des modèles déterministes aux simulations à évènements discrets*, Masson, 1997.

[6] J. Jorba, T. Margalef, E. Luque, J.C.S. Andre, D.X. Viegas. "Parallel Approach to the simulation of forest Fire Propagation", Proc. Environmental Communication in the Information Society, 16th International Conference, Informatics for Environmental Protection, pp. 25-27, September 2002.

[7] C. Lett, "Modélisation et simulation de la dynamique des écosystèmes forestiers: des modèles agrégés aux modèles individuels spatialisés". Thèse de doctorat en sciences du vivant. Université Louis Pasteur – Strasbourg 1. 1999.

[8] E. Innocenti, A. Muzy, A. Aïello, J. F. Santucci, and D.R.C. Hill, "Design of a Multithreaded Parallel Model for Fire Spread", Proc. Simulation in Industry, 15 th European Simulation Symposium, SCS European Council, pp. 104-109, November, 2003.

[9] J.L. Dupuy, "Slope and fuel load effects on fire behavior : laboratory experiments in pine needles fuel beds". *International Journal of Wildland Fire*, vol. 5, pp. 153-164. 1995.

[10] A. Muzy, G. Wainer, E. Innocenti, A. Aiello, J.F. Santucci. "Comparing simulation methods for fire spreading across a fuel bed", Proc. AIS, Simulation and planning in high autonomy systems, Lisbon, Portugal. pp. , 2002.

[11] Bernard P. Zeigler, *Theory of Modelling and Simulation*. Wiley, New York, NY. 1976.

[12] A. Muzy, E. Innocenti, G. Wainer, A. Aiello and J.F. Santucci, "Cell-DEVS quantization techniques in a fire spreading application", Proc. the Winter Simulation Conference, Exploring new frontiers ,San Diego, USA, pp. 542-549, 2002.