Applying Cell-DEVS Methodology for Modeling the Environment

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

Recent research efforts have focused on the analysis of environmental systems using cellular models. Although most of the existing solutions are based on the cellular automata formalism, this technique has some problems that constrain its power, usability and feasibility for studying large complex systems. Instead, combining cellular automata with discrete event systems specifications (DEVS) showed excellent results in terms of quality and performance. Despite these encouraging results, the environmental science/engineering community still prefers more traditional approaches, as DEVS-based techniques require a fundamental change of the modeling and simulation paradigm, while entailing expertise in advanced programming, distributed computing, etc. Cell-DEVS and the CD++ toolkit were created to address these problems: they simplify the construction of complex cellular models by allowing simple and intuitive model specification. The discrete event nature of the formalism provides better precision and performance, and models can run in different simulation environments (single user, real-time, distributed/parallel) without special expertise required. Environmental applications can be easily constructed, making it possible for users with basic training in the techniques and software tools to face the study of complex problems. We present the definition of different models of environmental applications, including the pollution on a basin, fire spreading, watershed formation and viability of a population, focusing on how to define such applications using Cell-DEVS methodology, using an approach that facilitates this paradigm shift.

Keywords: DEVS, Cell-DEVS, environmental models, cellular automata

1. Introduction

In recent years, a number of research efforts have presented solutions for modeling and simulation of environmental systems using cellular models. A cell space organizes the structure of the model of a given physical system by dividing the area of influence into geometrically distributed cells. This approach is useful, as most environmental systems are heterogeneous, and they must consider multiple variables, while being able to analyze their behavior in space and time. The most popular technique to study cell spaces is cellular automata (CA) [1], [2], which have been widely used to describe a variety of complex systems. A cellular automaton is organized as an n-dimensional infinite lattice of cells, each holding a state value and a computing apparatus. Conceptually, these local functions are computed synchronously and in parallel, using the state values of the present cell and its neighbors, as shown in Figure 1.

CA popularity has grown in the last few years, and they have been widely used in environmental sciences. The following is a non-comprehensive list of some recent efforts, which shows the variety of applications in this field that have been solved using this approach.

- Avolio [3] has presented a large-scale surface model for lava and debris flows using CA. The model reproduces the 2002 eruption of the Etna volcano with precision.
- Koh [4] has presented a model of erosion on terrain. The model considers the effect of wind, terrain slope and other factors in the formation of sand dunes.

SIMULATION, Vol. 82, Issue 10, October 2006 635-660

DOI: 10.1177/0037549706073698

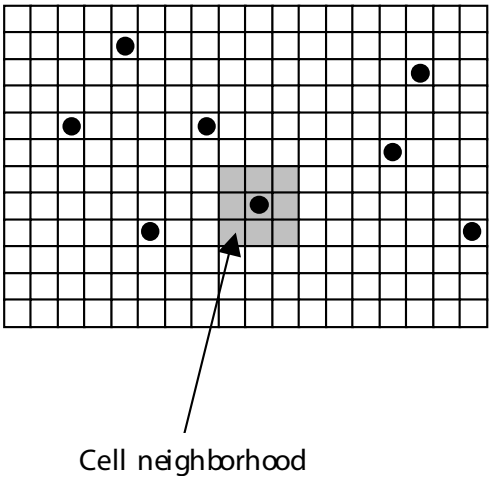


Figure 1. Schematic diagram of a cellular automaton

- El Yacoubi et al. [5] have introduced a simulation tool based on landscape modeling based on CA. Likewise, Pukkala [6] has presented a model on landscape modeling, which considers ecology information for forest planning.

- Colasanti et al. [7] have presented a two-dimensional (2D) cellular automaton to study seed dispersal, clonal expansion, and interactions with adjacent plants, in order to predict the impact of genetically modified plants on a community.

- Bagnoli [8] has presented a model to study competition and sorting of species. The model considers adaptation to environmental factors, and interaction between organisms (competition, association).

- Auger and Faivre [9] have introduced a study on two sibling species of Hippolais (a local bird species) using CA for interspecies competition.

- Molofsky and Bever [10] have presented a discussion on the use of CA to understand ecological communities.

- Dzwinel [11] has presented a model of a population of individuals infected by a periodic plague. The model shows how the age and proximity of the individuals affect the spread of the plague.

- Darwen and Green [12] have presented a study of viability of a population in a field; in Bianchini et al. [13], the spread of pollutants in the Venetian lagoon. Bandini and Pavesi [14] have shown the dynamics of vegetable population using CA.

- Spencer [15] has presented an experimental design to test the growth and survival of the mosquito *Culex pipiens*, studying how scale affects responses.

- Diverse authors [16]–[21] have shown the use of CA for modeling fire spreading phenomena.

As discussed in Muzy et al. [21] and Wainer and Giambiasi [22], CA have some problems that constrain their power, usability and feasibility to analyze complex systems. CA are restricted by the simplicity of their formal description: neighborhood, uniformity of the cells, one discrete state per cell, closure to external events, infinite lattices. According to experimental conditions, cell behavior and neighborhoods often need to be different. One state per cell is also usually not sufficient when dealing with complex deterministic systems. Finally, it can be necessary to have external events changing individual cells [21]. Another series of problems is related to performance. CA use a discrete time base for cell updates, constraining the precision and efficiency of the simulated models (small time steps must be used for higher precision, producing more demanding needs in terms of processing time). Likewise, CA often model systems that are asynchronous in nature using a discrete time implementation, which also makes it very difficult to handle time-triggered activities in each of the cells. Therefore, pure CA cannot be used for many applications, and they often need to be modified for simulation purposes [21].

An approach that has been shown to be useful to address these issues combines CA with discrete event systems specifications (DEVS). DEVS [23] is a formalism defined in the early 1970s to specify discrete event systems organized hierarchically and using a modular description. This strategy allows the reuse of tested models, allowing the reduction of development times [22]. DEVS models run asynchronously; consequently, every cell in a model runs independently from the others. The following is a noncomprehensive list of different efforts that have shown how to use DEVS for constructing cell spaces in environmental science.

- Large ecosystems, including watersheds [24], [25] (a main topic of study in hydrology). Analysis in Zeigler et al. [25] has shown that DEVS enable significant speedups and accuracy, especially for large-scale landscape models.

- Forecasting the development of *Caulerpa taxifolia*, a tropical alga whose severe invasive power was discovered in numerous places in the world [26], [27]. The biological parameters ruling the spread of algae include depth, temperature and substrate; combining this discrete event technique and a geographical information system (GIS) allowed the simulation of algae reproduction over a period of five years.

- Models of fire spreading [28]–[32], enabling the understanding and prediction of fire-related incidents.

Characteristics of the area, such as slope, humidity, wind and vegetation, are considered. The results can be used to determine the best place for firefighters in terms of safety and efficiency.

- Hu and Zeigler [33] have presented a high-performance environment for the modeling and simulation of large-scale models (for instance, fire spread models) based on cellular DEVS. Likewise, Filippi et al. [34] have presented a simulation environment integrating DEVS models and GIS for modeling environmental applications.

In most cases, cellular DEVS utilize continuous state variables and update functions. This requires mechanisms to transform the continuous model into a discrete event model. DEVS has been used recently for continuous systems simulation [23], [34]–[39]. Most of these techniques are based on Q-DEVS [34], whose main idea is to represent continuous signals by the crossing of an equal spaced set of boundaries. The results of these research efforts have shown that discrete event methods in general and DEVS in particular present several advantages:

- computational time reduction (for a given accuracy, the number of calculations can decrease);
- hierarchical modular modeling;
- seamless integration with models defined with other modeling techniques mapped to DEVS;
- simulation of discrete time models (can be seen as particular cases of discrete event methods);
- hybrid systems modeling, in which the discrete event paradigm provides the theory to develop a uniform approach to model and simulate systems with continuous and discrete components.

This approach requires the user to have expertise in advanced programming techniques (including object-oriented programming, distributed computing, standards, etc.). Accordingly, most success stories include multidisciplinary teams (with a group in charge of software development). This makes it difficult for the environmental experts to explore their models at first hand; consequently, we still see many examples of environmental studies developed as specific single-use programs (usually developed for a single study in a high-level language), which permits solving problems of a limited extent, and cannot be easily reused for related experiments. Figure 2 shows an example of a cellular model defined using Cellang (a CA modeling environment [40]), Cell-DEVS, and JDEVS [34] (a similar description of a cellular DEVS model can be found in Ntaimo et al. [30]).

As we can see in Figure 2, in a cellular DEVS environment we have first to define a DEVS model using a standard programming language (i.e., Java for JDEVS or DEVS/Java). Then, we need to define an array of cells and the input/output (I/O) ports interconnection, as shown in Figure 2(d). A CA-based language such as Cellang permits defining problems using simple rules, as we can see in Figure 2(b).

The Cell-DEVS formalism [41], [42] and the CD++ toolkit [43], [44] allow us to overcome these problems. Figure 2(c) shows the rule definition of a cellular DEVS model using the Cell-DEVS formalism in the CD++ environment. Cell-DEVS allows the definition of asynchronous cell spaces with explicit timing definition. This approach is still based on the formal specifications of DEVS, but it allows the user to focus on the problem to be solved by using simple rules for modeling (as with CA). Explicit timing delay constructions can be used to define precise timing in each cell. In this paper, we discuss how these methods can improve the creation of ecological and environmental models. Our approach allows us to enhance the modeling experience in different ways.

- It permits the environmental specialists to apply their expertise while facilitating the change of paradigm. The experts can apply our advanced environment with traditional techniques (CA, finite elements, finite differences, etc.). Then, they can perform a smooth transition from the discrete time to the discrete event based paradigm. This switch is usually difficult because one must change from a view dictating how the cellular model will evolve on the next time unit to a view in which one must think about each cell as an independent entity reacting to external events. Only active cells execute their local computing function, and the execution results are spread out after a predefined delay (only if a state change has occurred). Another source of difficulty is that the continuous local computing function can be quantized, which requires a fundamental shift in thinking about the system as a whole. Experimental data must be collected considering that instead of determining what value a dependent variable will have (its state) at a given time, we must determine at what time a dependent variable will enter a given state (therefore, the data collection must focus on the time for the state changes). The delay function provides a natural mechanism for implementing the quantization function. In addition, our tools facilitate the modeling task, as one does not need to focus on advanced programming or software environments. Wainer and Giambiasi [22] reported a gain in development times when using this approach, because the modelers can focus on defining smaller portions of a problem and expressing it using simpler equations, which can be solved more easily than the complete system, creating a very precise model of each cell.
- It has the potential to improve knowledge generation using a model-based approach. We have put

```
public class MyCellAtomic extends AtomicModel {
   Port in = new Port(this,"in","IN");
   Port out = new Port(this,"out","OUT");
...

public EventVector outFunction(Message m) {
//output function here }

public void intTransition() {
//internal transition function }

public EventVector extTransition(Message m){
//external transition function here   }

public int advanceTime() {
//time advance function here  }
}
```

