The cell-DEVS formalism as a method for activity tracking in spatial modelling and simulation

Gabriel A. Wainer

Department of Systems and Computer Engineering,
Carleton University,
4456 Mackenzie Building,
1125 Colonel By Drive, Ottawa,
ON, K1S 5B6, Canada
Email: gwainer@sce.carleton.ca

Abstract: The spreading of wildfires in forests is a complex natural phenomenon that depends on many different variables (such as the fuel, the geography of the area, the weather, etc.). We discuss different methods based on DEVS and cell-DEVS, which can be used to speed up the simulation and simplify the modelling of this kind of spatial system. We propose using a combination of quantised DEVS and dead reckoning to vary the length of the time steps taken by each cell. We also discuss different methods to adjust the computation method dynamically according to the level of activity, varying the length of the time taken by each cell. We show how these methods can be used to track the level of activity in the model automatically, without intervention by the modeller, and how this can improve the simulation performance for fire spreading and other environmental models. We show how the same models can be applied to a parallel simulation environment, allowing finding varied results faster than real-time.

Keywords: cell-DEVS; DEVS; cellular models; activity tracking; discrete-event simulation.

Reference to this paper should be made as follows: Wainer, G.A. (2015) 'The cell-DEVS formalism as a method for activity tracking in spatial modelling and simulation', *Int. J. Simulation and Process Modelling*, Vol. 10, No. 1, pp.19–38.

Biographical notes: Gabriel A. Wainer is a Professor at Carleton University. He has authored four books, and over 290 research articles. He is VP Conferences and was VP Publications of SCS (Society for Modeling and Simulation International). He is Special Issues Editor of *SIMULATION*, member of the editorial board of *IJPSM*, *IEEE CiSE*, and *Wireless Networks* (Elsevier). He has received the IBM Eclipse Innovation Award, the SCS Leadership Award, and various Best Paper awards; also, Carleton University's Research Achievement Award (2005, 2014), Carleton University's Mentorship Award, the First Bernard P. Zeigler DEVS Award (2010), SCS Outstanding Professional Award (2011) and SCS Distinguished Professional Achievement Award (2013).

1 Introduction

Modelling and simulation (M&S) has been widely used for studying the behaviour of complex systems in the environmental sciences. In particular, the spreading of forest fires is a complex phenomenon that many have tried to study using M&S software, as in a real life situation, one wants to predict how the fire will spread, and to have mechanisms to study different scenarios. The behaviour of wildfires depends on many different interrelated variables such the weather, the topology of the geographical area, the kind of vegetation (fuel) burning, etc., which makes it very difficult to be predicted. These aspects make it very difficult to build tools to predict the fire behaviour in real time (and preferably, faster than real-time which would enable fire experts to plan heuristics to control the spreading quickly and safely).

Many authors in this research area have focused on grid-based methods (in which one divides the physical area of interest into cells, with each cell exhibiting the same behaviour of the others) including Berjak (2002), Balbi et al. (1999), Vasconcelos et al. (1995), Rothermel (1972), Barros and Ball (1998), Johnston et al. (2008) and Wang and Wainer (2014). Most of these cellular modelling methods discretise space and time; and the model rules are computed at regular intervals. Using these discrete-time methods, the execution performance can be poor (especially for large models) as all the cells are computed on every timestep (which is not necessary, as we are only interested in the areas of fire activity). In order to deal with these problems, some authors proposed simplifying the complexity of the equations, using cellular automata (CA) as a mechanism for defining simple rules for the model's definition (Gutowitz, 1995). As discussed in Section 3 and in Bonaventura et al. (2013), this has a cost in the precision of the simulation.

Instead, the research presented in this article focuses on a set of new techniques that can be applied to building

advanced fire spreading and other environmental models. The methods are based on modelling the spreading phenomenon using discrete-event cellular models, and in automating the detection of the activity in the area. These ideas, which appeared first in Wainer and Zeigler (2000), are based on a combination of cellular models, discrete events systems specifications (DEVS) (Zeigler et al., 2000), and automated activity detection (Wainer and Zeigler, 2000; Muzy et al., 2002; Wainer, 2004; Bolduc and Vangheluwe, 2003; Muzy et al., 2005) Although these methods are based on DEVS and cell-DEVS formalisms (Wainer, 2009; Van Schyndel et al., 2014), which originally focused on discrete-event models, they have been recently extended for M&S of continuous systems (Bonaventura et al., 2013; Cellier and Kofman, 2006; Zeigler, 2005; Kofman, 200; Nutaro, 2003; Kofman and Junco, 2001; Giambiasi et al., 2000). Most of these techniques are based on the concept of quantised-DEVS systems (Q-DEVS), which represent continuous signals as a discrete-event approximation, represented by the crossing of an equal spaced set of boundaries (Zeigler, 1998). These research efforts showed that the discrete event methods in general (and DEVS in particular), present several advantages for M&S of continuous and hybrid systems:

- computational time reduction: for a given accuracy, the number of calculations can decrease
- hierarchical and modular modelling, which enhances the modelling activities
- seamless integration with models defined with other modelling techniques
- simulation of discrete time models: they can be seen as particular cases of discrete event methods
- generality: the discrete event paradigm provides a uniform theory for the M&S of systems with both continuous and discrete components.

The research presented here deals with different problems in currently used methods. One of the main contributions is that the models can use a simple set of equations, which can be modified by non-experts in the field as they learn about the phenomenon under study (providing *evolvability,* i.e., the ease for modifications and model evolution). The article also explores how these methods can decrease the number of messages used in the simulation (and hence the execution time, particularly if the models are executed in distributed memory computing devices, where the communication costs can be high). This improvement in performance has a cost of increased error in the simulation that can degrade the quality of the results obtained. The methods presented are based on techniques with bounded error and convergence, and we introduce varied experimental examples showing how the error can be limited while gaining in performance.

Another issue to be discussed is based on the traditional way of collecting experimental data to derive the equations that determine the temperature of a cell. As all existing models use equations based on the time advance at a specific time during the burning phase, many of the current implementations are not efficient, and they propose simple fixes that try to solve this problem in the best possible way. Instead, the methods in this paper require a shift in the experimental phase, as we need to find the equations to determine the time the cell will pass a boundary (instead of computing the values of the equations as a function of time). This value, determined by the quantum of a Q-DEVS model, should be used as a function of the current temperature. We will discuss how to combine these methods with dead reckoning algorithms (Lin, 1995), which contrasts with the traditional method of using an equation (fit from experimental data) that determines the temperature of a cell as a function of time.

2 Background

As discussed in the Introduction, the spread of fire depends on many different variables such as the material fuel, slope of the terrain, geographical information of the area, weather, etc. Many of the simulation-based methods used to study this phenomenon are based on extensions to the partial differential equations formalism, but new formalisms have been defined. In this section we introduce some of these formalisms, and we then focus on that have been employed for modelling and simulating forest fires.

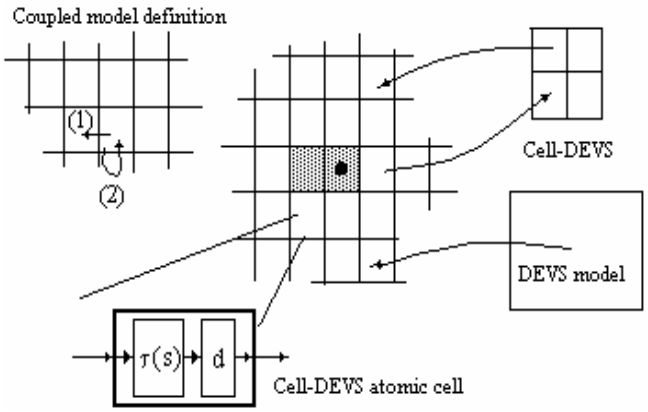
2.1 Formalisms for forest fire M&S

In the last 20 years, numerous authors used cellular computing methods; in particular, CA (Gutowitz, 1995) for modelling and simulating forest fires. CA are n-dimensional infinite lattices whose elements hold a state variable and a simple computing apparatus (Gutowitz, 1995). These local computing functions run synchronously and in parallel, using the present cell and neighbours. CA have been used in the environmental sciences (Dzwinel, 2004; Bandini and Pavesi, 2002; Bianchini et al., 1999; Inghe, 1989) and, in particular, in fire spreading M&S (Berjak, 2002; Balbi et al., 1999; Vasconcelos et al., 1995; Rothermel, 1972; Barros and Ball, 1998; Johnston et al., 2008).

The synchronous evolution of CA poses constraints in the precision of the simulation, reducing performance. Likewise, many of the physical systems in the environmental sciences are asynchronous in nature, and their implementation using synchronous algorithms is not natural. Furthermore, the discrete time used by CA makes it difficult to handle time-triggered activity in each of the cells or changes in the timing of the cells. Instead, cell-DEVS (Wainer, 2009; Van Schyndel et al., 2014), an extension to DEVS (Zeigler et al., 2000), focuses on solving these problems. Using cell-DEVS, a cellular model is described as a discrete event cell space in which explicit delays can be

used to model the cell's timing properties accurately (a sketch of cell-DEVS is presented in Figure 1).

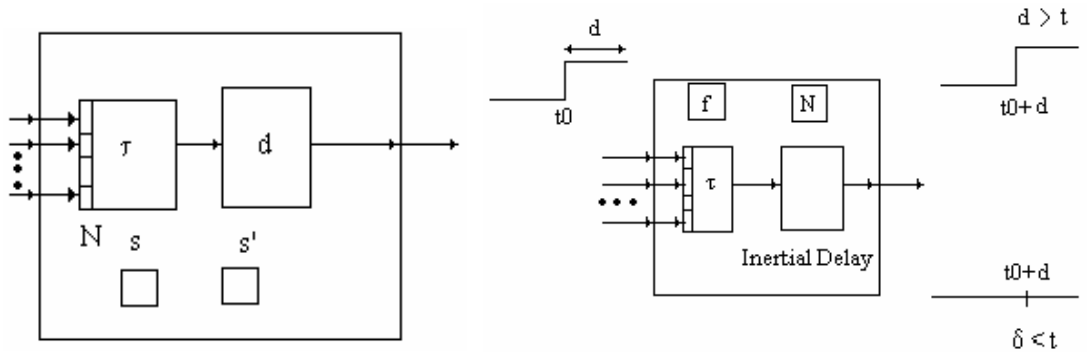
Figure 1 Informal definition of a cell-DEVS model



Each cell of a cell-DEVS model is a DEVS atomic model, and a procedure for coupling cells is defined based on the cell's neighbourhood relationship. Each cell uses a local computing function (τ), and explicit timing delay constructions (d). As the delays use a continuous time base, they allow accurate timing representation, providing an elegant mechanism for dealing with timing behaviour. The hierarchical nature of DEVS also permits the integration of these cellular models with others defined using different formalisms, resulting in enhanced facilities for modelling of complex systems [a detailed definition of DEVS and cell-DEVS specifications can be found in Wainer (2009) and Van Schyndel et al. (2014)]. *Inertial* and *transport* delays allow the definition of complex behaviours for each cell, improving the definition for each of the submodels. Transport delays have anticipatory semantics, that is, every output event is delayed. Inertial delays allows to represent more complex temporal behaviour because they have preemptive semantics and an event scheduled for a future time will not be necessarily executed.

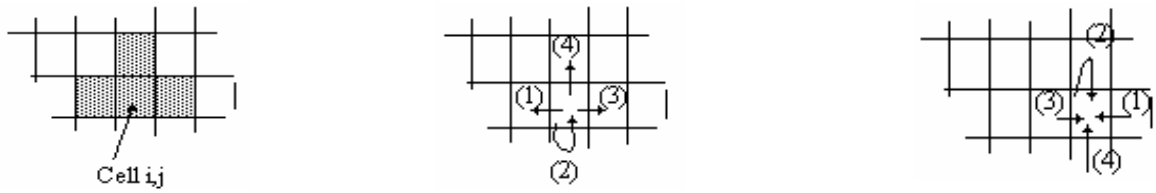
Figure 2 depicts informally the basic contents for an atomic cell. Upon the occurrence of an external event, the local function τ is executed, consuming the inputs N. As the influences must be activated only when the influencing cell changes (Zeigler et al., 2000), the result of the local computing function will be transmitted only when the state changes ($s \neq s'$). In that case, the state change is transmitted after a delay of d time units. A cell will be active while external events are received or internal events are scheduled. The cell passivates only when there are no further scheduled events to be transmitted.