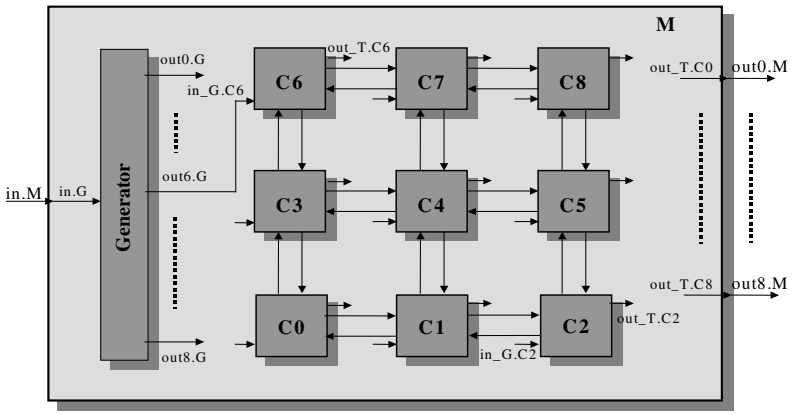
(a)

```
total :=[0,1]+[1,1]+[1,0]+[-1,0]+[0,-1]+
[-1,-1]+[1,-1]+[-1,1]

state:=1 when (total=2 & cell=1) | total=3
state:=0 otherwise
```

(b)

```
[life-rule]
Rule: 1 10 {(0,0)=1 and (truecount=3 or
                  truecount=4) }
Rule: 1 10 { (0,0) = 0 and truecount = 3 }
Rule: 0 10 { t }
```

(c)



(d)

Figure 2. (a) DEVS atomic model applied to a cell space [34]. (b) Cell space coupled model sketch [31]. (c) Life game definition in Cellang [40]. (d) Structure of a cellular DEVS coupled model [47].

into consideration two important issues: how to keep the ability of CA to describe very complex systems using very simple rules (which is its main advantage), and how to bridge the gap between a discrete time and a discrete event description such as DEVS, while allowing users to focus on the application itself. The independent simulation mechanisms permit these models to be executed interchangeably in single-processor, parallel or real-time simulators without any changes. Cell-DEVS enables the definition of specialized behavior in certain areas of the space, thus permitting modeling modified phenomena in particular regions. Such combined analysis is unfeasible using Partial Differential Equations (PDE)s or CA. Likewise, the use of DEVS as the basic formal specification mechanism enables us to define interactions with models defined in other formalisms: individual cells can provide data to those models, and integration between them could enable defining complex hybrid systems. If the complexity of the model increases, the user can still define the cell's behavior using a DEVS atomic model defined in C++ (while using CD++ facilities to create a cell space, I/O ports, execute in parallel, visualize the results or obtain the results through remote execution). This approach provides us with evolvability of the models through a technique that is easy to understand and to map into other existing techniques, while having the potential to evolve into more complex entities.

- It enables high-performance execution. We want to achieve higher precision and improved resolution in the results obtained when executing these models. Only the active cells in the space are triggered, independently from any activation period (as in other cellular DEVS approaches). Cell timing delays can be used to define asynchronous behavior for each

cell. Then, the continuous model defined in each cell can be automatically executed using Q-DEVS. Our previous results confirm that these goals can be achieved using our current methods. Previously we showed [22] that the discrete-event nature of our approach can improve execution times in several orders of magnitude. We have also shown [28], [37] varied experiments in which the application of Cell-DEVS combined with quantization techniques can improve the execution performance of continuous models with an exponential reduction in the total execution time (while introducing error that increases linearly). As the simulation engines are independent from the models themselves, we have introduced a parallel simulation engine, which can improve performance further without modifications to the model [45]. For detailed results on performance results, the reader might refer to Wainer and Zeigler [37], Wainer [38] and Wainer and Chen [45]; for analysis on modeling improvement results, see Wainer and Giambiasi [22], [43] and Muzy et al. [47].

In the following sections, we show how this approach can be applied by environmental systems experts to build advanced models based on cell spaces, keeping complexity low, allowing the modeler to focus on the tasks to solve only, while being able to execute in a high-performance environment. We present different facilities provided by the formalism and the tools, and discuss different examples of application in environmental sciences, showing how a team with basic training can create complex applications without difficulties (the models presented were developed by teams of two over a two-week period, after 12 hours of training, which shows the high learning curve for newcomers into the field). In order to show how to produce a paradigm change while guaranteeing the correctness of the results obtained, we start our experiments by creating cellular models that have been previously validated by other research. In this way, we can guarantee the validity of the results, as we were able to reproduce the original results in every example presented here (presuming that the published results are correct). We show models on diffusion (viability of a population, pollution of the Venetian basin), hydrology and fire spreading, focusing on how our techniques can facilitate the task of the environmental modeler. Then, we modify such models, introducing some of the novel facilities available in CD++ (these results serve as a proof of concept to show how an expert can enhance their existing models based on our methodology). This allows us to show how modelers can address the creation of advanced discrete event models, by extending their basic experience using cellular models. The different examples presented here show the numerous new advanced facilities of CD++ and its application to the field of environmental sciences: execution of models with varied topologies, advanced rule definitions (with multiple state variables in each cell and multiple I/O ports to transfer information between submodels), integration of the results into visualization environments, and seamless execution with high performance (including a parallel simulator and quantization algorithms).

2. Cell-DEVS and CD++

Cell-DEVS was defined as an extension to CA combined with DEVS [23]. DEVS provides a formal framework for the construction of hierarchical modular models. A DEVS model is seen as composed by behavioral (atomic) submodels that can be combined into structural (coupled) models. As the formalism is closed under coupling, coupled models can be seen as new basic models that can be integrated hierarchically [23]. Cell-DEVS defines a cell as a DEVS atomic model and a cell space as a coupled model. Each cell of a Cell-DEVS model holds a state variable and a computing function, which updates the cell state by using its present state and its neighborhood (Figure 3).

A Cell-DEVS atomic model is defined as TDC = $\langle X, Y, S, N, delay, \delta_{ext}, \delta_{int}, \tau, \lambda, D\rangle$. A cell uses a set of N input values to compute its future state, which is obtained by applying the local function τ. A *delay* function is associated with each cell, after which the state values are transmitted. After this basic behavior for a cell is defined (δ_{ext}, δ_{int}, λ and D are defined in Cell-DEVS formal specifications), a complete cell space can be built as a coupled Cell-DEVS, GCC = $\langle Xlist, Ylist, X, Y, n, \{t_1, \ldots, t_n\}, N, C, B, Z\rangle$. A coupled Cell-DEVS is an array of atomic cells (C), each of which is connected to the cells in the neighborhood (N). The border cells (B) can be provided with a different behavior than the rest of the space. The Z function defines the internal and external coupling in the model. *Xlist* and *Ylist* are used for defining the coupling with external models, while the neighborhood definition is used for the internal couplings.

The performance of models with continuous variables is constrained by the number of activations of the simulator (which, in general, uses a discrete time approach). In recent years, DEVS has been extended to accommodate this type of continuous and hybrid system using a more efficient approach. Most existing solutions in this area are based on the concept of quantized state functions (Q-DEVS) [34]. The main idea of Q-DEVS is to represent continuous signals by the crossing of an equal spaced set of boundaries (defined by a quantum size). This operation reduces substantially the frequency of updates, while potentially incurring into error. QSS (quantized state systems) [36] extends Q-DEVS with hysteresis. Differential equation systems can be approximated by legitimate DEVS models with QSS; the addition of hysteresis removes the possible infinite number of transitions performed by a model in a finite time interval, and the existence of a minimum time interval between events constitutes a sufficient condition to obtain legitimate models.

Cell-DEVS with dynamic quantization [37], [38] tries to reduce the error of a cell-QDEVS simulation by improv-

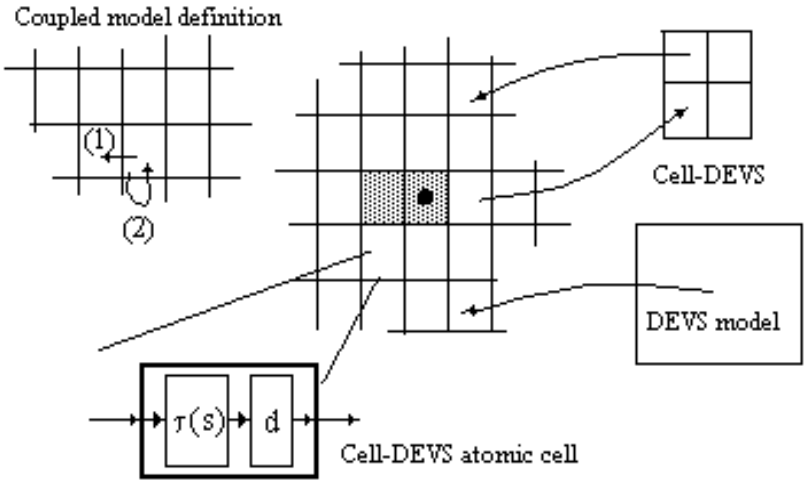


Figure 3. Informal definition of Cell-DEVS

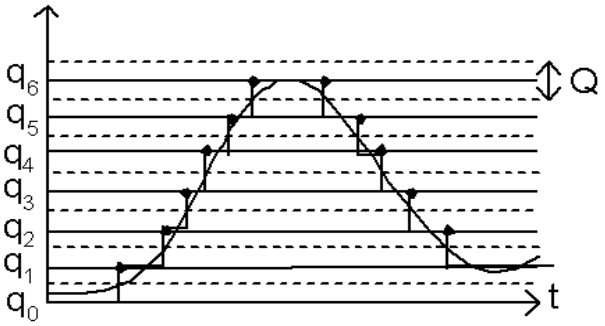


Figure 4. Signal quantization

ing its precision. An active cell can appear as quiescent as a result of a quantum size covering the activity area; if the quantum size is reduced, a smaller error will be obtained. Simultaneously, if we increase the quantum size in the cells with steep update functions, execution time can be improved at a low cost in terms of the error introduced. Two different strategies were proposed to adjust the quantum. Strategy 1 reduces the quantum size for cells with high update rates, and increases it for cells with low update rates. Strategy 2 reduces the number of activations at a cost of a higher error. 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= *Last Threshold Value*, v'= *new value*, regionChange$(v,v',q) = (v = \phi | q = 0 \mid (q \neq 0 \wedge [v/q] \neq [v'/q]))$ and $q(0) = q$, then

strategy 1 $\neg$ regionChange$(v,v',d) \Rightarrow d = q^*$(1- ratio);
RegionChange$(v,v',d) \Rightarrow d = q^*$(1+ratio);
strategy 2 regionChange$(v,v',d) \Rightarrow d = q^*$(1-ratio);
–regionChange$(v,v',d) \Rightarrow d = q^*$(1+ratio).

GDEVS [39] uses polynomials of arbitrary degree (as opposed to constant values) to represent the piecewise I/O trajectories of a DEVS model. In GDEVS, the system is modeled through piecewise polynomial coefficients. As these have piecewise constant trajectories, we can build a discrete event abstraction in the coefficient space using the concept of coefficient event (an instantaneous change of, at least, one of the values of the coefficients defining the polynomial trajectory). An event is a list of coefficient values defining a polynomial describing the trajectory.

CD++ implements DEVS and Cell-DEVS theories, including GDEVS, Q-DEVS, QSS and Q-DEVS with dynamic quantization. The general characteristics of the tool, as reported previously [22], [28], [41], [43], are the following.