Figure 2 Informal description of an atomic cell, (a) transport delays (b) inertial delays



(a) (b)

Figure 3 (a) A cell, its neighbourhood and the neighbour's list (b) Connecting the output ports of cell I, j (using the neighbourhood list) (c) Connecting input ports of cell I, j (using the inverse neighbourhood list)



Neighbourhood list:
{(0, –1), (0, 0), (0, 1), (–1, 0)}
Inverse Neighbourhood list:
{(0, 1), (0, 0), (0, –1), (1, 0)}
Note: –1: left, up;
1: right, down

(a)

$P_{ij}^{Y_1} \rightarrow P_{i,j-1}^{X_1}$	(1)	$P_{ij}^{Y_1} \leftarrow P_{i,j+1}^{X_1}$	(1)
$P_{ij}^{Y_2} \rightarrow P_{ij}^{X_2}$	(2)	$P_{ij}^{Y_2} \leftarrow P_{ij}^{X_2}$	(2)
$P_{ij}^{Y_3} \rightarrow P_{i,j+1}^{X_3}$	(3)	$P_{ij}^{Y_3} \leftarrow P_{i,j-1}^{X_3}$	(3)
$P_{ij}^{Y_4} \rightarrow P_{i-1,j}^{X_4}$	(4)	$P_{ij}^{Y_4} \leftarrow P_{i+1,j}^{X_4}$	(4)

(b) (c)

A sketch of this procedure can be seen in Figure 3. Figure 3(a) shows the neighbourhood of cell (*i*, *j*) and its representation using the neighbour's list. Figure 3(b) shows how the first output port of cell (*i*, *j*) is connected with the first input port of the first neighbour in the list; the second port with the second neighbour, etc. Instead, for the input ports, the connection is done through the inverse neighbourhood list. For each pair (*i*, *j*) in the neighbourhood, the pair (*–i*, *–j*) must be included in this list.

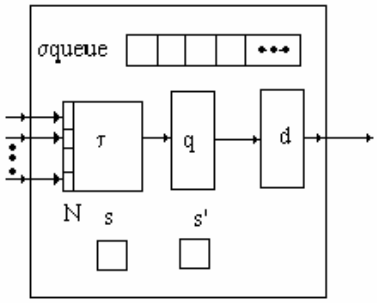
Finally, two extra sets are needed. *Xlist* is a list of cell's positions where the model's external events are received. *Ylist* is a list of cell's positions whose outputs will be collected to be sent to other models in the hierarchy. The values of these cells will be considered the inputs and outputs of the complete cell space.

DEVS and cell-DEVS simulators usually evolve by means of event messages. The overhead produced by these intermodule interactions could be high, and in that case, the computing time employed in the synchronisation of the active cells can overrule the performance improvements of the asynchronous algorithms. There is always a break-even point where discrete-event cellular models have worse performance than their discrete-time versions (therefore, a modeller must decide on the right method to use). From now on, we assume that a discrete-event simulation provides faster results than the corresponding discrete-time CA, or that the modeller is interested in other advantages provided by DEVS, such as model interoperability and integration (and we focus on techniques to improve the performance of these simulations even further). In Wainer and Giambiasi (2001), we showed varied results comparing the performance of models using discrete-time and discrete-event cell spaces. In general, a discrete time model has a constant delay for each generation (because all cells are scanned, and always in the same order). Instead, the discrete event transition increases linearly depending on the number of active cells (due to the time spent handling the event list). The performance of the traffic model improves in several orders of magnitude depending on the complexity of the models used (which included traditional models like the Life game, and complex ones based on traffic simulation).

Although DEVS was defined as a discrete-event M&S methodology, it has been recently extended to include continuous and hybrid systems. Most of the techniques are based on quantised systems (Q-DEVS), whose main idea is to represent continuous signals by the crossing of an equal spaced set of boundaries (called the *quantum*). This operation reduces substantially the frequency of message updates, while potentially incurring into error (Zeigler, 1998). As seen in the figure, when using Q-DEVS, the outputs are only transmitted when its difference with the previous value is larger than a threshold. A continuous signal is thus represented by the crossings of an equal spaced set of boundaries, and by means of a *quantiser*, an artefact that checks for the boundary crossings. This approach requires a fundamental shift in thinking about the system as a whole: instead of determining what value will a dependent variable have at a given time (its state), we must determine at what time a dependent variable will enter a given state. When applying the *quantised state systems* (*QSS*) method (Cellier and Kofman, 2006; Kofman and Junco, 2001), the continuous or hybrid signals are represented using quantisation and hysteresis. In Cellier and Kofman (2006) and Kofman and Junco (2001), it was proven that, when the hysteresis width is set equal to the quantum size, we obtain the smallest possible error. This means that if a value changes its direction with respect to the last threshold value, the next value will have to change two regions to be transmitted. The idea of adding hysteresis is to change the quantum value to the double its size when there are direction changes. Therefore, oscillations can only be large, and cannot occur instantaneously. This provides strong stability, convergence and error bounded properties, as the number of updates is limited by the hysteresis function, and it reduces the number of computations.

In quantised cell-DEVS (Wainer and Zeigler, 2000), each cell is equipped with a quantiser, and the cell's state will be only informed to the neighbouring cells if it crosses the boundary defined by the quantum size. This is shown in Figure 4. The idea is that every cell includes a quantiser q, the value produced by the local computing function τ is quantised, and this value is then compared to the quantum threshold. If the boundary was reached, an output is provided. This output is delayed d time units using transport or inertial delays. Instead, if the threshold was not reached, the change is not sent to other models. A detailed discussion on how to choose the quantum size for QSS simulations can be found in Cellier and Kofman (2006).

Figure 4 Quantised cell-DEVS atomic cell



The performance of these models can be improved by adding dynamic quantisation, which was defined in Wainer and Zeigler (2000). Using this method, one can reduce the simulation error by improving the precision of the local computations. The numerous experiments carried out in Wainer and Zeigler (2000). showed that, using this method, the error in the simulation is reduced when the cells analysed were far from the more active ones. Based on these results, two heuristics for dynamic quantum adjustments were defined:

a Reduce the quantum of the most inactive cells to improve the precision of the inactive cells. Using a quantiser, a very active cell can appear as quiescent. Therefore, if the quantum is reduced, the error

introduced can be improved. In addition, the quantum is increased for the most active cells, improving the overall execution time.

b Increase the quantum of the most inactive cells to make them to passivate faster. These cells are quickly eliminated from the simulation. Likewise, the most active cells will have higher quanta and smaller error.

This was the first attempt in automating the detection of activity in cellular models, and, as we will show in the following sections, these strategies can help in improving the simulation speed, introducing small error. The use of quantised DEVS deactivates the cells in fewer simulation steps, and the dynamic adjustment improves these results further (other conditions, as the shape and size of the neighbourhood and the dimension of the cell space are also a major influence in performance, and the modellers should consider these carefully).

The tests introduced in the rest of this paper were carried out using the CD++ toolkit (Van Schyndel et al., 2014; Wainer and Liu, 2009; Wainer, 2002), which allows implementing DEVS and cell-DEVS models. CD++ has been used in numerous areas, including urban traffic, environmental science, biological systems, chemistry, etc. (Wang and Wainer, 2014; Bonaventura et al., 2013; Van Schyndel et al., 2014; Liu and Wainer, 2007; Wainer and Davidson, 2007). CD++ atomic models can be programmed and incorporated into a basic class hierarchy programmed in C++, while coupled and cell-DEVS models are defined using built-in languages.

2.2 Forest fire Simulation and DEVS

In this section, we briefly discuss the related work that has been done recently in the area of forest fires simulation, in particular those using DEVS. In general, these models compute the temperature of a cell at discrete timesteps, usually as an averaging function of its own temperature and that of its neighbours. Once ignited, the cell's temperature increases to a peak and then falls back down, modelling the exhaustion of fuel in the cell.

One of the most popular models in this field is due to Rothermel (1972). Based on the environmental and vegetation conditions, this model computes the spread ratio (i.e., the distance and direction the fire moves in a minute) and the intensity of the fire. Three parameter groups determine the fire spread ratio:

a vegetation type (caloric content, mineral content and density)

b fuel properties (the vegetation is classified according to its size and type)

c environmental parameters (wind speed, humidity and field slope).

The Northern Forest Fire Laboratory (NFFL) model classifies the vegetation in 13 groups, representing the majority of existing forest types in this region. When Rothermel's rules are applied to a fuel model using given environmental parameters (the speed and direction of the wind, the terrain topology and the dimensions of the cellular space) it can determine the spread ratio in every direction. Different authors introduced DEVS models based on Rothermel's rules (i.e., Vasconcelos et al., 1995; Barros and Ball, 1998), and we introduced a cell-DEVS version of Rothermel's model in Wang and Wainer (2014) and Wainer and Castro (2010) and an extended version using the FireLib Library (Bevins, 2011) in Liu and Wainer (2007, 2010a).

The DELTA environment (Barros and Mendes, 1997) deals with wildfire spreading M&S by including dynamic structure changes. The idea is to make easy the change of parameters dynamically, even in the middle of the simulation. This particular model was tested and compared against many land types, fuel types, topographies and fire spreading information. The principal benefit of using DS-DEVS was the improved use of resources when running the simulation, as the formalism only runs over active cells, and the inactive ones are kept passive for any given time.

Another interesting example of the use of DEVS for fire spreading (that we will use to illustrate the proposed techniques in the rest of the paper) was presented in Muzy et al. (2005). The model is based on experimental results obtained using a 1 m^2 testbed, using earth and pine needles as fuel, and no wind or slope (Balbi et al., 1999). The result of the experiments produced a one-dimensional semi-empirical model in which the temperature of each cell is represented by a PDE. In this PDE, the energy emitted by the cell was considered proportional to the difference between the temperature of a cell and the ambient temperature, as represented by the following equations:

$$\frac{\partial T}{\partial t} = -k\left(T - T_a\right) + K\Delta T - Q\frac{\partial \sigma_v}{\partial t} \quad \text{in the domain} \tag{1a}$$

$$\frac{\partial \sigma_v}{\partial t} = 0 \text{ for an inert cell} \tag{1b}$$

$$\frac{\partial \sigma_v}{\partial t} = -\alpha\sigma_v \text{ for a burning cell} \tag{1c}$$

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

$$T(x, y, 0) = T_a \text{ for the non burning cells at } t = 0 \tag{1e}$$

$$T(x, y, 0) = T_{ig} \text{ for the burning cells at } t = 0 \tag{1f}$$

Here, T_a is the ambient temperature, T_{ig} is the ignition temperature, t_{ig}(s) is the ignition time, T (K) is the temperature, K(m^2/s) is the thermal diffusion constant, α (1/s) 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). Combustion occurs above the threshold temperature T_{ig}; above this boundary, the fuel mass decreases exponentially, and the quantity of heat generated by the combustion reaction per unit fuel mass is constant.

The model was originally run using two numerical methods to discretise the model (finite elements and finite

differences) to approximate the previous equations. The authors used an approximate algebraic equation. Figure 5(a) shows the temperature curve of a burning cell, derived from the equations above.

Although the simulation was efficient, it did not support program evolvability. To improve the model, a modification of the original model (Santoni and Balbi, 1997) used CA with continuous state variables (the cells' temperatures). This CA was simple to define and efficient to run, but it did not provide detailed system behaviour, as each cell only includes a limited number of states (and advanced behaviour in the model would require a major reprogramming of the CA). In all cases, the authors obtained the best performance by activating only the cells neighbouring the front flame. Nevertheless, this model and its optimisations were too complex to be used and modified by non-computer science specialists. Likewise, the performance was not enough for real time simulation.