- The environment is built as a hierarchy of models, each related with a simulation entity.
- CD++ can simulate in single-processor, parallel or real-time modes. The execution engine uses the model's specifications to build one object to control each component in the model hierarchy. The tool includes facilities to run quantized Cell-DEVS, including models with dynamic quantization.
- Atomic and coupled models can be described using a built-in graph-based specification language or C++. The graph-based notation is more abstract, while C++ increases flexibility.
- Most environments for cellular modeling use 2D or three-dimensional (3D) models. Nonetheless, there are a variety of theoretical problems requiring four or more dimensions. Our simulator uses an array of $\Pi_{i=1...n} d_i$ cells for a model with dimension (d_1,

```
[Tension]
type : cell          dim : (40,40)  delay : transport        border : wrapped
neighbors : (-1,-1) (-1,0) (-1,1) (0,-1) (1,-1) (1,0) (1,1) (0,0) (0,1)
localtransition : Tension-rules

[Tension-rules]
rule : 0 100 { statecount(0) >= 5 }
rule : 1 100 { t }
```

Figure 5. A Cell-DEVS specification in CD++: surface tension

$d_2, \ldots, d_n$). CD++ also permits defining *zones* with differentiated behavior, each defined by a different set of rules on the same cellular model.

- Cell-DEVS models are defined using a built-in language. The local computing function evaluates the model's rules until one of them is satisfied (or there are no more rules). The main operators available to define rules include: Boolean, comparison, arithmetic, neighborhood values, time, conditionals, angle conversion, pseudo-random numbers, error rounding and constants (i.e., gravitation, light, Planck, etc.). Figure 5 shows a simple example representing a surface tension model [48], which follows Cell-DEVS formal definitions.

In this case, $Xlist = Ylist = \{\emptyset\}$, the size of the cell space $\{t1, t2\}$ is defined by *dim* (here $t1 = t2 = 40$). The N set is defined by the *neighbors* keyword. The border B is wrapped (cells in one border communicate with the opposite one). Using this information, the tool builds a cell space, and the Z translation function. The local computing function executes two simple rules, which, as shown in Ameghino et al. [49], reproduce the behavior found in surface tension models (such as those founds in oil spills). We have two states: presence (value 1) or absence (value 0) of oil particles. The model represents a "majority vote" system: a cell becomes occupied if at least five of the nine are; otherwise it becomes empty. Figure 6 shows how particles concentrate where there is higher tension, a high-level representation of the majority vote rules.

We are interested in focusing on some of the facilities recently introduced to CD++, which include those given in the following subsections.

2.1 Advanced Rule Definition

The behavior of the local computing functions has been extended to include a set of rules with the form: POSTCONDITION ASSIGNMENTS DELAY PRECONDITION. When the PRECONDITION is satisfied, the cell state changes to the designated POSTCONDITION, and its output is DELAYed for the specified time. If the model's state variables need to be modified, we use the ASSIGNMENTS [47]. CA include only one state variable per cell.

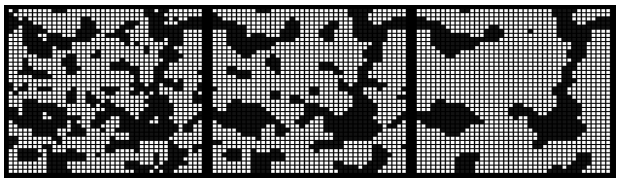


Figure 6. Execution results of the surface tension model

Although the nature of CA (infinite number of cells, n dimensions) provides the mathematical power to solve any computable problem, the model definition with one state per cell is cumbersome, and (as we show in Sections 5 and 6) it can make a complex model too difficult to understand and test. To address this issue, CD++ now allows the definition of multiple state variables, which are declared as follows [47]:

```
StateVariables: pend temp vol
StateValues: 3.4 22 -5.2
rule: {(0,0,0)+$pend} 10 { (0,0,0)>4.5 and
$vol<22.3 }
rule: { (0,0,0)+1 } { $temp:=$vol/2;
$pend:=(0,1,0);}10{(0,1,0) > 5.5}
```

The first line declares the list of state variables, and the second their initial values. Here, the first line assigns 3.4 to *pend*, 22 to *temp* and –5.2 to *vol*. State variables can be referenced as in the third and fourth lines. The identifier ":=" is used to assign values to a state variable. Here, if the condition (0,1,0) > 5.5 is true, the variable *temp* will be assigned half of *vol* value, and *pend* will be assigned the value of the neighboring cell (0,1,0). These assignments are executed immediately (delay functions are not applied to assignments).

Although these new facilities greatly improve the model's definition, sometimes the complexity of the functions is such that some users still prefer to use a programming language. For these cases, we allow the user to define parts of the model using a C++ method, which can be called within a Cell-DEVS specification, as shown in Figure 7. In this case, the user has to define a function (*AP*), which is activated by CD++ upon the execution of the rules.

```
type : cell                                  dim : (5,5,2)
delay : transport                            border : nowrapped
neighbors : (-1,-1,0)(-1,0,0)(-1,1,0)(0,-1,0)(0,0,0)(0,1,0)(1,-1,0)(1,0,0)(1,1,0)(0,0,1)
localtransition : AP

[AP]
rule : { AP(cellpos(0) } 1 { cellpos(2)=0 }
rule : { if( (0,0,0) = 1.0 or (0,0,0) = -83.0, 0.0, 1.0) } 1 { cellpos(2)=1 }
```

Figure 7. Definition of a Cell-DEVS model with an external function in C++

2.2 New Definitions for I/O Ports on Each Cell

When a cell is created, two ports (*NeighborChange* and *Out*) are defined. *NeighborChange* receives values from the neighbors; input ports *In* are created only for those cells connected with external models. *Out* connects the cell with the neighbors, while output ports *OutX* can be defined by the modeler, and they are created dynamically only for the cells that send outputs to other DEVS models.

Besides the basic I/O ports between cells, a user can now define new ports as follows [47]:

```
NeighborPorts: alarm weight number
```

Input and output ports share their names, making it possible to automatically define the translation function: an output from a cell will influence exclusively the input port with the same name in every cell in its neighborhood. A cell can read the value sent by one of its neighbors as follows:

```
rule : 1 100 { (0,1)~alarm != 0 }
rule: { ~alarm := 1; ~weight :=
(0,-1)~weight; } 100 { (0,1)~number
> 50 }
```

If the cell receives an input in the *alarm* port from the cell to the right, and that value is not 0, its status will change to 1, and this change will be sent through the default output port after 100 ms. As it might be necessary to output values through many ports at the same time, the assignment can be used as many times as needed, as seen in the second line: if we receive a value larger than 50 in the port *number* on the cell to the right, we wait 100 ms, generate an output of 1 in the *alarm* port, and we copy the *weight* value received from the cell to the left to the *weight* output port.

2.3 Definition of Varied Topologies

Although most cellular environments use rectangular topologies, it has been shown that square meshes can restrict defining the behavior of advanced cell spaces. Triangular meshes, instead, cover areas of a more varied topology while keeping a limited number of neighbors. Likewise, hexagonal geometries have high isotropy (i.e., the capacity to represent equivalent behavior in every possible direction), which results in more natural model rules. However, these models are more complex to represent, and building visual aids is more complicated. CD++ Lattice Translator allows us to define the cells' behavior using hexagonal and triangular geometries, and to translate them into CD++ models using a square lattice. The mechanism is depicted in Figure 9. For hexagonal meshes, we use a function that shifts alternate rows in opposite directions. The mapping of the triangular lattice is similar: every second cell has a different orientation, and each row of triangles is mapped to one row of squares depending on the parity of $x+y$. This approach allows us to execute these models without needing to recreate the simulation engine, which is still based on a rectangular mesh.

2.4 Advanced Software Technologies

CD++ can execute using a client/server architecture. Users can create models in local workstations, execute them in a remote high-performance simulation engine, then receive, visualize and analyze the results locally with easy-to-use 2D and 3D interfaces. An integrated development environment (IDE) based on Eclipse permits the users to easily interact with model definitions (Figure 10a). Figure 10(b) shows the 2D visualization engine for Cell-DEVS models (which also includes triangular and hexagonal outputs for the Lattice Translator rules). A 3D interface supporting multiview outputs was built using VRML in order to facilitate Web-based visualization (showed in Figure 10c). This 3D graphical user interface (GUI) provides varied functions (select geometries for the nodes, assign colors, edit individual nodes and navigate in the field of visualization). The GUI was designed with the intention of being used by various users from remote locations, and it was extended to run advanced visual models built with Maya, as seen in Figures 10(d), (e) and (f) [50].

3. Modeling Phenomena with Diffusion Behavior

Many existing CA for environmental systems are based on modeling phenomena that can be represented as diffusion models. In this section, we present two such models, focusing on solving them using the facilities provided by Cell-DEVS and CD++. The first model we introduce is devoted to the viability of population growth on a field, defined in Darwen and Green [12] as a CA. The popu-

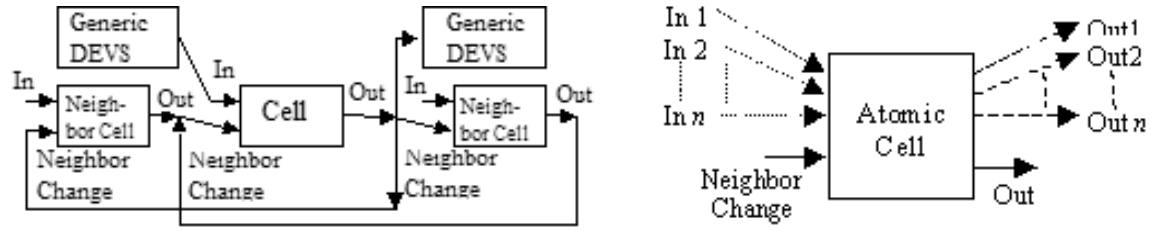


Figure 8. Structure of an atomic cell

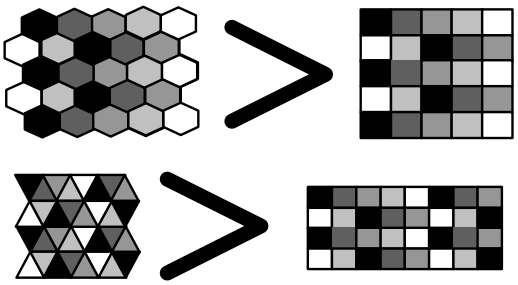


Figure 9. Shift mapping to the square lattice

lation can be vegetal or animal, and the goal is to study the connection between the area initially occupied by the population and its survival chances. Population members can move freely. The CA in Darwen and Green [12] focuses on extinction or wild oscillations, going beyond the traditional reasons for this behavior (namely, populations too small or confined to a small area in high population density). This study has shown that when a population occupies an unconfined region smaller than a critical size, it can also become extinct (even with a healthy population density).