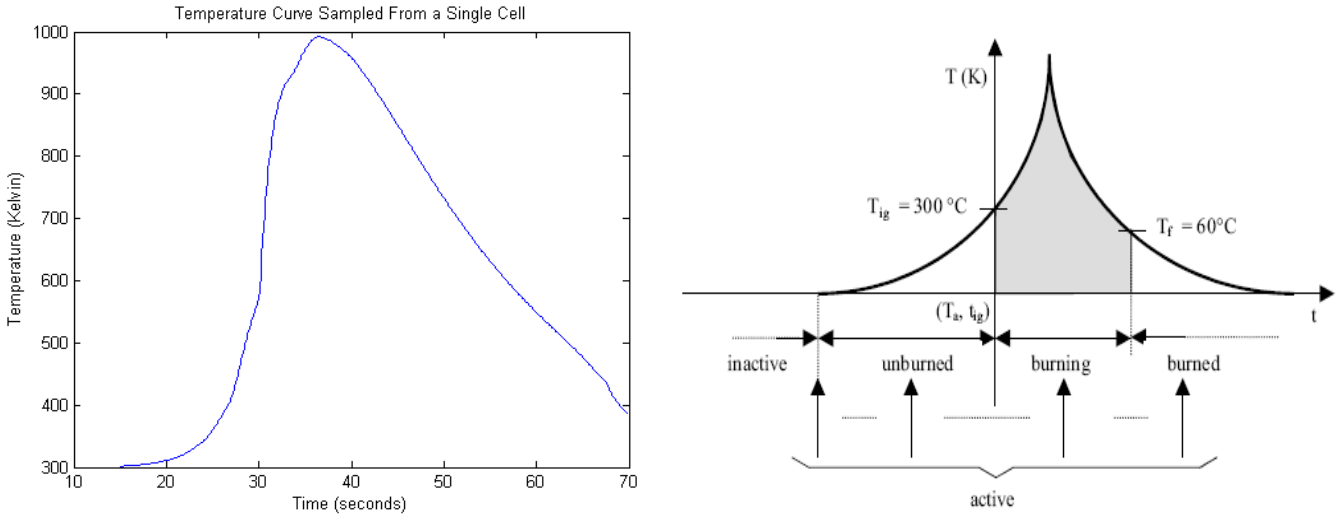
Instead, cell-DEVS is well fitted for solving these problems, as showed in Wang and Wainer (2014), Muzy et al. (2005) and Van Schyndel et al. (2014). This 3D cell-DEVS model is organised in two planes: the first one represents the fire spreading itself (in which each cell calculates its temperature) and the second stores the ignition times for the corresponding cells. Figure 5(b) shows a simplified diagram of the complete curve used for that version of the model (Muzy et al., 2005). The curve is divided in four stages: an *inactive* cell has very low temperature and no neighbours with a temperature higher than T_a. The *unburned* cells have low temperature, which is calculated as the weighted average of the neighbourhood. The *burning* cells have reached the ignition temperature T_{ig} (573K = 300°C), making the fuel burn; the cell's temperature increases until the peak, and when the fuel is consumed, it falls back. In this phase, the temperature is again the weighted average of the temperatures around it, but adding an exponential that describes the temperature behaviour. The fourth stage is *burned* (the cell has exhausted its fuel, the temperature gets lower than 333K = 60°C because the fuel mass is consumed and it cannot longer reignite). In this phase (and when the cell is *inactive*) the cell's temperature does not change (these passive cells will respond to any temperature changes in their neighbourhood, and they could ignite).

Figure 6 shows how we can use CD++ to build a cell-DEVS version of the fire spreading model presented in Muzy et al. (2005).

We use two planes to model our fire-spread model. The plane 0 to store the cell temperatures, and plane 1 stores the ignition time needed by the model. Figure 6 starts with the cell-DEVS coupled model definition (including the neighbourhood definition). Then, the *ti* rules show how to store ignition times: if a cell in plane 0 starts to burn, we record the current simulation time in plane 1. To make this happen, we include a clause specifying to identify the layer in which the current cell is located (*cellpos*(2) = *x*).

The macros show the rules corresponding to the temperature calculus: cells can be inactive, unburned, burning and burned. The first rules in the figure correspond to the phase *unburned*. An unburned cell's temperature is lower than 573K. If the cell belongs to the plane 0, and its temperature at the next time step is greater than the current one, the cell will take the value given by the unburned rule. The same occurs if the simulation time is smaller than 20 (transient period) and it is neither burning nor burned. A cell starts burning at 573K and its temperature increases for a while; then it start decreasing as the fuel mass is consumed. When the temperature gets lower than 333K, the cell enters the burned phase. The first rule in Figure 6 applies to unburned cells, whose temperature in the next step will be higher than its current one. The second rule applies to burning cells.

Figure 5 (a) Temperature curve (b) Simplified temperature curve (see online version for colours)



(a) (b)

Source: From Muzy et al. (2005)

Figure 6 Fire spread model specification and model macros

```
[ForestFire]    % Cell-DEVS Coupled Model Definition
dim : (100,100,2)    border : nowrapped                          %Dimension and Borders
neighbours : (-1,0,0) (0,-1,0) (1,0,0) (0,1,0)(0,0,0)(0,0,-1)(0,0,1)    %Neighbours
zone : ti { (0,0,1)..(99,99,1) }            % A Special rule, ti, is used in the 2nd plane
localTransition : FireBehaviour                                % Everywhere else, the

[ti]
     %Postcondition   Delay                    Precondition
rule : { time/100 }    1      {  cellpos(2)=1 AND (0,0,-1)>=573 AND (0,0,0) = 1.0 }
     %If the cell below is burning (precondition), store the current time. Wait 1 time unit.


[FireBehaviour]
------------------------------------------------------------------------------------------------------
% Rule for Unburned Cells
rule: {#unburned} 1 {(0,0,0)<300 AND (#unburned>(0,0,0) OR time<=20)}

% If the cell is unburned (less than 300 K) and temperature is increasing, compute the first eq.
% The unburned macro computes the value according to the equations

------------------------------------------------------------------------------------------------------
% Rule for Unburned Cells
rule: {#burning} 1 {cellpos(2)=0 AND ( ((0,0,0) > #burning AND (0,0,0)>333) OR (#burning> (0,0,0)
       AND (0,0,0)>=573) ) } %Burning

% If the cell is burning (less than 300 K) and temperature is increasing or decreasing, c
% the second eq. The unburned macro computes the value according to the equation

------------------------------------------------------------------------------------------------------

rule : { (0,0,0) } 1 { t }   %Stay Burned or constant

#BeginMacro(unburned)       % Equation used for unburned cells
(0.98689 * (0,0,0) + 0.0031 * ( (0,-1,0) + (0,1,0) + (1,0,0) + (-1,0,0) ) + 0.213 )
#EndMacro

#BeginMacro(burning)     % Equation used for burning cells
(0.98689*(0,0,0)+.0031*((0,-1,0)+(0,1,0)+(1,0,0)+(-1,0,0))+
              2.74*exp(-.19*((time+1)*.01-(0,0,1)))+.213)
#EndMacro
```

The *ti* rules show how we store the ignition times. The condition is if the cell belongs to the plane 1, and the corresponding cell in plane 0 begins to burn, the cell will take the real time value (simulation time multiplied by the time step).

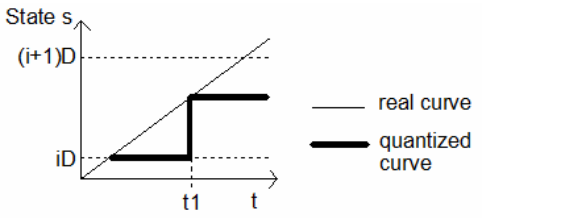
One of the advantages of using cell-DEVS is that all cells in the *inactive* or *burned* phases will remain passive, and thus the calculations will be automatically confined to the fire front. Also, as shown in Wang and Wainer (2014) and Van Schyndel et al. (2014), changing the rules that model the cell's behaviour when new scientific results are available is simple. When running this model using a linear ignition, the prediction of spread rate and the propagation were in agreement with the experimental data. Likewise, the cell-DEVS execution time was better than a previously existing model in terms of CPU and memory usage (a 100×100 cells simulation could not run).

The techniques introduced in this section usually discretise time (using a discrete event simulator on for discrete-time models, which can introduce performance issues). Many of the solutions mix the simulation artefacts with the model (like dynamic structure algorithms to improve memory usage and performance). Likewise, the models are sometimes complex (trying to identify the front flames), and evolvability can be compromised (mostly experts in DEVS and the particular tools can modify these models with ease). In the next sections, we will show how these procedures can be automated by the simulation engine, hiding the optimisations from the users, and improving performance through automated detection of activity.

3 Experiences with dynamic quantisation

As discussed in the previous sections, when using quantised DEVS, a state value will be only informed to its neighbours when it crosses the quantum threshold. This operation potentially incurs into error while improving performance substantially. The inclusion of hysteresis provides stability, convergence and bounded errors. Thus, we propose combining these methods with cell-DEVS and dynamic quantisation. One of the problems with these techniques is the choice of the quantum size: an active cell can appear as quiescent if a quantum covers the activity area (resulting in a larger error) or produce many oscillations if the quantum is too small. Figure 7 shows a case where the value computed by the cell at time *t1* does not reach the threshold $(i + 1)D$.

Figure 7 A cell missing activity



Hence, as discussed in Figure 4, when the quantiser compares this value with the last one computed (iD), the cell is considered quiescent. Unless a neighbour reactivates it, the cell will passivate and the simulation might not evolve. Instead, if the quantum size is reduced, activity will be detected and a smaller error will be obtained. Conversely, if we increase the quantum size in very active cells, the execution times can be improved (introducing some error).

In order to automate these activities, we proposed two different heuristics to adjust the quantum size, based on the signal-to-noise-ratio (SNR). Let q be the base quantum, r the adjustment ratio for the dynamic quantum, and $d(t)$ the quantum value used in time t. If $v = i.D$, $v' = \tau(s)$ (the new computed value), and $q(0) = q$, then for

$$regionChange(v,\ v',\ q)$$
$$= (v = \phi \mid\ q = 0\ \mid (q \neq 0 \wedge [v/q] \neq [v'/q])):$$

- high SNR heuristics:

$$\neg regionChange(v,v',d) \Rightarrow d = q*(1-ratio);$$
$$regionChange(v,v',d) \Rightarrow d = q*(1\ +ratio);$$

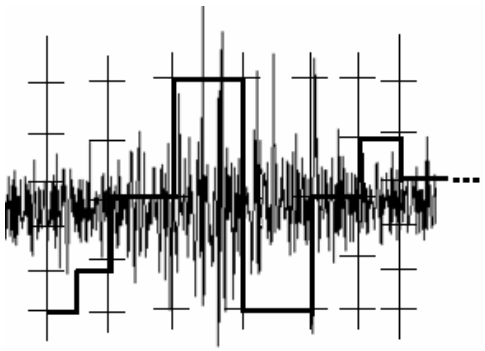
- low SNR heuristics:

$$regionChange(v,v',d) \Rightarrow d = q*(1-ratio);$$
$$\neg regionChange(v,v',d) \Rightarrow d = q*(1\ +ratio).$$

The idea of the high SNR strategy is as follows: if the result of updating the cell's value is below the boundary (like in Figure 7), the quantum size is reduced. Otherwise, the quantum size is increased. When the level of activity in the cell is low, we increase its precision, and the activity on the cell will be now detected, reducing the quantum size allows one to detect small levels of activity). If a cell becomes very active and it crosses many boundaries quickly, we increase the quantum size. This technique assumes that in regions of high activity, the function does not change often, and we can speed up the computation; this will increase the simulation speed with small error added.

The low SNR strategy, instead, focuses on signals with high SNR, as the one in Figure 8. The method increases the quantum size every time a threshold is crossed. By doing that, we can filter noise and extract the relevant events as seen in Figure 8. Otherwise, the quantum size reduces. This strategy will reduce the number of messages involved in the simulation at a cost in higher error. Figure 8 shows a noisy signal. Initially, we use a small quantum size (five regions in total), and the signal is represented by a piecewise constant trajectory with five levels. With this quantum size, and as we cross the boundaries at every step, the quantum size is increased. In the centre of the figure, we only have one region for the whole spectrum of the signal. When no change is detected during a given time, the quantum size is reduced again, as we can see at the end of the figure, where there are five regions and a smaller quantum again.

Figure 8 Low SNR signal and dynamic quantisation



We conducted varied experiments using both strategies in the context of different applications. One of them is a cell-DEVS implementation of the action potential (AP) in the cells of human atria. This was based on a model originally defined by Hodgkin and Huxley (1952), who investigated the behaviour of the heart muscle membrane, and presented the detailed behaviour of the inter-membrane AP. Whereas solving the Hodgkin-Huxley equations using numerical methods is feasible for one cell, the realistic reproduction of the heart tissue (consisting of millions of cells) is unfeasible. Consequently, different authors tried to introduce simpler rules, using CA instead of the Hodgkin-Huxley equations. This has a cost in precision (Bonaventura et al., 2013), as CA use a discrete time base, and because introducing modifications to the basic behaviour of individual cells results in complex code reorganisation (for instance, arrhythmias, modelled by a different AP function for isolated groups of cells, can be hard to code).

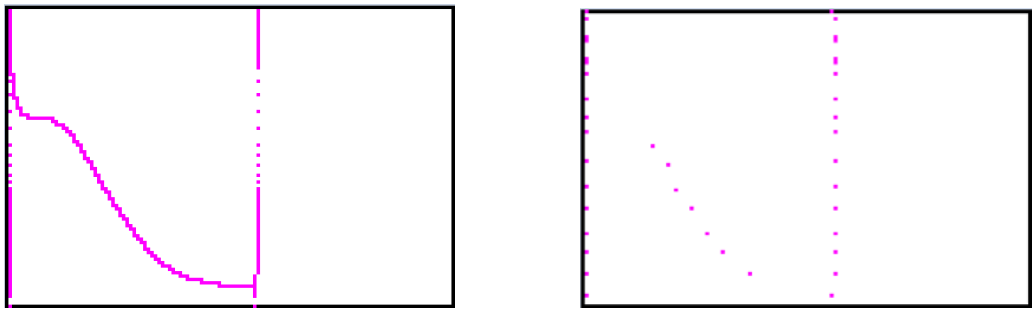
Using cell-DEVS with dynamic quantisation, one can define detailed behaviour for each cell using easy to modify rules and well defined functions for timing, which allows one to introduce new arguments and rules with ease. This method also discretises the model automatically, as the dynamic quantiser detects activity in the model, improving its precision and speed. Figure 9 illustrates this with two examples using different quantum sizes.

Figure 9 shows the execution of the AP function (the X axis represents one AP that takes 50 ms; the Y axis represents the current in the heart tissue membrane, which ranges from –20 mV to 5 mV). The figure shows how Q-DEVS automatically discretises the model while keeping the original rules used to generate the behaviour unchanged. We do not need to define artificially the state values for the regions of interest, as done for CA (in which the excitation, relaxation and refractory periods are usually identified, making the model very efficient to compute, but very imprecise when one needs to study the individual cell

behaviour in detail). The AP rules in the figure are evaluated only if the cell is resting and a positive voltage is detected in the cell's neighbourhood. This activity will trigger the update of the cell state using the Hodgkin-Huxley equations.

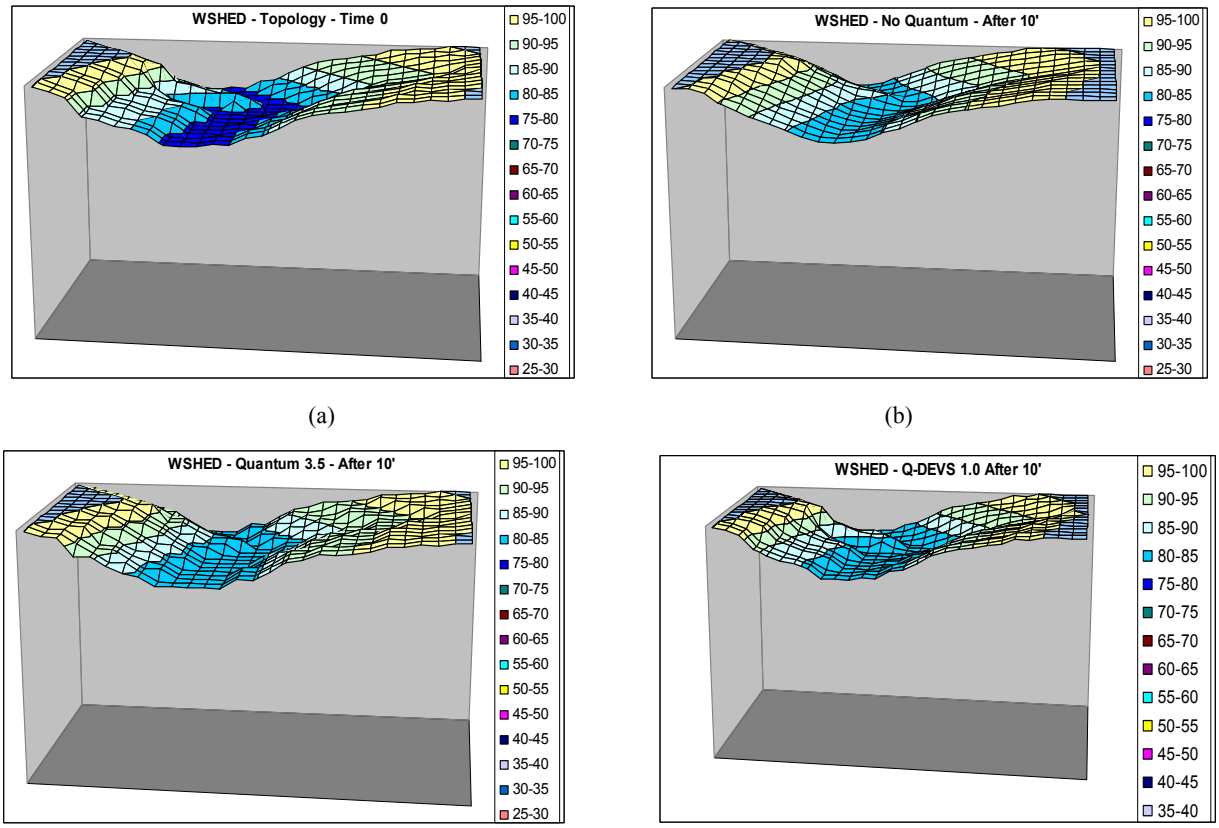
Similarly, Figure 10 shows the results of a watershed formation model presented in Moon et al. (1996). In this case, the rain is absorbed first by the vegetation in the surface of the terrain, and the rest (effective quantity of water) is accumulated at the surface. Depending on the topology of the land, the cells can also receive/send, water from/to the neighbours. Part of the water received is lost due to the filtration over the land and stones. The accumulated water depends on the quantity of effective rainwater, and the quantity of water from the neighbour cells minus the water filtered by the stones and soil. Figure 10(a) shows the initial state, which represents the slope of the terrain before raining (each cell occupies 1 km^2). The remaining figures show the execution results after intense rain (0.0022 mm/s) after ten minutes of simulated time, showing the changes in the surface of (and the accumulation of water) for different simulation heuristics and quanta (the figure represents the surface of the terrain measured in metres).

Figure 9 Sample execution results of the heart tissue model, (a) quantum = 0 (b) quantum = 20 (see online version for colours)



(a) (b)

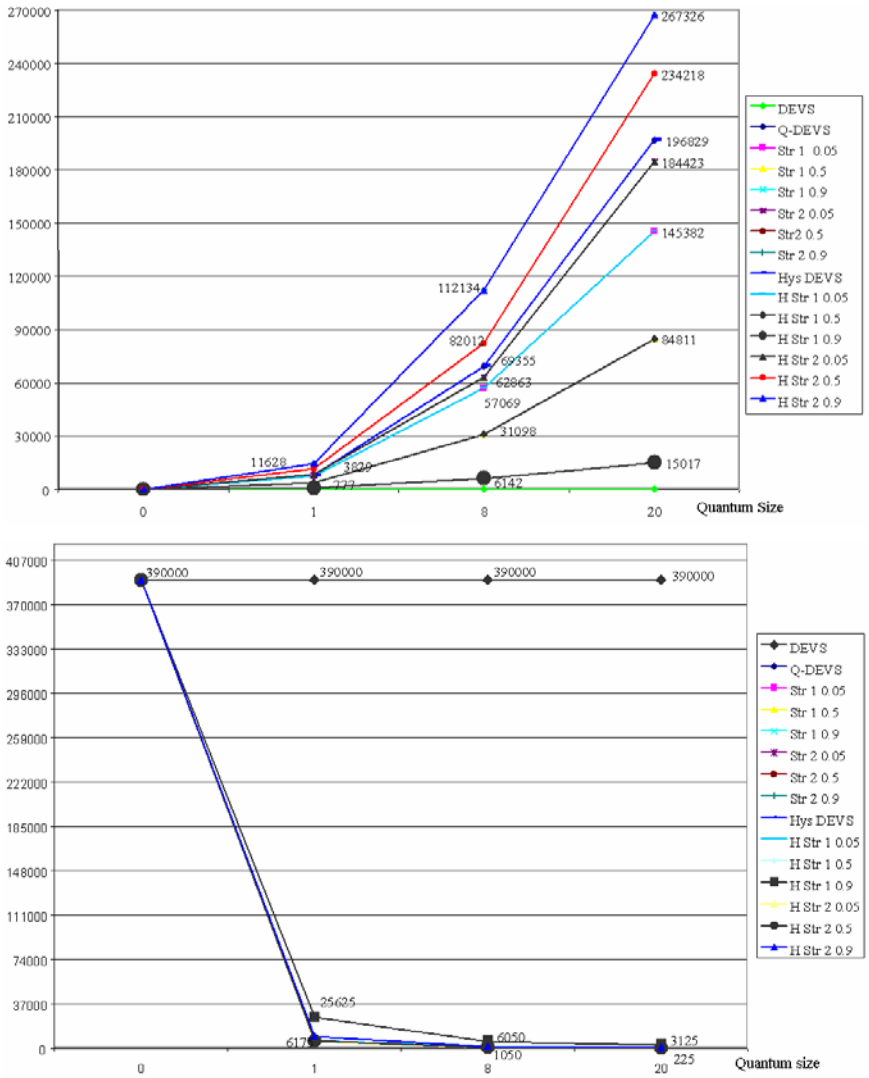
Figure 10 Watershed simulation results, (a) initial watershed state (b) quantum = 0 (c) quantum standard 3.5 (d) dynamic Q-DEVS high SNR strategy 1.0 (see online version for colours)



(a) (b)

(c) (d)

Figure 11 Heart tissue model, cumulative error number obtained; number of messages interchanged (see online version for colours)



In Figure 11, we show some experimental results obtained using the automated dynamic quantisation. We analyse two main metrics: the execution time (represented by the number of messages involved in the simulation), and the amount of error introduced by each strategy. We executed a large number of tests in different categories, including non-quantised cell-DEVS, QSS and Q-DEVS, dynamic quantisation with high SNR and low SNR strategies. Different combinations of the previous categories with different quantum sizes and different update ratios were used (we discuss representative results of the tests; more results can be found in Wainer, 2004). Figure 11 presents the cumulative error and the number of messages for the heart tissue model.

The error was obtained by comparing the values in quantised simulations against those in non-quantised cases. The cumulative error at time t(E_t) was computed as

$$E_t = \sum_{t=0,\, t<\delta} \sum_{i=1..n} \left(\left(s_{i,t} - q_{i,t} \right) / s_{i,t} \right) / n,$$

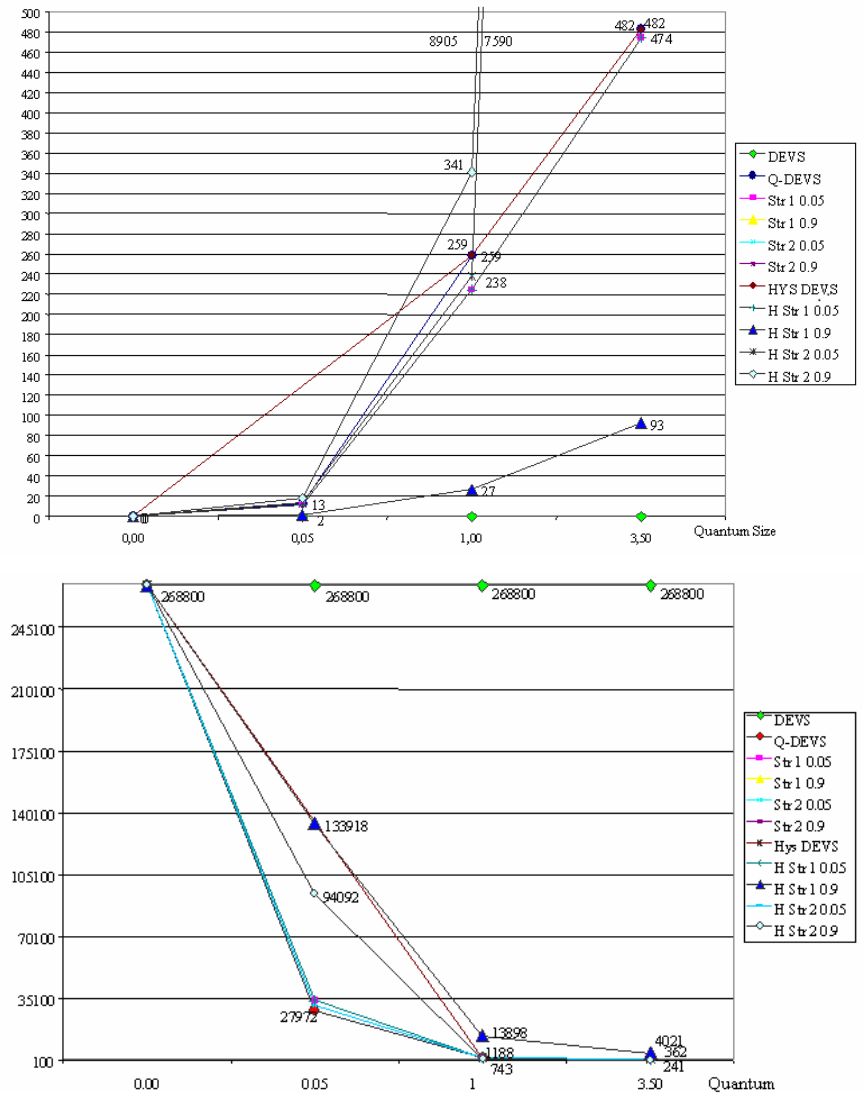
with n the total number of cells, δ the programmed simulation time end, $q_{i,t}$ the value of the quantised value of cell i at time t, and $s_{i,t}$ the value of cell i at time t. In this case, the results obtained with Q-DEVS and QSS were similar, as QSS works better with noisy signals (it applies hysteresis when there are changes in direction and, as seen in Figure 9, there is only one change in direction on each activation of this model). The lowest error was obtained with the high SNR strategy, using an update ratio of 0.9. When we analyse the heart tissue function, we can see it has high SNR. There is an initial spike in the function, which increases the quantum every time the boundary is crossed (thus reducing the number of messages). Simultaneously, the error is constrained (except for very large quantum sizes). As the cell is updated every 100 ms, the quantum reduces very quickly, and it increases quickly again, keeping the error limited. Likewise, all the high SNR results were better than the low SNR strategy and Q-DEVS (the larger the ratio, the better the result). This result was expected for this kind of model, which is a function with high SNR. Instead, the low SNR strategy gets a very large error (worse when the ratio is higher): on each update during the transient period, the quantum size increases, generating a large amount of error (and on the linear