Different parameters influence the population expansion, which could result in indefinite expansion (viability), growth up to a steady state, or extinction. The model considers two types of dynamics: the local, governed by parameters of fertility and maximum population per cell, and migration, which considers population movement from neighboring cells. As discussed in Darwen and Green [12], each cell on the field contains a part of the population, and the dynamics on each cell is defined by the discrete logistics equation (1) for a single population, which applies to a wide range of populations models (but works better for species reproducing seasonally):

$$N(t+1) = rN(t)\left[1 - \frac{N(t)}{K}\right]. \tag{1}$$

Here, Nrepresents the size of the population on a cell, t represents the current time, $t+1$ denotes the next time step, $r \geqslant 0$ is the reproduction rate, and K is the maximum local population on each cell. The model separates the fertility rate $\alpha = (r-1)$ and the population mortality rate $\beta = (-r/K)$, by rewriting equation (1) as follows:

$$N(t+1) = r\,N(t) + \left(\frac{-r}{K}\right)N(t)^2 \Leftrightarrow N(t+1)$$

$$= N(t) + r\,N(t) - N(t) + \left(\frac{-r}{K}\right)N(t)^2$$

$$\Leftrightarrow N(t+1) = N(t) + (r-1)\,N(t) + \left(\frac{-r}{K}\right)N(t)^2. \tag{2}$$

The model also considers migration between the four adjacent cells (N, S, E, W), as follows:

$$\delta^2 N_{x,y} = N_{x,y-1} + N_{x,y+1} + N_{x+1,y} + N_{x-1,y} - \left(4N_{x,y}\right).$$

Consequently, if we add this term to equation (2), we can obtain the behavior of the cell (x, y) at time t:

$$N_{x,y}(t+1) = N_{x,y}(t) + \alpha N_{x,y}(t) + \beta N_{x,y}(t)^2$$

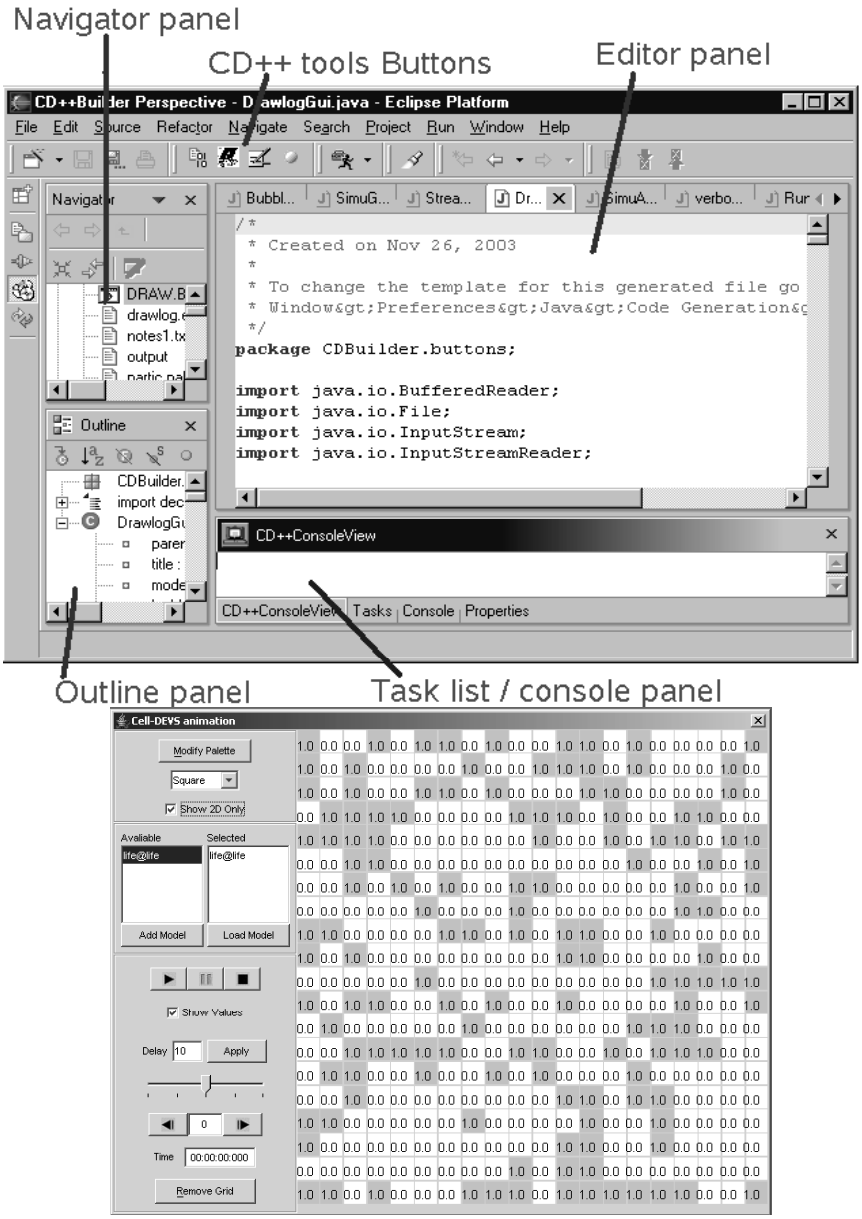
$$+ \gamma\delta^2 N_{x,y}(t). \tag{3}$$

Here, $\gamma < 1$ is the proportion of the population on a cell that is ready to migrate, and it is used as a diffusion coefficient. γ can be also used as a scale factor: as each cell contains the maximum population size in the area, a large γ can be used to represent species moving rapidly or a smaller area. Finally, considering that population is never negative, and equation (3) can violate this condition, the Heaviside operator $H(z)$ is used ($H(z) = z \forall z > 0$, and $H(z) = 0$ otherwise), resulting in

$$N_{x,y}(t+1) = H\left[N_{x,y}(t) + \alpha N_{x,y}(t) + \beta N_{x,y}(t)^2\right.$$

$$\left. + \gamma\delta^2 N_{x,y}(t)\right]. \tag{4}$$

If the population expands up to the borders of the field of study, the population is viable. The population is not affected by external factors, including external immigration.



(a)

Figure 10. GUI and visualization facilities *(continues on next page)*

If the population reaches the value

$$N_{\text{eq}} = \frac{r-1}{r}K,$$

the system is in steady state, and the population in successive generations does not change.

The original model was built as a finite-difference CA using CM-Fortran executing under a CM-2 supercomputer. A specialized graphical output was also built for the project. In our case, the model was developed by a team of two students working part-time for one month. The model can be executed in a CD++ parallel simulation engine [45] without modifying a single line. This presents a substantial improvement with respect to the original approach to solving these problems. The first step of the model creation is the construction of a formal specification (this step, which was carried out in this and the following examples, allows the modeler to think about the problem in an abstract way prior

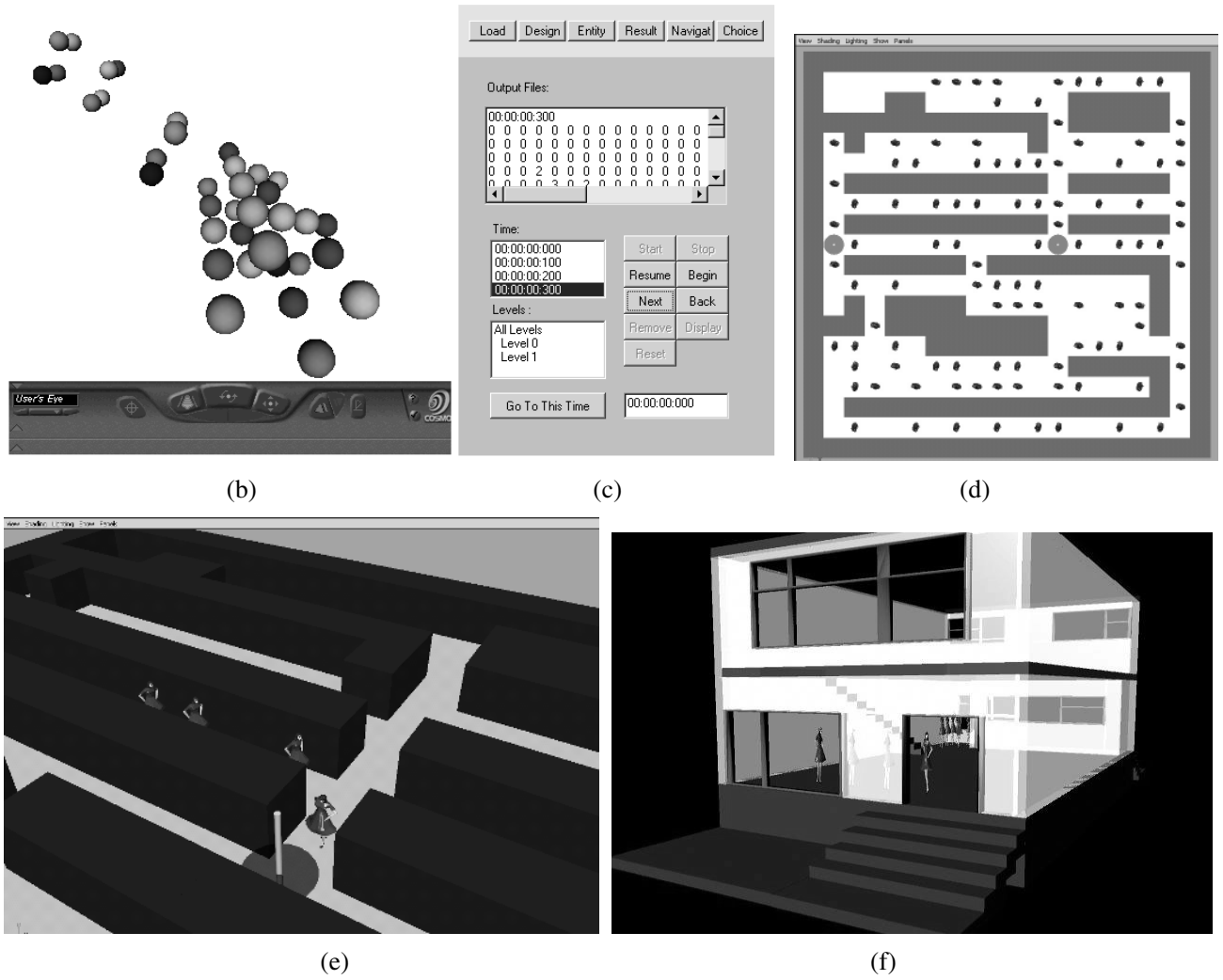


Figure 10. *(continued from previous page)*

to any implementation). The Cell-DEVS coupled model for this application is formally defined by

$$\text{Viability} = \langle Xlist, Ylist, X, Y, n, \{t1, \ldots, tn\}, N, C, B, Z \rangle.$$

$Xlist = Ylist = \{\emptyset\}$; $t1 = t2 = 40$; $N = \{(-1,0), (0,-1), (0,0), (0,1), (1,0)\}$; $C = \{C_{ij} | i \in [1,t1], j \in [1,t2]\}$, where C_{ij} is a Cell-DEVS atomic model; $B = \{C_{ij} | C_{ij} \in C, i \in [1, t1], j \in [1, t2]\}$, and Z is defined by the neighborhood. For each cell C_{ij}, the Cell-DEVS atomic model is defined by $C =< X, Y, S, N, d, \delta\text{int}, \delta\text{ext}, \lambda, \tau, ta >$, with $N \in Z^{\eta}$; $\tau : Z^{\eta} \rightarrow Z$ is defined by equation (4); $d =$ transport. These formal specifications can be directly mapped to CD++ as shown in Figure 11.

The model follows a Cell-DEVS formal specification: a 40×40 cell space using the N/S/E/W neighbors. Using this information, a complete cell space is built. The cell's rules (*localtransition* : *viability*) define the local computing and delay functions using CD++ notation. We use a macro (*calc*) to implement equation (4), using $r = 1.24$, $K = 100$ and $\gamma = 0.22$. The first rule defines a limit for growth. If this threshold is not crossed, and there is activity (i.e., the result of equation (4) is not zero), the next step is computed. In any other case ($t =$ true), the cell becomes extinct. As in any Cell-DEVS, if a cell state does not change, the cell becomes dormant, and it will be reactivated only upon detection of activity. As we can see, modifying this specification is very simple, which enables the modeler to run advanced experiments quickly and easily, without special concerns about the simulation engine or the software development activities. Figure 12 shows graphical simulation results (from CD++) for some basic cases, in which

```
type : cell     delay : transport   dim : (40,40)     border : nowrapped
neighbors : (-1,0) (0,-1) (0,0) (0,1) (1,0)
localtransition : viability

[viability]
rule : #K 1 { #calc > #K}
rule : { #calc } 1 { #calc > 0 }
rule : 0 1 { t }

#BeginMacro(calc)
trunc((#R *(0,0) * (1-(0,0)/#K)) + trunc(#gamma*((-1,0)+(0,-1)+(0,1)+(1,0)- 4 * (0,0))))
#EndMacro
#R = 1.24; #K= 100; #gamma=0.22
```

Figure 11. Cell-DEVS specification of a portion of the model using CD++

the three main behaviors can be established for different fertility and mortality parameters using the same initial conditions.

Figure 12(a) considers a viable population that will expand to the borders of the grid in 200 transitions. We achieve such behavior by using large fertility and small mortality rates ($K = 200$, $r = 1.2$, $\alpha = 0.2$, $\beta = -0.006$ and $\gamma = 0.18$). Figure 12(b) shows a steady-state population, which is a result of the reduction in the fertility rate combined with the small mortality rate ($r = 1.1$, $\alpha = 0.1$, $\beta = -0.0055$). Figure 12(c) shows how the population increases, and diminishes up to extinction ($r = 1.05$, $\alpha = 0.05$, $\beta = -0.00525$). This is the result of an even smaller fertility rate. Figure 13 shows the execution results for a more complex scenario.

Figure 13 shows the combined behavior of expansion and migration. These results are similar to those presented in Darwen and Green [12]: small patches shrink and disappear, medium-sized patches remain constant or oscillate between two sizes, and large patches expand indefinitely. In Figure 13(a), we see that the population initially expands, but later becomes extinct (as a result of low fertility, higher mortality and low mobility). Figure 13(b) shows a viable case as a result of a high fertility rate. In Figure 13(c), the population becomes steady. Although mobility increased, lower fertility makes growth slow, producing a steady-state condition in which the population stops growing.

Performance experiments carried out with this example allow us to show that the approach scales up without any problems. Figure 14 shows the execution times for the first 75 time steps in varied configurations. Similar results were obtained by Wainer and Giambiasi [22] and Wainer and Chen [45] for different cellular models.

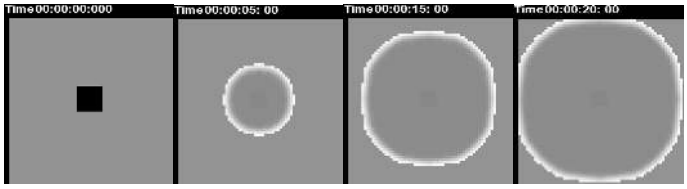
Our second study was carried out using a different set of facilities in CD++. The model was executed remotely by a group of students at the University of Buenos Aires, Argentina, using a CD++ server [44] (installed at 134.117.60.106:9001). The simulation results were presented using a CD++ visualization applet (http://www.sce.carleton.ca/faculty/wainer/wbgraf/). Figure 15 shows the architecture of the solution, which allows us to show our basic facilities for Web-based simulation.

The model developed by the remote team is based on that presented by Bianchini et al. [13], which was a study on the contamination of the Venetian lagoon, produced by substances such as nitrogen and phosphorus. The goal was to learn about this ecosystem, in order to be able to control contamination produced the various lakes of the region. The basic nature of the problem was the creation of an industrial zone, which compromised the sustainability of the lagoon as a result of increased pollution. The manufacturing debris increased the salinity and hydraulic volume (converting it into a sea-like environment), increased the number of algae, and decreased oxygenation and sediment deposits. The authors proposed to monitor these parameters, using simulation for forecasting changes in the ecosystem. To achieve this goal, their research team built a dedicated CA environment. The system, called SeTA (Sea Transformation Automaton), was developed using C/C++ and Unix Bourne Shell scripts, combined with ARC/info.

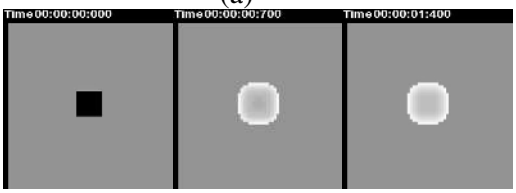
The CA measures the quantity of polluting substance within a concentrated range. A hydrodynamic model represents the lagoon's water velocity and the morphology of the basin. The CA uses the initial scenario of substance concentration, velocity maps and vegetation, and it generates a transformed scenario. Each cell contains the speed of the water flowing in the cell (and its direction), and the level of contamination of the cell. Pollution is produced when the lake receives certain nitrogen from the exterior. The model's rules are the following.

1. The value of the current cell is computed as (0,0) + $\sum$ [((20 – (0,0))/20)*Si*Wi], in which the addition is carried out in all of the cells influencing the cell (0,0). In this case, Si represents the concentration of the pollutant on cell i, Wi the speed on the cell, and Si*Wi is the contribution of a neighboring cell in the direction to cell (0,0). Finally, (20 – (0,0)) / 20 represents the reception capacity of the current cell.

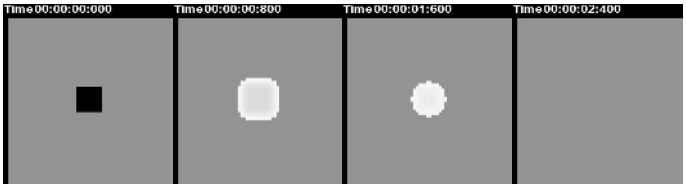
2. After evaluating rule l, and after consuming a delay representing the contamination rate, the new value



(a)

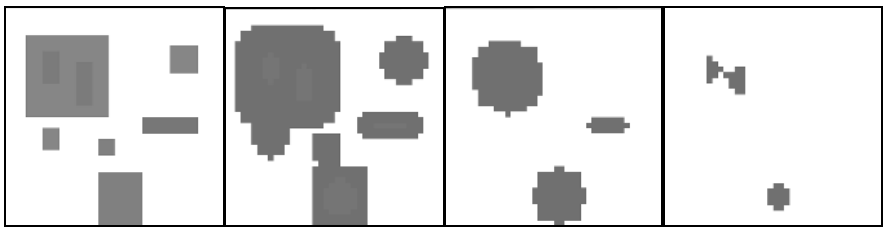


(b)

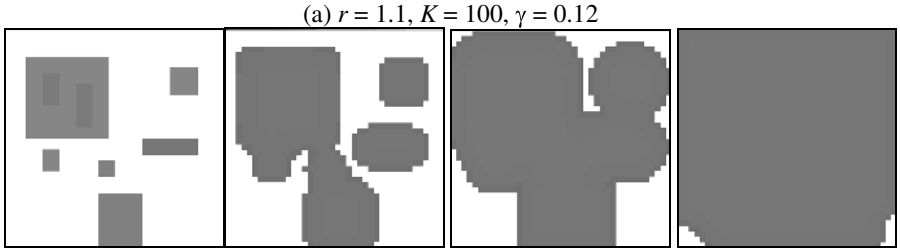


(c)

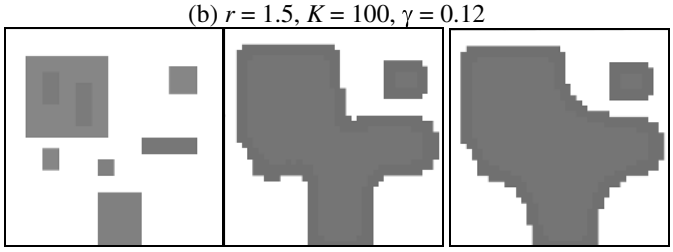
Figure 12. Viability rules: basic behavior



(a) $r = 1.1$, $K = 100$, $\gamma = 0.12$



(b) $r = 1.5$, $K = 100$, $\gamma = 0.12$



(c) $r = 1.25$, $K = 100$, $\gamma = 0.22$

Figure 13. Viability analysis

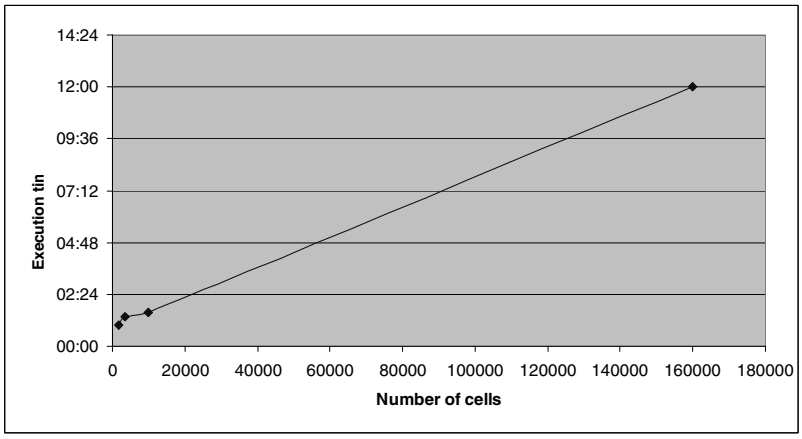


Figure 14. Execution times of the viability model with different sizes of the cell space

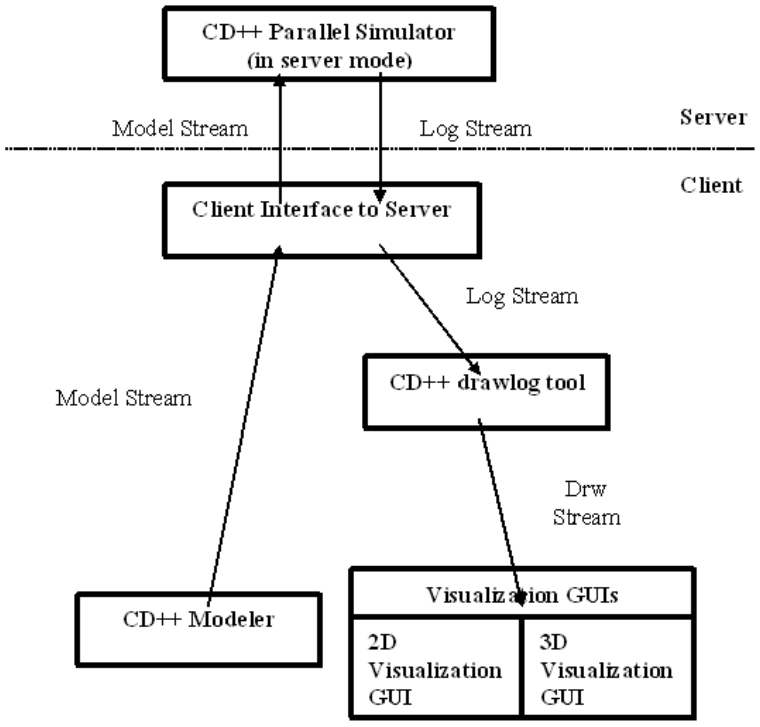
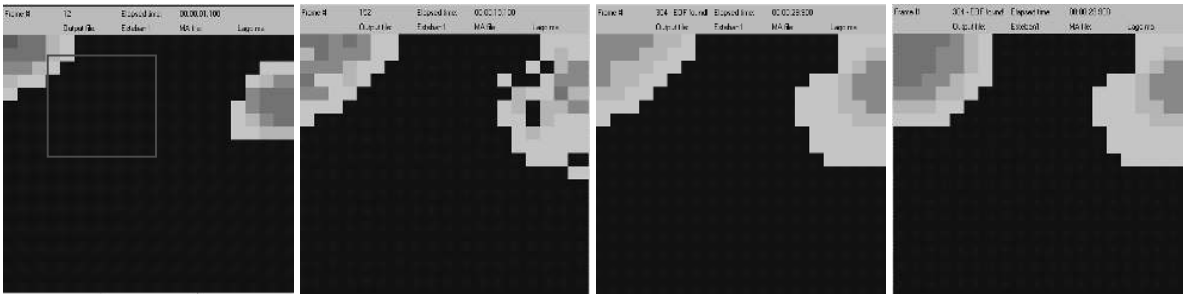


Figure 15. CD++ client/server architecture [44]

of contamination of the cell is computed as (0,0) – [(20 – *Si*/20)*(0,0)**WC*], where *WC* is the velocity on the central cell, *Si* represents the concentration of the contaminant in the cell *i* (the cell receiving pollution), and (20 – SC(*t*))/20 is the capacity of reception of the receiving cell.