section, the quantum is reduced, gaining little precision at a high cost for the simulation). In this case, the strategy assumes a high SNR (which is not the case for this function), which gets higher precision every time a threshold is crossed. Here, every time the cell is updated, the boundary is not crossed, thus, the quantum size increases (highly reducing the number of messages but increasing the error, which diverges for large quanta). A higher update ratio makes things worse for the low SNR strategy because it is aggressively searching for noise (and there is none in this function).

Figure 12 presents the results of the watershed model using the land topology presented in Figure 10. We could observe a similar pattern than the one obtained for the heart tissue model. Again, there was no difference between Q-DEVS and QSS: each cell uses a linearly increasing function. The lowest error was obtained with the dynamic quantum with high SNR (ratio 0.9), and all the results obtained with the high SNR strategy were better than those obtained with low SNR and standard Q-DEVS, as the water accumulation function has even a lower SNR than the heart tissue AP function. Although the order of the different heuristics is repeated, in this case the total error is smaller than in the heart tissue case, because the cell's function increases slowly, and in a linear fashion.

The figure shows a particular phenomenon: using Q-DEVS with $q = 0.05$ provides better results than the dynamic strategies. Studying the behaviour of each cell, we could see that they increase linearly approximately 0.07 units on each update. Consequently, by increasing the quantum size, we obtain a value larger than 0.07, which reduces again in the next simulation step. The quantum changes oscillate around the boundary, resulting in an increase in the total number of messages. This number increases more when the update ratio is higher, as the quantum varies a larger rate at every step. This does not occur when the quantum size is fixed.

Figure 12 Q-DEVS watershed model output messages; watershed model cumulative error (see online version for colours)



Using a low update ratio improves the number of messages involved while increasing the error. By paying a small cost in the extra execution overhead, we were able to reduce the error involved (up to 75%). The low SNR reduces the number of messages involved in a higher rate than high SNR, but incurring in a higher amount of error. If we consider, for instance, $q = 1$ with high SNR and ratio 0.9, the error introduced is minimum and the number of messages is highly reduced. If we consider $q = 3.5$, the error obtained with high SNR strategy is better than the remaining techniques with a larger quantum, while the number of messages involved is comparable.

In every case, the lowest error was obtained with the high SNR strategy. Updating the dynamic quantum size with higher/lower ratios improved the simulation results even more. Simultaneously, the number of messages was reduced significantly. The dynamic quantisation is adapted to improve the error involved; while making the overall execution time highly reduced. In other experiments, we could also see that the introduction of hysteresis permits to obtain a more controlled behaviour, even for applications with cells executing with a nonlinear pattern.

4 Quantising the fire spread cell-DEVS model

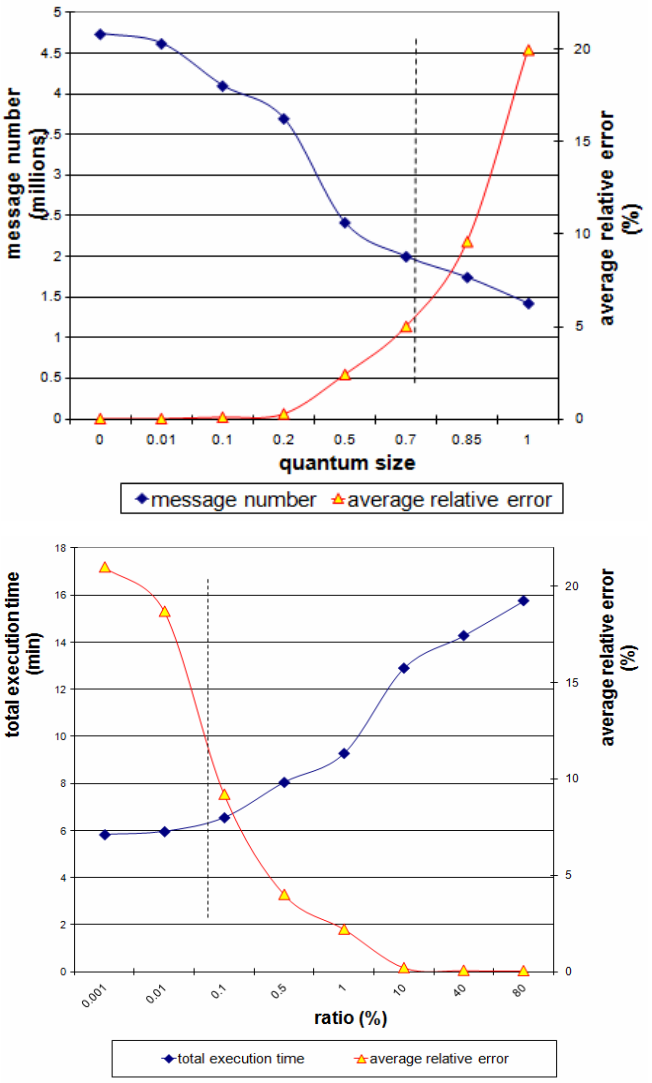
The level of activity is usually measured by seeing how much the cell changes. In this section, we discuss the cell-DEVS quantisation techniques for fire spreading modelling. Figure 13 shows an example for a 20 × 20 propagation domain during 20 s using different quantum sizes. Figure 13(a) shows a reduction in the message number and the execution time, and the error obtained for both cases, which is similar to the results in previous sections. Figure 13(b) shows different ratios applied to $q = 1$: the larger the ratio is, the lower the error (and the longer the calculation time). For a ratio of 0.1%, the error decreases from 21% to 9.2%, adding a small execution time overhead (from 5:50 min to 6:33 min). The dynamic quantisation allowed us to optimise the quantum size of each cell according to cell's phase. Hence, the error and the execution time have been reduced. Nevertheless, error still does not converge for high quanta.

As discussed in Figure 7, one of the main problems for this model (and many other similar ones) is that time advances from time t to $t + h$, and quantisation can only improve the results up to a certain extent. We could also see that if the quantum is too large and the energy brought by the neighbouring cells is not enough, the temperature of the cell cannot reach the boundary, and it remains between two states (hence, the cell is considered inactive). These issues will be discussed in the following sections, where we propose new methods for solving this problem.

4.1 Quantised function definition

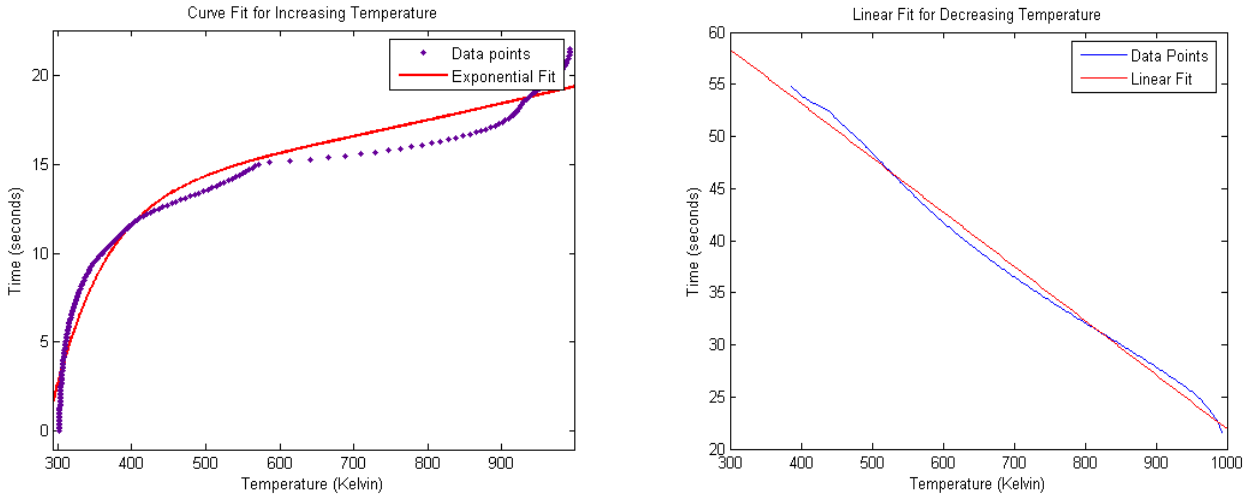
As discussed in Section 3, each cell will send outputs to its neighbours if its temperature has exceeded the next boundary. As previously shown in Figure 7, when the quantiser compares the new computed value with the last one computed, *iD*, the cell is considered quiescent, and unless a neighbour reactivates, the cell will passivate forever. This prevents the model from evolving, as all cells could be considered passive and the simulation will finish.

Figure 13 Message and error comparison; error and execution time comparison for $q = 1$ (see online version for colours)



The main reason for this problem is that most of these models use a discrete timestep whose results are then quantised. The reason for this is that most experimental studies are currently carried out using a time-based approach; most equations are derived as a function of time. Instead, we need to calculate time based on temperature, rather than temperature as a function of time, defining the quantised functions as a function of the quantum boundaries. Figure 14 shows (an approximation to) the inverse of the temperature curve for a typical cell. Given such a function $f(T)$, we can calculate the amount of time it will take to reach the next quantum level as the difference $f(T_2) - f(T_1)$. This saves unnecessary calculations, as cells will only become active when a significant change in temperature occurs.

Figure 14 Inverted temperature function, (a) increasing (b) decreasing (see online version for colours)



(a) (b)

As seen in Figure 5, the temperature function fails the horizontal line test, and therefore is not directly invertible. Hence, we divided it into increasing and decreasing components, giving us two invertible functions. Figure 14(a) shows a fitted function for the increasing temperature portion of the curve that uses a sum of two exponential functions:

$$f(T) = 11.56 * e^{0.0005187*T} - 784.7 * e^{-0.01423*T} \quad (2)$$

where T is the temperature in K. This can be used for cells in the *burning up* phase. Similarly, the decreasing portion (or *cooling* phase) is fit with the linear function:

$$f(T) = 0.052 * T \quad (3)$$

(we also found higher order functions that fit the functions with greater accuracy if needed). Using these functions, most of the problems discussed in previous sections are not an issue anymore: these functions can be used for updating the cell's values directly, and they will not compute intermediate values (and will only activate the cell after the boundary has been reached). The function gives us the time corresponding to a specific temperature, thus we can apply dead reckoning techniques (discussed in the next section) and use the function to calculate what the time will be when the cell's temperature crosses the next quantum. By doing this, the cells remain inactive until they reach the next quantum boundary. At that time, the cell will calculate the new state and the next time at which they will cross the next quantum boundary (and then it passivates until that time arrives).