3. After evaluating rules 1 and 2, and waiting for the delay representing the time to execute rule 2, the next rule includes the cases in which one or more velocity directions in the surrounding cells enter within the cell, and the water in the origin cell is zero. This is computed as (0,0) + $\sum$ [((20 – (0,0))/20)**Si***Wi*] – [(20 – *Si*/20)* (0,0)**WC*].

4. In any other case (i.e., there is no neighboring cell pointing to the origin, and the speed of water in the cell is not affecting the origin cell), the current pollution value is maintained.



(a) (b) (c) (d)

Figure 16. Two sources of constant pollution

We have run different test scenarios for the model, which have allowed us to obtain results equivalent to those in the original study, as discussed in the following. We show the simulation results in Figures 16–18, in which we distinguish water (black cells) or contamination (lighter gray; darker cells are contaminated). The simulation results illustrate different modes of diffusion and the space–time variations of the concentration of nitrogen in the lagoon, according to the different phenomena taken into consideration (advection, diffusion, vegetation absorption, sources of emission).

In Figure 16, the model is fed with pollution from two different sources (built as standard DEVS models representing nitrogen generators). In this case, we show the simulation results on a continuous focus of contamination during several hours (the factories discharge 560 l h^{-1}). We modified the original model, adding subsurface vegetation (which makes the diffusion of pollutants to be slower). This is implemented as a zone (marked by a square in the figure) in which the model introduces a different behavior than the rest of the cell space. Introducing this change is straightforward, while doing this in the original CA environment requires an extension of the SeTA environment we created for this application.

We can see how the pollutant concentrates in the places where the contaminant is being discharged. As the hydrological map and the presence of vegetation allow stationary water, diffusion is slow. The differences between the second and third images (which represent 24 hours of simulated time) are not large: the microalgae avoid further expansion of the contamination. Figure 16(d) shows the results obtained when vegetation in the model is eliminated. As we can see, the leftmost part of the model does not contain any vegetation (which was the case in Figures 16a–c), and therefore the pollution concentrates and expands more quickly. On the north-east side of the figure, where there was no vegetation, the evolution is exactly the same as in the previous case.

Figure 17 shows the influence of the hydrological map. Here, all the cells have a single direction of movement, using random equiprobable values. Pollution expands more homogeneously, as water flows in every direction. Likewise, we can see that there are higher levels of contamination getting to farther places than in the previous case, contaminating places distant from the source of pollution.

Figure 18 presents the original model (with vegetation and slow probability of absorption of the pollutant by the plants), using a single source of nitrogen at the beginning of the simulation (representing accidents, showing how the toxic elements will spread in these cases). When the source of pollutant stops, the contamination is slowly absorbed by vegetation in the lake. We can see that, although the algae collaborate in eliminating the pollution, the hydrological characteristics of the lake makes a polluted region in the north-west area that does not disappear.

4. Modeling Watersheds

Gutowitz [52] and Gaylord and Nishidate [54] introduced a Cell-DEVS model of a watershed that was previously defined in Ameghino et al. [55] using DEVS/C++. They combined GIS data (topography, soil, rain) with a DEVS environment to project the evolution of a watershed. A watershed is a natural region that acts as the water-receiving area of a drainage basin. The water that accumulates has different origins, such as rain, rivers and melting snow, as shown in Figure 19.

The hydrology model considers the watershed as divided into cells organized in layers: air, vegetation, surface water, soil, ground water and bedrock. The model represents the water flow and accumulations based on the characteristics of the different layers. Water accumulation is computed as shown in Figure 20. Each cell is an atomic model, and then these build an array of spatially organized models.

Here, we show how CD++ can be used to define this model with ease, introducing varied behavior in different areas of the model. To do so, we built a Cell-DEVS version of the original model using CD++. Figure 21 shows the execution results for this model. Figure 21(a) shows the model's initial state, representing the slope of the terrain before rain. Each cell occupies 1×1 m^2. Figure 21(b)

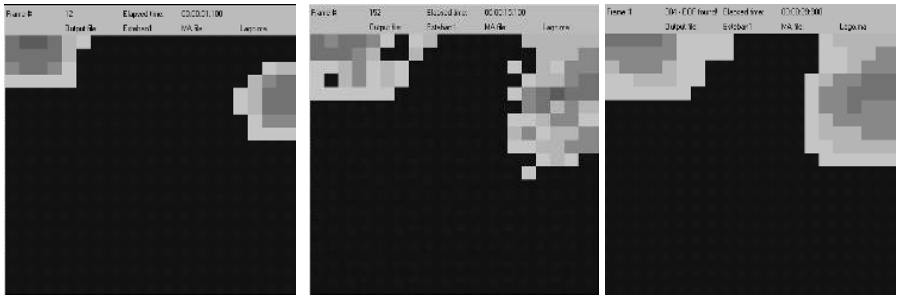


Figure 17. Variation in the hydrology model

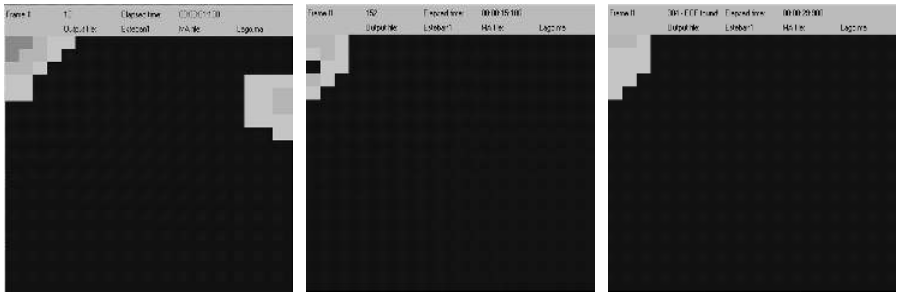


Figure 18. GUI and visualization facilities

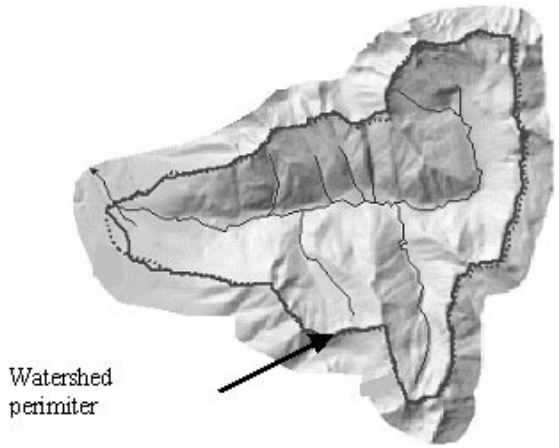


Figure 19. Topography of a watershed

shows the execution results after intense rain (7.62 mm h^{-1}) after 2 h. We can see that the rain is accumulated in the lower levels of the terrain, and a stream is formed.

The height of accumulated water depends on the rain water that reaches the ground, the water received from neighboring cells, the water that overflows to neighboring cells and the water the ground has absorbed. Based on the equations for this model, the CD++ model developed simulates the accumulation of water under the presence of constant rain. The original model in Gutowitz [52] and Ameghino et al. [55]4] assumed the soil in the whole watershed area was of the same type. Here, we show how to expand the original model by defining areas containing different soil types (grass or rock). Figure 22 shows the implementation of the model in CD++: it represents the accumulation of water by taking the present amount of water in the cell, and adding the rain fall up to the present. Then, we consider how much water must be passed to the neighbors, and how much water is received from the inverse neighborhood.

We use a 3D model to defined a different behavior using two overlapped planes, one to represent the height of the water retained (surface 0) and one to represent the topography of the terrain (surface 1). These values for ground elevation remain unchanged throughout the simulation, and they are used to calculate the water overflow to neighboring cells. We use two zones, representing the sets of cells that will model grass and rock areas. For each zone, different sets of rules apply. Each rule calculates the new water height by applying the hydrology model equation. These rules represent the water accumulation changing the surface vegetation and ground filtration parameters, as shown in Figure 23. A modeler with some experience can define this varied behavior with ease.

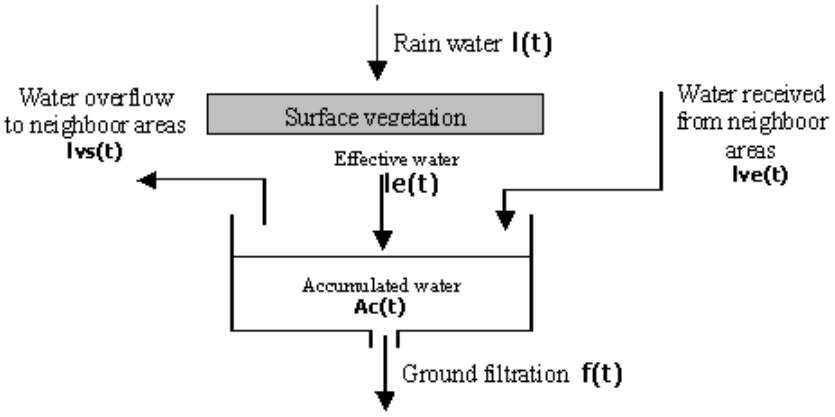
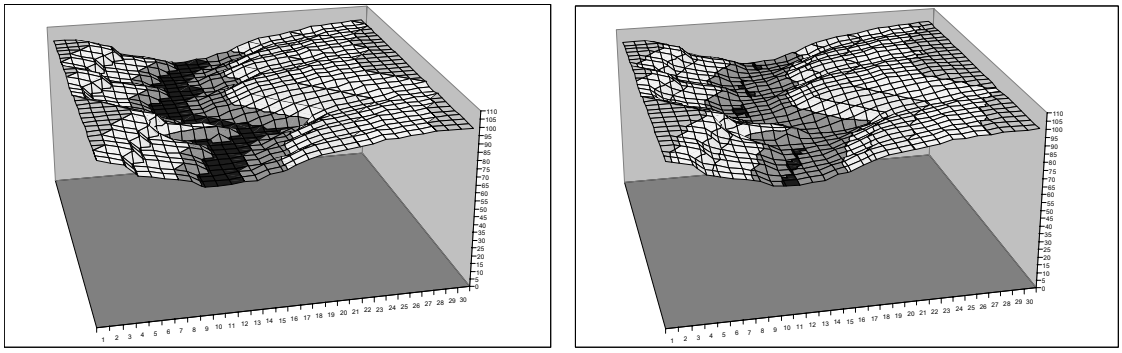


Figure 20. Hydrology model [55]



(a) (b)

Figure 21. Execution of the hydrology model [52]

We repeated the execution of this model using Q-DEVS, QSS and Q-DEVS with dynamic quantization. These results were obtaining by running the hydrology equations in CD++, and activating the corresponding simulation engine at runtime (the user only needs to select the quantization parameters upon execution, and does not need to worry about implementation issues). Figure 24 presents performance results obtained with the different techniques.