4.2 Dead reckoning

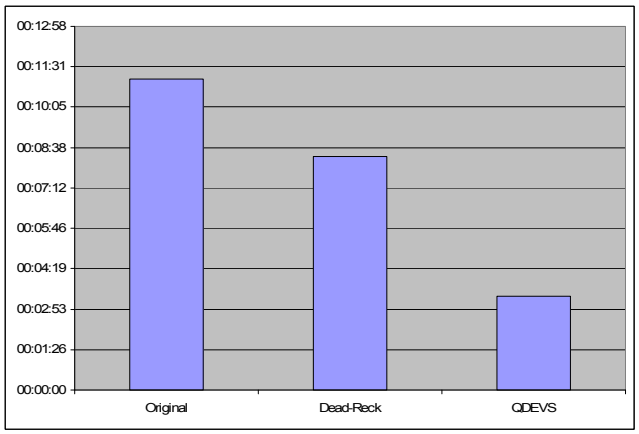
Although using dynamic Q-DEVS is straightforward, combining dynamic quantisation with the boundary detection (as discussed in the previous section) and dead reckoning can improve the results considerably. To do this, one needs to track the current slope of the function and extrapolate from there to predict the next change. Accurate calculations of the current temperature could be made after either every jump (or a state variable could be kept to limit it to every N jumps, improving the performance further). This saves execution time, since the cells only activate on the significant event of crossing a threshold of interest. Collecting experimental data using this method is also more efficient: instead of sampling every cell every 1 ms (or similar sample time), one only needs to record the times of boundary crossings. This would potentially save data storage and make better use of network bandwidth in the testbed. This sort of technique has already been applied to distributed simulation as shown in Lin (1995), Cai et al. (1999) and Lin and Schab (1994), and in the case of cell-DEVS, it can be implemented at the delay function, see Figure 15.

This cell-DEVS model uses two planes: the temperatures are computed on the first plane (which is detected using the *cellpos*() function, which returns in which plane is the current cell), and the ignition times are in the second plane. The cells in the *Burning* phase have not yet reached their peak temperature. These cells will calculate the delay after which they should increment their temperature [according to the *burning* function defined in equation (3) in Section 4.1]. This value now depends on the quantum size q. As we can see, the cell value is transmitted only after the boundary is reached, and then the cell passivates until the next time arrives. Cells in the *cooling* phase are still burning, but have reached their peak temperature, which is falling from here on in. These cells will calculate (according to the burning down function) the delay after which they should decrement their temperature, and then they become passive for that amount of time.

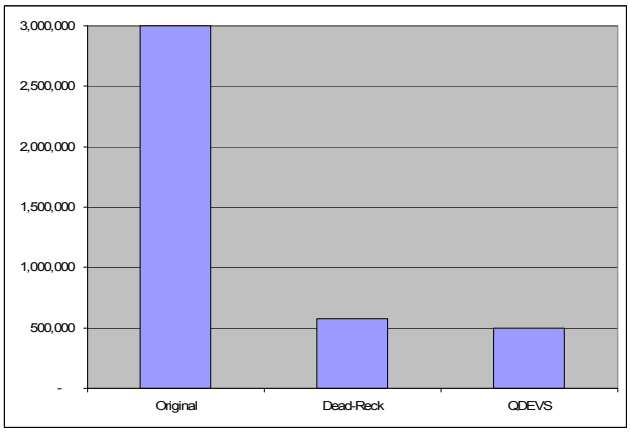
Figure 15 Dead-reckoning implementation

```
[FireBehaviour]
------------------------------------------------------------------------------------------------
% Rule for Unburned Cells
rule : {
            % Rule Postcondition: compute the cell's value: will now become a Burning cell
        #macro(unburned) + #macro(q) }

            % Delay for the Cell's output: computing the speed rate according to the formulas
       {  round (((11.56 * exp(0.0005187*((0,0,0)+#macro(q))) - 784.7
         *exp(-0.01423 *((0,0,0)+#macro(q))) ) - (11.56* exp(0.0005187*(0,0,0) ) - 784.7
         * exp(-0.01423 * (0,0,0) ) ) ) * 100)}

            % Rule Precondition: check that it is not a burning or burned cell and there is fire
       { cellpos(2)=0 AND #macro(unburned)>(0,0,0) AND (0,0,0)<573 AND (0,0,0)!=209 }

------------------------------------------------------------------------------------------------
% Rule for Burning Cells
rule : {
            % Rule Postcondition: compute the cell's value: compute the next burning value
        #macro(burning) + #macro(q) }

        % Delay: computing speed rate according to formulas for burning cells
        {round (((11.56*exp(0.0005187*((0,0,0)+#macro(q))) - 784.7*
          exp(-0.01423*((0,0,0)+#macro(q))) ) - (11.56* exp( 0.0005187*(0,0,0) ) - 784.7 *
            exp(-0.01423 * (0,0,0) ) ) ) * 100)}

            % Rule Precondition: check that temperature is increasing; or decreasing and not burned
        { cellpos(2)=0 AND ( ((0,0,0)>#macro(burning) AND (0,0,0)>333)
           OR (#macro(burning)>(0,0,0) AND (0,0,0)>=573) )AND (0,0,0) != 209 }

------------------------------------------------------------------------------------------------
% Rule for Burned Celles: make the cell have a fixed value of 209 degrees as postcondition
rule : {209} 100 { cellpos(2)=0 AND (0,0,0)>#macro(burning) AND (0,0,0)<=333 AND (0,0,0)!=209 }

------------------------------------------------------------------------------------------------
% Second plane (cellpos(2) = 1): stores the time when the cell below starts burning.

rule : { time * 0.01 } 1 { cellpos(2) = 1 AND (0,0,-1) >= 573 AND (0,0,0) = 1.0 }
```

Figure 16 (a) Execution times (b) Number of messages passed (see online version for colours)



(a)



(b)

We ran the fire spread simulation in CD++ using dead reckoning and the original model, in order to compare the results of the two. The initial values used were similar to that in the provided model, representing a line ignition scenario. As seen in Figure 16, Q-DEVS and a combination with dead-reckoning (the third column) reduced the messaging between cells dramatically. The number of messages in the simulation was reduced by more than even when almost all cells in the model are active (gains were even larger when only a few cells were initially activated).

As we had noted earlier, reducing the execution time of the simulation could result in reducing the accuracy of the model and generating a large error. As we can see in Figure 17, the results we obtained using our model are similar to the ones obtained with the original model, which matches the experimental data. The cumulative average weighted error for the simulations was below 2%, following the trend presented in previous sections.

Figure 17 Execution results at 0, 300 and 1,000 time units (see online version for colours)

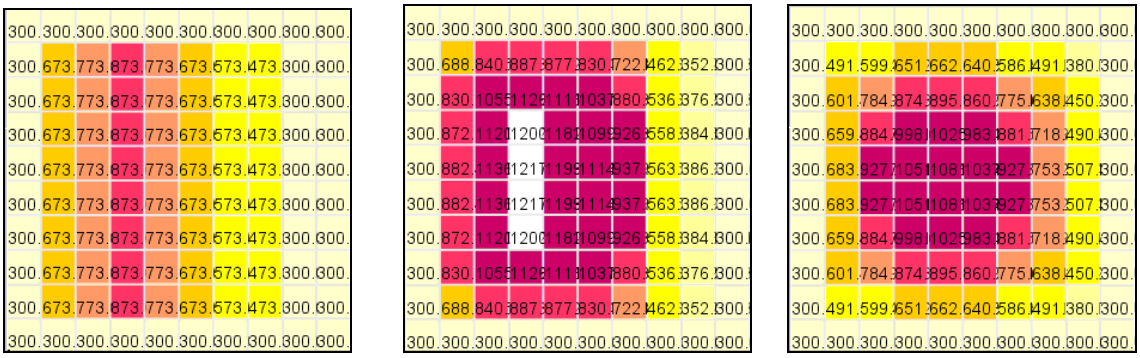


Table 1 Fire1 test results (s)

Size	*1*	*2*	*4*	*6*	*8*	*10*	*12*	*14*
100 × 100	34.03	30.45	*29.35*	30.74	33.24	36.57	39.63	45.48
200 × 200	565.44	437.88	379.47	*373.07*	381.11	393.54	405.42	436.87
300 × 300	2,872.10	2,234.75	1,890.38	1,812.64	1,773.66	1,777.89	*1,765.33*	1,865.92
500 × 500	22,537.10	16,476.80	14,209.30	13,377.10	13,031.00	12,891.80	12,793.90	*12,570.60*

4.3 Parallel processing

An interesting advantage of the methods presented here is that, being applied to the simulation engines, the original models do not need to change. Based on these models, we can achieve higher performance applying parallel discrete event simulation (PDES) techniques. We built different parallel simulation environments, namely, the conservative CD++ (CCD++) and different Optimistic versions (Jafer and Wainer, 2010a, 2010b; Liu and Wainer, 2010b). In all of them, the cell-DEVS model definitions remain unchanged. Therefore, we can apply the same models discussed throughout this paper without any changes to the original specifications, achieving large speedups in the execution results. In this section, we show some results of two of the forest fire propagation discussed in earlier sections, based on Rothermel's definitions. The models, which will be called *Fire1* and *Fire2*, differ in how the spread rates are calculated. *Fire1* uses a predetermined rate at reduced runtime computation cost, while *Fire2* invokes the FireLib library to calculate spread rates dynamically based on a set of parameters such as fuel type, moisture, wind direction and speed. For all the models showed here, we used a partition strategy that evenly divides the cell space into horizontal rectangles.

Table 1 gives the resulting total execution time for *Fire1* of varied sizes on different numbers of nodes. The best execution times in each series are shown in bold. The model was tested using cell spaces of 100 × 100, 200 × 200, 300 × 300, and 500 × 500, on 1 to 26 nodes. As we can see, the conservative parallel DEVS simulator reduces the execution time as the number of nodes increases until it reaches the best execution time (the value in bold). We are able to execute very large models (up to 250,000 cells). The smallest execution time is achieved at 4, 6, 12, and 14 nodes respectively and after that, the execution increases when more nodes are engaged (due to the simulator's overhead). Also, when the number of nodes gets closer to the boundary value, the difference among execution times is not meaningful (for example, for the 200 × 200 model, the execution time decreases by only 1.7% from four to six nodes, and for 500 × 500 cells it only decreases by 1.8% from 12 to 14 nodes). This is because when a model, especially a small one, is partitioned onto more and more nodes, the overhead involved in inter-processor communication and the increasing number of support messages eventually degrades the performance. We need to consider the tradeoffs between the benefits of higher degree of parallelism and the associated overhead needs when choosing different partitioning strategies.

Figure 18 shows the results obtained when applying the simulation engine in the Cell-BE processor for the *Fire2* model (Liu et al., 2010). This model uses a 1,024 × 1,024 cell space (over one million cells) to simulate a wildfire scenario based on predetermined spread rates for 50 virtual hours. The model is evaluated sequentially by the active cells at each virtual time to determine their future states. The figure shows the overall simulation time obtained on the Cell-BE processor (the X axis shows the simulation results obtained when the model was executed in multiple partitions in different cores and different number of cores), and on an Intel E6400. On the Cell-BE, the simulation time was reduced from over three hours (one power processing element; PPE-optimised, which runs in only one power PC CPU) to just 20 minutes with a kernel split on 16 synergistic processing element (SPE). Comparing to the baseline and optimised CD++, the simulation achieved speedups up to 134.34 and 9.74 on CBE and up to 41.23 and 1.92 on the E6400 respectively, running faster than real-time.

Figure 18 Optimistic simulation on parallel Cell-BE architecture (see online version for colours)

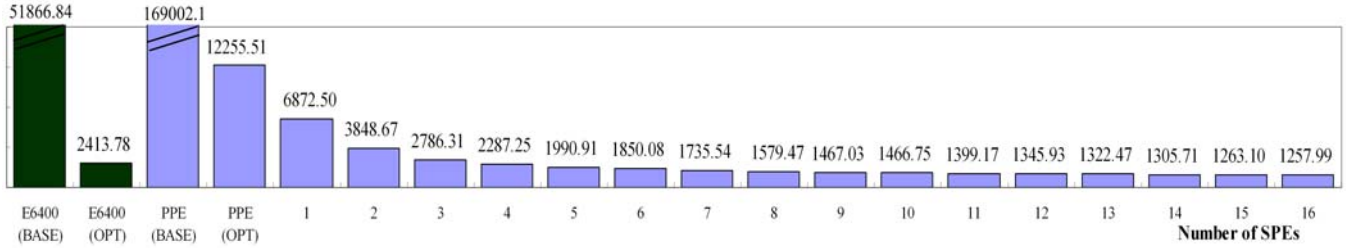
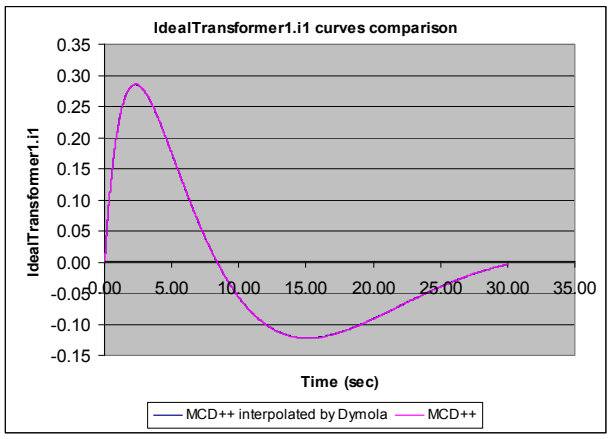
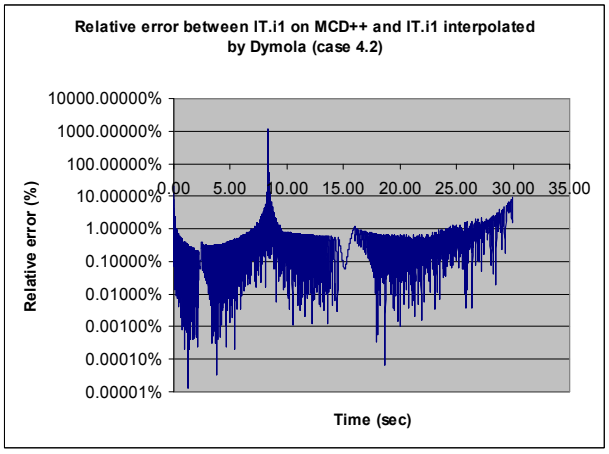


Figure 19 Comparing QSS-based and time-based modelica simulators: trajectories and relative error (see online version for colours)





4.3 *Higher order quantisers*

One of the problems found in the fire models presented in previous sections is derived from the fact that the quantisers do not provide good results for the local maxima/minima as well as values close to zero. For instance, the following figures show the current trajectories (input/output flow) and the error for an ideal transformer plotted based on the results of two modelica compilers: Dymola (Dynasim Laboratories, 2004) and M/CD++ (Wainer and D'Abreu, 2014). Dymola is a commercial toolkit for complex physical systems M&S with full support of the Modelica language. M/CD++ is an open-source Modelica compiler running on CD++ that uses a quantised DEVS model to approximate the functions, while Dymola uses an advanced solver based on the DASSL algorithm.

On the examples in Figure 19, we can see that the approximate solution based on QSS (which uses first order integration methods) has a high error around zero. A higher relative error was obtained for values near to zero on the trajectory. The main reason for this is that using a fixed quantum size (as provided by the quantisation function) over the state trajectory can increase the relative error for smaller values. Likewise, a first order approximation requires a large number of steps to obtain the required accuracy (Kofman et al., 2011).

In order to deal with these problems, we propose to use higher order methods to quantise each cell (which preserve the second, third derivatives, etc.). This approximation gives a piecewise linear output trajectory, and one can transmit the slope change coefficients when they differ from the previous one in more than the quantum size Δq. Similarly, higher order methods can be used. It has been proven that these methods are stable and converge, and they can be represented as cell-DEVS models. In that case, the model will use as many input/output ports as the number of coefficients used by the approximation functions. Then, the cell will transmit the coefficients (one of them on each input/output port), and this is done only when the coefficients differ from each other by more than the quantum size Δq.

Using a higher order approximation requires a smaller number of activations and we need a more precise approximation (which also takes longer to compute). With higher order methods, we transmit more messages on each activation (because now each cell needs to use one port per coefficient) but the number of activations is much lower (Kofman et al., 2011). The lower order method can be computed much faster. Using higher order approximations reduces the error around zero and on local maxima/minima, at a cost of overhead in computing the higher order function. Therefore, in order to improve these results further, we also propose using an adaptive quantisation function on each cell, which will make the quantum vary according to the trajectory evolution, adapting the two heuristics proposed in Section 3 (low and high SNR). The idea is that, instead of adjusting the quantum size, we switch between the different orders methods according to the precision needed. The goal is to apply the two SNR strategies and switch between one method and the other dynamically according to the level of noise in the cell. In this case, the dynamic quantiser will be in charge of

changing between one method and the other according to the level of noise detected: if the number of activations is high, we can switch to a higher order method.

4.4 Input/output activity tracking

Another issue with earlier versions of the models introduced in Sections 2 to 4, is that all of them use the model's state for activity tracking. Nevertheless, as discussed in Wainer and Zeigler (2000) the level of activity depends not only on the state of the particular cell, but also on the level of activity of the neighbourhood. Therefore, we propose analysing the level of activity as the number of external inputs. From the point of view of cell-DEVS, a high level of internal events can be ignored: a cell with high internal activity will produce state changes that will be transmitted to the neighbours. Those changes will be detected by the neighbours and this will increase the overall local activity. If the activity level remains high, the neighbours will react and will influence the original cell again. If that does not happen, and these changes are ignored by the neighbouring cells, that means that this is not a relevant event, and the area is, for all purposes, inactive. Likewise, a very active cell that only includes internal events and does not communicate them to the neighbouring cells (for instance, because we decided to use a quantum size much larger than the values of internal events occurring on the cell). This is equivalent to have a non-active cell (i.e., the numerous internal events are filtered by the quantiser; therefore, they are not of interest and should not be simulated).

If this filtering is not done on purpose, this creates a problem like the one discussed in Figure 7: the cells change internally from time t to $t + h$; but the change is small compared to the quantum, the cell becomes inactive, and this could result in the whole simulation to stop. In Muzy et al. (2005), the problem was solved by introducing a new plane parallel to the others, which was used to keep the model running. Figure 20 shows a different method to solve this problem.

This version is still divided in four phases as discussed in Section 2. In this case, the first rule applies to unburned cells, whose temperature in the next step will be higher than its current one. The second rule is based on the same principles, but it applies to burning cells. The third rule is used when the cells start burning to modify the temperature and the ignition time. The fourth rule updates the ignition time and the temperature of burning cells when their temperature is decreasing. The fifth rule sets the burned flag (temperature equals 209K) when a burning cell crosses down the 333K threshold. In this case, the temperature *temp* is stored as a cell's input/output port and the ignition time *ti* in a state variable (*temp* is passed to the neighbour cells, while *ti* value is only used internally to the cell). Finally, if none of the rules applies, we execute the last rule, which keeps the cell active.

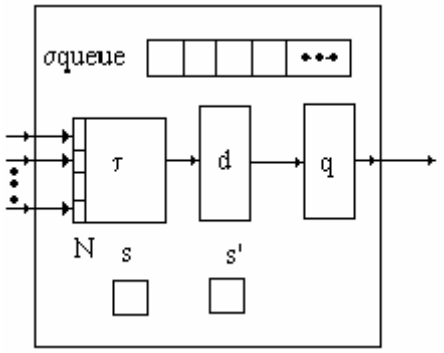
Similarly, when we analyse the original quantiser shown in Figure 4, this organisation causes problems: the computed value is quantised, and if the result is below the boundary, the cell is considered quiescent and it passivates. Nevertheless, this is not the case (for instance, in the model above: if we use a quantum size of 200, the cell will passivate quickly and the simulation will end. The cells are still active internally). To deal with these issues, we changed the cell organisation and quantiser to be as in Figure 21.

Summarising this new cell organisation the previous discussions, a cell will first compute the next state for the cell using function τ. If the new value obtained is the same than the previous for the cell (i.e., $s = s'$), then the cell passivate. Otherwise, the next state change is computed, using dead reckoning for computation of the next boundary using the delay function d. After the delay is consumed, the delayed value is quantised by the quantiser q, and if the threshold is passed, the value is transmitted.

Figure 20 Multiple variables and ports: keeping the model running

```
rule : { ~temp := #macro(unburned); } 1 { (0,0)~temp!=209 AND (0,0)~temp<573
              AND #macro(unburned) > (0,0)~temp  }    %Unburned

rule : { ~temp := #macro(burning); } 1 { (0,0)~temp>333 AND ( (0,0)~temp<573
             OR  (0,0)~ti!=1.0 ) AND (0,0)~temp > #macro(burning) }
rule : { #macro(burning) } 1 { (0,0)> 333 AND ( (0,0)< 573 OR $ti != 1.0) AND
              (0,0)>#macro(burning) } %Burning

rule : { #macro(burning) } { $ti := if($ti = 1.0, time/100, $ti); } 1
               { (0,0)>=573 AND #macro(burning)>=(0,0) }
rule : { #macro(burning) } { $ti := time / 100; } 1
                 { $ti=1.0 AND (0,0)>=573 AND #macro(burning)<(0,0) }  % ti

rule : { ~temp := 209; } 100 { (0,0)~temp > #macro(burning) AND
         (0,0)~temp <= 333 AND (0,0)~temp != 209 }  %Burned

rule : { (0,0) } {~t := time+1 } { t }  % Otherwise, keep the model running
```

Figure 21 New quantised atomic cell

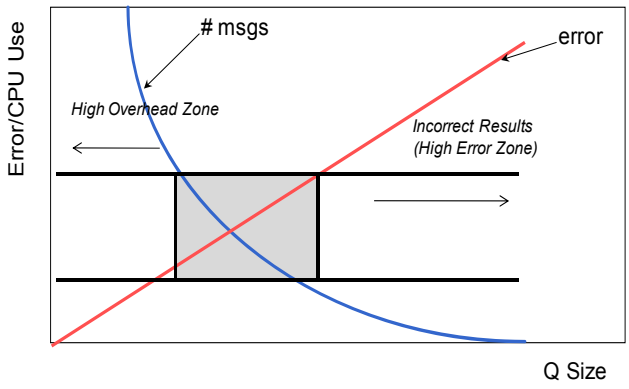


In addition, the idea is to compute the level of activity at the cell's inputs *N*, and we can apply our heuristics for adjusting the quantum size depending on this level of activity.

4.5 Quantum size adjustment

One of the main problems with dynamic quantisation is how to choose the quantum size, and how to limit the changes in order to obtain good execution results. As discussed in Section 4.5, the idea is to compute the level of activity at the cell's inputs *N*, and to apply our heuristics for adjusting the quantum size depending on this level of activity. Nevertheless, we need to control the quantum size: although a limit for the relation between message number and error was found in Zeigler (1998), which is depicted in Figure 22 using dynamic quantisation methods (which change the quantum size dynamically) could have problems. In order to do this, we propose a method based on the working set algorithms used traditionally for virtual memory management in demand paging operating systems. The idea of these algorithms is to use information provided implicitly by the running process to adjust the virtual memory size. In our adaptation, we use the quantum as a measure: a large quantum size means that the number of messages (and CPU time) is reduced, but the error increases; a small quantum size implies a small error (and a cost in speed). Then, when we have a large number of messages, most of the time the CPU will be busy computing results with a high precision that is probably not needed.

Figure 22 Message frequency model (see online version for colours)

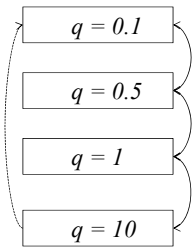


Considering these facts, we propose to control the quantum size indirectly, using a method that gave good results when applied to model virtual memory systems. This method, called the "page fault frequency model" (Wulf, 1969) is an algorithm that allows avoiding thrashing with a very low level of overhead. Our technique, called the 'message frequency model', applies the same strategy and it studies the number of messages in a cell, using this information to constrain the modification of the quantum sizes. As seen in Figure 22, when the level of activity increases, so it does the number of messages involved in the simulation (exponentially). Instead, if the number of messages is low, that means that the degree of activity is low (and we have a very large error). In Figure 22, this means that we could end outside the shaded area, in which the number of messages is small and the error very high, or the error is limited and the overhead high. The goal of this method is to remain in the shaded area. To do that, we count the number of messages in a period, and choose a minimum and maximum threshold (defined by the horizontal lines in the figure), which allows maintaining the level of activity high, while reducing the simulation overhead. If the number of messages is below the lower boundary for the cell, this means that the amount of error is very high. Therefore, at that point, we must reduce the quantum size. Likewise, if the number of messages is too high, this is a signal of high overhead, and we increase the quantum size.

Deciding the quantum size based on this algorithm only introduces a minimum extra overhead (we just need a timer and a message counter, and to compare these values in order to decide if the quantum size should change and how).

A different method for adjusting the dynamic quantum efficiently is to limit the number of quantum sizes to a reduced number of categories, only allowing changing between them, as in Figure 23.

Figure 23 Multiple quantum categories



The idea is to have a few limited quantum categories, starting with a small size (in our example, 0.1). If the activity is high (i.e., we surpass the threshold), we switch to the larger quantum category. If activity is low, we reduce quantum to a smaller category. In this way, the model will adjust to the right quantum size dynamically: if we have a very active model, it will change to the larger quantum, until it is large enough for the current level of activity. At that point, the quantum size will not change until there is a change in the activity level. The idea is that, when the threshold is not passed, we stay at the current quantum

level. In order to avoid the problems of Figure 7, the simulator for each model will change to a lower category after a given amount of time, after which the quantum is adjusted to the lower category. A more aggressive policy is to move directly to the smallest quantum category, giving the model a second chance to run with high precision.