Figure 24(b) presents the cumulative error obtained with different strategies for this model. The figure compares the different strategies (using different update ratios for the dynamic DEVS strategies 1 and 2). The results are obtained with standard and hysteresis quantum overlap, because the results of hysteresis quantum differ from the standard when direction changes are present. The lowest error was obtained with dynamic quantum Str1 with ratio 0.9. Str1 results were better than Str2 and standard Q-DEVS (the larger the ratio, the better the result), as expected. The quantum size is adjusted very quickly, which reduces the amount of error obtained. The best execution time was for Q-DEVS and Str1 with an update ratio of 0.05 (larger update ratios adjust the values quicker, reducing the error while increasing the number of messages). When using $q = 0.05$, Q-DEVS provides better results. As each cell increases by approximately 0.07 units in each update, changing the quantum size makes it oscillate around the function value, resulting in an increase in the total simulation messages. This does not occur with fixed quantum size. Likewise, once the quantum size varies, the dynamic quantum strategies have lower message interchange. A low update ratio improves the number of messages involved, while increasing the error. Paying a small cost in the extra execution overhead, we were able to reduce the error involved (up to 75%). Str2 reduces the number of messages using higher rates when compared to Str1, but incurring in a higher amount of error. If we consider, for instance, $q = 1$ with Str1 and ratio 0.9, the amount of error introduced is minimum and the number of messages has been greatly

```
type : cell           dim : (30,30,2)
delay : inertial      border : nowrapped
neighbors : (-1,0,0) (0,-1,0) (0,0,0) (0,1,0) (1,0,0)(-1,0,1) (0,-1,1) (0,0,1)(0,1,1) (1,0,1)
zone : grass { (0,0,0)..(29,10,0) }
zone : stones { (0,20,0)..(29,29,0) }

[grass]
rule : {0.07 + (0,0,0) - if(((((0,0,1) + (0,0,0))>((-1,0,1) + (-1,0,0)))),(((((0,0,0) + (0,0,1) -
(-1,0,0) - (-1,0,1))/1000) * (0,0,0))/1000),0) - if(((((0,0,1) + (0,0,0))>((1,0,1)  +
(1,0,0)))),(((((0,0,0) + (0,0,1) - (1,0,0) - (1,0,1))/1000) * (0,0,0))/1000),0) - if(((((0,0,1) +
(0,0,0))>((0,-1,1)+(0,-1,0)))),(((((0,0,0) + (0,0,1) - (0,-1,0) - (0,-1,1))/1000) *
(0,0,0))/1000),0) - if(((((0,0,1) + (0,0,0))>((0,1,1) + (0,1,0)))),(((((0,0,0) + (0,0,1) - (0,1,0)
- (0,1,1))/1000) * (0,0,0))/1000),0) + if(((((-1,0,1) + (-1,0,0))>((0,0,1) + (0,0,0)))),((((-1,0,0)
+ (-1,0,1) - (0,0,0) - (0,0,1)) * (-1,0,0))/1000),0) + if(((((1,0,1) + (1,0,0))>((0,0,1) +
(0,0,0)))),((((1,0,0) + (1,0,1) - (0,0,0) - (0,0,1)) * (1,0,0))/1000),0) + if(((((0,-1,1) + (0,-
1,0))>((0,0,1) + (0,0,0)))),((((0,-1,0) + (0,-1,1) - (0,0,0) - (0,0,1)) * (0,-1,0))/1000),0) +
if(((((0,1,1) + (0,1,0))>((0,0,1) + (0,0,0)))),((((0,1,0) + (0,1,1) - (0,0,0) - (0,0,1)) *
(0,1,0))/1000),0) } 1000 { cellpos(2)=0 }
rule : { (0,0,0) } 1000 { t }

[rock]
rule : {0.09 + (0,0,0) - if(((((0,0,1) + (0,0,0))>((-1,0,1) + (-1,0,0)))),(((((0,0,0) + (0,0,1) -
(-1,0,0) - (-1,0,1))/1000) * (0,0,0))/1000),0) - if(((((0,0,1) + (0,0,0))>((1,0,1)  +
(1,0,0)))),(((((0,0,0) + (0,0,1) - (1,0,0) - (1,0,1))/1000) * (0,0,0))/1000),0) - if(((((0,0,1) +
(0,0,0))>((0,-1,1)+(0,-1,0)))),(((((0,0,0) + (0,0,1) - (0,-1,0) - (0,-1,1))/1000) *
(0,0,0))/1000),0) - if(((((0,0,1) + (0,0,0))>((0,1,1) + (0,1,0)))),(((((0,0,0) + (0,0,1) - (0,1,0)
- (0,1,1))/1000) * (0,0,0))/1000),0) + if(((((-1,0,1) + (-1,0,0))>((0,0,1) + (0,0,0)))),((((-1,0,0)
+ (-1,0,1) - (0,0,0) - (0,0,1)) * (-1,0,0))/1000),0) + if(((((1,0,1) + (1,0,0))>((0,0,1) +
(0,0,0)))),((((1,0,0) + (1,0,1) - (0,0,0) - (0,0,1)) * (1,0,0))/1000),0) + if(((((0,-1,1) + (0,-
1,0))>((0,0,1) + (0,0,0)))),((((0,-1,0) + (0,-1,1) - (0,0,0) - (0,0,1)) * (0,-1,0))/1000),0) +
if(((((0,1,1) + (0,1,0))>((0,0,1) + (0,0,0)))),((((0,1,0) + (0,1,1) - (0,0,0) - (0,0,1)) *
(0,1,0))/1000),0) } 1000 { cellpos(2)=0 }
rule : { (0,0,0) } 1000 { t }
```

Figure 22. Watershed model definition with zones for differentiated terrain information

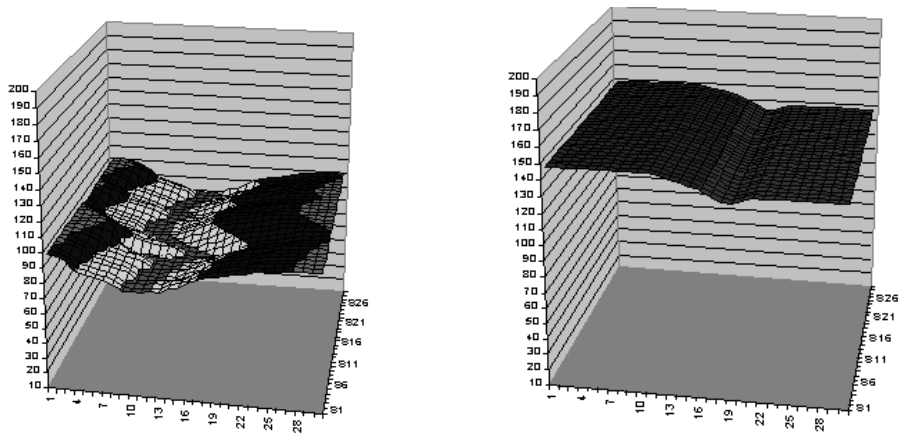
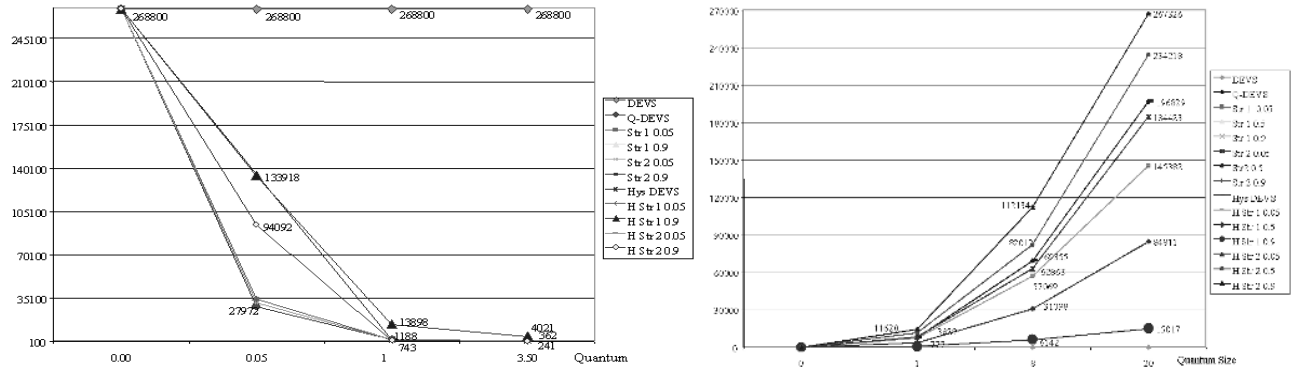


Figure 23. Watershed simulation differentiated zones for terrain information

reduced. If we consider now $q = 3.5$, the error obtained with Str1 is better, while the number of messages involved is comparable.

5. Modeling Fire Spread

Forest fires destroy important resources, and hence enormous efforts have been made to prevent them. Many forest fire models have been developed to study how the fire spreads under different environmental conditions. Wainer and Chen [45] have described a fire model using Cell-DEVS. The model is based on experimental fires conducted on Pinus pinaster litter, in a closed room without any air motion [16]. The study domain is meshed uniformly with cells of 1 cm^2. The physical model is solved by the finite difference method, which leads to the following equation



(a) (b)

Figure 24. Quantized watershed simulation results (a) execution time; (b) cumulative error

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

where T_{ij} is the temperature of a grid node. The coefficients a, b, c and d depend on the considered time step and mesh size [16]. As described in Figure 25, we used two planes representing the different variables in our model. The first plane stores the cell's temperature. The second plane stores the ignition time of the cells in the propagation plane. As discussed in [45], this model accurately reproduces the experimental results. As we can see in Figure 24, we need $n \times m \times 2$ cells (double the size of the simulated area). The new facilities provided by CD++ allow us to save time and memory space by reducing the model to one plane only (see Muzy et al. [47] for a report on the savings).