5 Conclusions

We have introduced different methods based on DEVS and cell-DEVS, which can be used to speed up the simulation and simplify the modelling of fire spreading and other environmental spatial models. The methods use a combination of quantised DEVS, dead reckoning, and dynamic quantum adjustment according to the level of activity. We have showed how these methods can be used to track the activity in the model. In the long term we want to provide guidelines to reduce the intervention by the modeller to a minimum, based on heuristics selection of the different techniques.

The method also allows the modellers to use a simple set of equations. This provides evolvability, and models can be modified by non-experts in the field (as they learn about the phenomenon under study). We also showed a different method for finding the equations to determine the time the cell will pass a boundary (instead of computing the values of the equations as a function of time). This value, determined by the quantum of a Q-DEVS model, should be used as a function of the current temperature.

The techniques presented here were able to improve performance, and in the case of combining them with parallel processing, we could simulate a 50-hour wildfire in approximately 20 minutes, giving the chance to conduct multiple studies in real time for fire contention. Quantisation was implemented by calculating the time steps between temperatures, instead of the temperatures at time steps. This achieved the goal of keeping all cells inactive until a significant event takes place. Another modification was to keep cells in the unburned state passive until they are seen to reach the ignition temperature. This increased performance, but had problems with accuracy, and required some prior knowledge of how the fire would develop to obtain good equations. We found that the general direction and speed of fire spread was maintained by our model, although some finer details such as peak temperatures and temperatures of cells at the fire front lost accuracy.

We are currently experimenting with new extensions and with the implementation of some of the techniques introduced in this article. We are interested in providing generic guidelines, and define a systematic method in order to help the modeller to decide which of the different strategies is best suited for a particular problem, based on the characteristics of the problem and the objectives to achieve.

References

Balbi, J.H., Santoni, P.A. and Dupuy, J.L. (1999) 'Dynamic modeling of fire spread across a fuel bed', *International Journal of Wildland Fire*, Vol. 9, No. 4, pp.275–284.

Bandini, S. and Pavesi, G. (2002) 'Simulation of vegetable population dynamics based on cellular automata', *Proceedings of 5th International Conference on Cellular Automata for Research and Industry*, Geneva, Switzerland, LNCS, Vol. 2493.

Barros, F. and Ball, G.L. (1998) 'Fire modeling using dynamic structure cellular automata', *3rd International Conference on Forest Fire Research*, *14th Conference on Fire and Forest Meteorology*, Vol. 1, pp.879–888.

Barros, F.J. and Mendes, M.T. (1997) 'Forest fire modelling and simulation in the DELTA environment', *Simul. Pract. Theory*, Vol. 5, No. 3, pp.185–197.

Berjak, S.G. (2002) 'An improved cellular automaton model for simulating fire in a spatially heterogeneous Savanna system', *Ecol. Model.*, Vol. 148, No. 2, p.133.

Bevins, C.D. (2011) *fireLib User Manual and Technical Reference*, 20 June [online] http://www.fire.org/downloads/fireLib/1.0.4/firelib.pdf.

Bianchini, A., Indovina, F. and Rinaldi, E. (1999) 'Cellular automata for the study of the diffusion of pollutants within the basins of the lagoon: the case of the Venetian lagoon', in *Proceedings of 6th International Conference on Computers in Urban Planning and Urban Management*. Venice, Italy.

Bolduc, J. and Vangheluwe, H. (2003) 'Mapping ODEs to DEVS: adaptive quantization', *Summer Computer Simulation Conference*, pp.401–407, Montréal, Canada.

Bonaventura, M., Wainer, G. and Castro, R. (2013) 'A graphical modeling and simulation environment for DEVS', *Simulation: Transactions of the Society for Modeling and Simulation International*, January, Vol. 89, No. 1, pp.4–27.

Cai, W., Lee, F.B.S. and Chen, L. (1999) 'An auto-adaptive dead reckoning algorithm for distributed interactive simulation', in *Proceedings of Pads*, p.82.

Cellier, F.E. and Kofman, E. (2006) *Continuous System Simulation*, Springer-Verlag, New York.

Dynasim Laboratories (2004) *Dassault Systems*, Dymola [online] http://www.3ds.com/products-services/catia/capabilities/modelica-systems-simulation-info/dymola/ (accessed January 2014).

Dzwinel, W. (2004) 'A cellular automata model of population infected by periodic plague', in *Proceedings of ACRI*, *LNCS*, Vol. 3305, pp.464–473.

Filippi, J., Morandini, F., Balbi, J.H. and Hill, D.R.C. (2010) 'Discrete event front-tracking simulation of a physical fire-spread model', *Simulation*, Vol. 86, No. 10, pp.629–646.

Giambiasi, N., Escude, B. and Ghosh, S. (2000) 'GDEVS: a generalized discrete event specification for accurate modeling of dynamic systems', *Transactions of the SCS*, Vol. 17, pp.120–134.

Gutowitz, H. (1995) 'Cellular automata and the sciences of complexity. Part I-II', *Complexity*, Vol. 1, No. 5, pp.16–22.

Hodgkin, A. and Huxley, A. (1952) 'A quantitative description of membrane current and its application to conduction and excitation in nerve', *Journal of Physiology*, Vol. 117, No. 4, pp.500–544.

Inghe, O. (1989) 'Genet and ramet survivorship under different mortality regimes – a cellular automata model', *Journal of Theoretical Biology*, Vol. 138, No. 2, pp.257–270.

Jafer, S. and Wainer, G. (2010a) 'Conservative DEVS: a novel protocol for parallel conservative simulation of DEVS and cell-DEVS models', in *Proceedings of Proceedings of the Spring Simulation Multiconference*.

Jafer, S. and Wainer, G. (2010b) 'Conservative vs. optimistic parallel simulation of DEVS and cell-DEVS: a comparative study', *Proceedings of the 2010 ACM/SCS Summer Computer Simulation Conference*.

Johnston, P., Kelso, J. and Milne, G.J. (2008) 'Efficient simulation of wildfire spread on an irregular grid', *International Journal of Wildland Fire*, Vol. 17, No. 5, pp.614–627.

Kofman, E. (2003) 'Quantized-state control. A method for discrete event control of continuous systems', *Latin American Applied Research Journal*, Vol. 33, No. 4, pp.339–406.

Kofman, E. and Junco, S. (2001) 'Quantized state systems. A DEVS approach for continuous systems simulation', *Transactions of SCS*, Vol. 18, pp.123–132.

Kofman, E., Cellier, F. and Migoni, G. (2011) 'Continuous systems simulation and control', in G. Wainer and E. Mosterman (Eds.): *Discrete-Event Modeling and Simulation: Theory and Applications* (*Computational Analysis*, *Synthesis*, *and Design of Dynamic Systems*, CRC Press.

Lin, K. (1995) 'Dead reckoning and distributed interactive simulation', in *Distributed Interactive Simulation Systems for Simulation and Training in the Aerospace Environment; Proceedings of the Conference*, Orlando, FL, USA, pp.16–36.

Lin, K.C. and Schab, D.E. (1994) 'The performance assessment of the dead reckoning algorithms in DIS', *Simulation*, Vol. 63, No. 5, p.318.

Liu, Q. and Wainer, G. (2007) 'Parallel environment for DEVS and cell-DEVS models', in Q. Liu and G. Wainer (Eds.): *Simulation: Transactions of the Society for Modeling and Simulation International*, Vol. 83, No. 6, pp.449–471.

Liu, Q. and Wainer, G. (2010a) 'Accelerating large-scale DEVS-based simulation on the cell processor', *Proceedings of TMS/DEVS*, Orlando, FL.

Liu, Q. and Wainer, G. (2010b) 'Exploring multi-grained parallelism in compute-intensive DEVS simulations', *Proceedings of the 24th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS)*, Atlanta, GA.

Liu, Q., Wainer, G., Lu, L. and Perrone, M. (2010) 'Novel performance optimization of large-scale discrete-event simulation on the cell broadband engine', in *Proceedings of High Performance Computing and Simulation* (*HPCS*), *International Conference on*, pp.108–114.

Moon, Y., Zeigler, B., Ball, G. and Guertin, D.P. (1996) 'DEVS representation of spatially distributed systems: validity, complexity reduction', *Proc.of 6th Annnual Conference on Artificial Intelligence, Simulation & Planning in High Autonomy Systems*, La Jolla, CA, pp.288–296.

Muzy, A., in nocenti, E., Aiello, A., Santucci, J. and Wainer, G. (2005) 'Discrete-event modeling and simulation of fire spreading across a fuel bed', *Simulation: Transactions of the Society for Modeling and Simulation International*, Vol. 81, No. 2, pp.103–117.

Muzy, A., Wainer, G., Innocenti, E., Aiello, A. and Santucci, J. (2002) 'Dynamic and discrete quantization for simulation time improvement: fire spreading application using the CD++ tool', *Proceedings of Winter Simulation Conference San Diego*, USA.

Nutaro, J. (2003) *Parallel Discrete Event Simulation with Application to Continuous Systems*, PhD thesis, University of Arizona, Tucson, AZ.

Rothermel, R. (1972) *A Mathematical Model for Predicting Fire Spread in Wildland Fuels*, Research Paper INT-115.Ogden, UT: US Department of Agriculture, Forest Service, in termountain Forest and Range Experiment Station, Vol. 40.

Van Schyndel, M., Wainer, G.A., Goldstein, R., Mogk, J. and Khan, A. (2014) 'On the definition of a computational fluid dynamic solver using cellular discrete-event simulation', *Journal of Computational Science* (Elsevier), November, Vol. 5, No. 6, pp.882–890.

Vasconcelos, M., Pereira, J. and Zeigler, B. (1995) 'Simulation of fire growth using discrete event hierarchical modular models. EARSeL', *Advances in Remote Sensing*, Vol. 4, No. 3, pp.54–62.

Wainer, G. (2002) 'CD++: a toolkit to develop DEVS models', *Software Practice and Experience*, Vol. 32, No. 3, p.1261.

Wainer, G. (2004) 'Performance analysis of continuous cell-DEVS models', in *Proceedings of High Performance Computing & Simulation; 18th European Simulation Multiconference*, Magdeburg, Germany.

Wainer, G. (2009) *Discrete-Event Modeling and Simulation: A Practitioner's Approach*, Taylor and Francis, Boca Ratón, FL.

Wainer, G. and Castro, R. (2010) 'A survey on the application of the Cell-DEVS formalism in cellular models', *Journal of Cellular Automata*, Vol. 5, No. 6, pp.509–524.

Wainer, G. and D'Abreu, M. (2014) 'Using a discrete-event system specifications (DEVS) for designing a modelica compiler', *Advances in Engineering Software*, Elsevier, September.

Wainer, G. and Davidson, A. (2007) 'Defining a traffic modeling language using cellular discrete-event abstractions', *Journal of Cellular Automata*, Vol. 17, No. 4, p.10.

Wainer, G. and Giambiasi, N. (2001) 'Application of the cell-DEVS paradigm for cell spaces modeling and simulation', *Simulation*, Vol. 76, No. 1, pp.22–39.

Wainer, G. and Liu, Q. (2009) 'Tools for graphical specification and visualization of DEVS models', *Simulation*, Vol. 85, No. 3, pp.131–158.

Wainer, G. and Zeigler, B.P. (2000) 'Experimental results of timed cell-DEVS quantization, AI and simulation', *AIS 2000*, pp.203–208, Tucson, AZ.

Wang, S. and Wainer, G. (2014) 'A simulation as a service methodology with application for crowd modeling, simulation and visualization', Accepted for publication in *Simulation: Transactions of the Society for Modeling and Simulation International*, October 2014.

Wulf, W. (1969) 'Performance monitors for multiprogramming systems', *Proceedings of the 2nd.ACM Symposium on Operating Systems Principles*, pp.175–181.

Zeigler, B.P. (1998) *DEVS Theory of Quantization*, Tech. Rep. DARPA Contract N6133997K-0007, ECE Department, the University of Arizona, Tucson, AZ.

Zeigler, B.P. (2005) 'Continuity and change (activity) are fundamentally related in DEVS simulation of continuous systems', in *Proceedings of Proceedings of AIS*, *Artificial Intelligence*, *Simulation and Planning*, pp.1–17, Jeju Island, Korea.

Zeigler, B.P., Praehofer, H. and Kim, T.G. (2000) *Theory of Modeling and Simulation*, 2nd ed., Academic Press, New York.