The first step was to add a state variable *ti* to remove one layer of cells and to replace all the references to this layer by references to the state variable. For instance, two of the original rules in this model were defined as [45]:

```
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) ) AND (0,0,0)!=209 }
  % Burning
rule: { time/100 } 1 { cellpos(2)=1
  AND (0,0,-1)>=573 AND (0,0,0) = 1.0}
  % ti
```

These were replaced by

```
rule: {#burning} 1 {((0,0)>#burning
  AND (0,0)>333) OR
(#burning>(0,0) AND (0,0)>=573) AND
  (0,0)!=209} % Burning
```

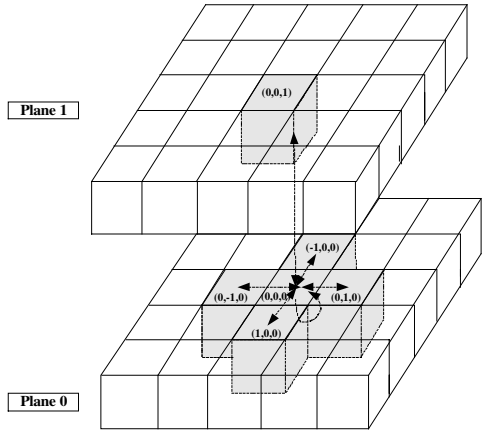


Figure 25. Cell's neighborhood specification [45]

```
rule : { (0,0) } { $ti := time/100;}
  1 { (0,0)>=573 AND $ti = 1.0 } % ti
```

The model does not use multiple planes, as in the previous case, and therefore we do not need to check the plane we are using (*cellpos*). Simultaneously, the references are always to 2D cells. The *ti* rule records the moment when the cell starts burning. In this case, we use a new state variable instead of an independent plane, as in the original model presented in Wainer and Chen [45]. As these rules are more compact, we can manipulate them to obtain better performance and easier understanding. To shorten execution time, the number of rules was reduced and the clauses in the rules' condition reordered. For instance, we can see that both rules need (0,0)!=209, and (0,0)>333 and (0,0)$\geqslant$ 333 $\Rightarrow$ (0,0) $\neq$ 209. Hence, (0,0)!=209 can be removed. Rule *ti* overlaps with the second part of the rule *burning*,

so they were merged. The two rules were merged into one, which will assign the new value to *$ti* depending on *$ti*'s original value:

```
rule : { #burning } 1 { (0,0)> 333
  AND ((0,0)< 573 OR $ti != 1.0) AND (0,0)
  >#burning }
rule : { #burning }{ $ti := if($ti =
  1.0, time/100,$ti); } 1 { (0,0)>=573 AND
  #burning>=(0,0) }
rule : { #burning } { $ti := time /
  100; } 1 { $ti=1.0 AND (0,0)>=573
  AND #burning<(0,0) }
```

A second step optimization is based on the fact that CD++ is capable of using short-cut evaluation (in the same style as the C programming language). When the left expression of an "and" operation evaluates to false, the whole operation will evaluate to *false*, so it is useless to evaluate the right expression. Similarly, when the left expression of an "or" operation evaluates to *true*, the whole operation will evaluate to true, and so there is no need to evaluate the right expression of the operation. By simply reordering the operations and their parameters, we can save execution time. The idea is to execute the simplest conditions first, while leaving the more complex ones to the end:

```
rule : { #burning } 1 { (0,0) > 333
  AND ( (0,0) < 573 OR $ti != 1.0 )
  AND (0,0) > #burning }
rule : { #burning }{ $ti := if($ti=
  1.0,time/100, $ti); } 1 { (0,0)>=
  573 AND #burning >= (0,0) }
rule : { #burning }{ $ti := time /
  100; } 1 { $ti = 1.0 AND (0,0) >=
  573 AND #burning < (0,0) }
```

This problem can also be solved using multiple ports to replace the extra plane. When we use multiple ports we do not need to store the values internally, but to transmit them through the ports. So, there is no need to set values, but just send them out though the corresponding port. In this case, two ports are declared: *temp* and *ti*. The port *temp* exports the cell's temperature, while the port *ti* exports the ignition time:

```
rule : { ~temp := #burning; } 1
  { (0,0)~temp>333 AND ( (0,0)~temp<
  573 OR (0,0)~ti!=1.0 ) AND
(0,0)~temp > #burning }
rule : { #burning } 1 { (0,0)> 333
  AND ( (0,0)< 573 OR $ti != 1.0)
  AND (0,0)>#burning }
rule : { #burning } { $ti := if($ti=
  1.0, time/100, $ti); } 1 { (0,0)>=
  573 AND #burning>=(0,0) }
rule : { #burning } { $ti := time /
  100; } 1 { $ti=1.0 AND (0,0)>=573
  AND #burning<(0,0) }
```

These two new versions of the model behave exactly the same as the original, but with clear gains in the modeling itself, which permits a user to describe more complex phenomena easily. Likewise, the different optimizations presented have allowed us to obtain gains in execution times of up to 40% just by reordering and factoring the rules to execute more efficiently using the CD++ evaluation mechanism. Figure 26 shows a 3D version of the execution results for this model using CD++/Maya. These visualizations can be easily expanded to include terrain and climate information, which would be useful for training, online visualization and decision-making.

Many of the models defined up to now have been based on a traditional definition for cellular models. We have shown how to use these basic definitions to create Cell-DEVS models, and how to expand them by using some of the basic facilities provided by CD++. Nevertheless, one of the fundamental advantages in the use of Cell-DEVS (which requires the most complex adaptation effort) is in the definition and use of the delay functions. We show an advanced Cell-DEVS fire model based on a well-known model for fire propagation in forests by Rothermel [56], in which we can see how to make use of the explicit time delay functions to improve model definition. This model uses environmental and vegetation conditions, and it computes the ratio of spread and intensity of fire. Three parameter groups determine the fire spread ratio: (i) vegetation type (caloric content, mineral content and density); (ii) fuel properties (the vegetation is classified according to its size); (iii) environmental parameters (wind speed, fuel humidity and field slope). When Rothermel's rules are applied to a given fuel model and environmental parameters, it can determine the spread ratio (i.e., the distance and direction the fire moves in a minute). The first step is to use the fuel model, the speed and direction of the wind, the terrain topography and the dimensions of the cellular space to obtain the spread ratio in every direction. For instance, Figure 27 shows the values obtained for a fuel model group number 9, a south-east wind of 24.135 km h^{-1} and a cell size of 15.24 × 15.24 m^2. Instead of using a time-based approach, the model uses the delay function to compute the fire spread, as seen in Figure 28.

The rules defining the local computing function are devoted to detecting the presence of fire in the eight neighboring cells. For instance, the first rule checks if the current cell is not burning: (0,0) = 0) and if the south-west neighbor has started to burn (0 < (1,–1). If this condition holds, the new value of the cell will be (1,–1) + (21.552615/17.967136), which is the time the fire will start in the cell. As the spread ratio is 17.967136 mpm and a cell has a diagonal of 21.552615 m, it will take (21.552615/17.967136) min for the fire to reach the cell once it has started in its south-west neighbor. Therefore, we use a delay of (21.552615/17.967136)*60,000 ms after which the present cell state will spread to the neighbors. The remaining rules represent a similar behavior for the re-

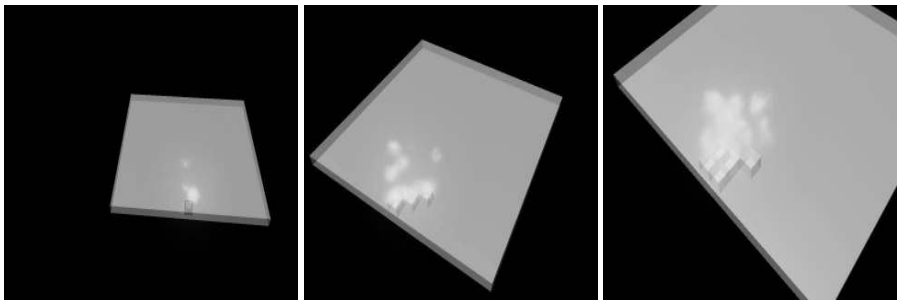


Figure 26. Visualization of the model in CD++/Maya

```
Wind direction = 45.000000 (bearing)                Wind speed = 8.045000 [kph] NFFL model = 1
Cell Width = 15.240000 [m] (E-W)                    Cell Height  = 15.240000 [m] (N-S)
Max. Spread = 17.967136 [mpm]
0° Spread = 5.106976 [mpm] Distance = 15.2400m      45° Spread = 17.967136 Distance = 21.552615
90° Spread = 5.106976    Distance = 15.240000       135° Spread = 1.872060 Distance = 21.552615
180° Spread = 1.146091   Distance = 15.240000       225° Spread = 0.987474 Distance = 21.552615
270° Spread = 1.146091   Distance = 15.240000       315° Spread = 1.872060 Distance = 21.552615
```

Figure 27. Parameter definition computed using Rothermel's model

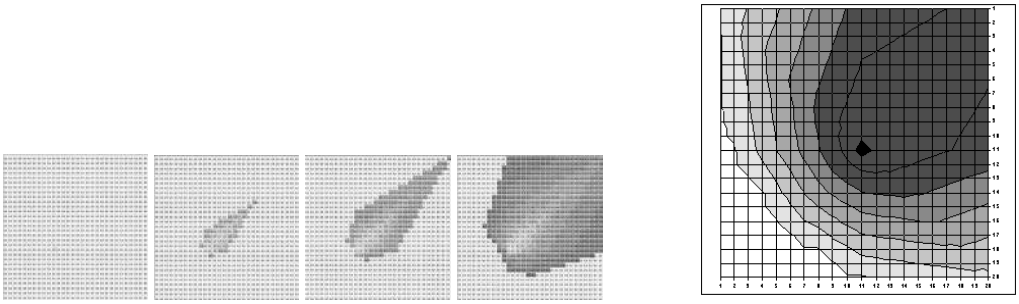
```
type : cell           dim : (20,20)
delay : inertial      border : nowrapped
neighbors : (-1,-1) (-1,0) (-1,1) (0,-1)  (0,0)  (0,1) (1,-1)  (1,0)  (1,1)
localtransition : FireBehavior

[FireBehavior]
rule : {(1,-1)+(21.552615/17.967136)} {(21.552615 / 17.967136)*60000} {(0,0)=0 and 0<(1,-1)}
rule : {(1,0)+(15.24/5.106976)} {(15.24 / 5.106976)*60000} {(0,0)=0 and 0<(1,0)}
rule : {(0,-1)+(15.24/5.106976)} {(15.24 / 5.106976)*60000} {(0,0)=0 and 0<(0,-1)}
rule : {(-1,-1)+(21.552615/1.872060)} {(21.552615 / 1.872060)*60000} {(0,0)=0 and 0<(-1,-1)}
rule : {(1,1)+(21.552615/1.872060)} {(21.552615 / 1.872060)*60000} {(0,0)=0 and 0<(1,1)}
rule : {(-1,0)+(15.24/1.146091)} {(15.24 / 1.146091)*60000} {(0,0)=0 and 0<(-1,0)}
rule : {(0,1)+(15.24/1.146091)} {(15.24 / 1.146091)*60000} {(0,0)=0 and 0<(0,1)}
rule : {(-1,1)+(21.552615/0.987474)} {(21.552615 / 0.987474)*60000} {(0,0)=0 and 0<(-1,1)}
rule : {(0,0)} 0 { t }
```

Figure 28. Definition of a fire forest model [48]

maining neighbors. The results of running this model are shown in Figure 29. As we can see, the burning time of a cell depends on the spread ratio in the direction of the burning cell. This value is used as the delay component for the rules. It is important to note that the cells are updated at different times, as set by a rule's delay component. This is a clear departure from the classical approach to CA, where all active cells are updated at the same time. A non-burning cell in the direction of the fire spread will be updated in a shorter period of time than a non-burning cell in the opposite direction. Another advantage is that the expression of a timing delay is done in a natural fashion, allowing the modeler to reduce the development time related with timing control programming.

Another advantage is that the complexity of this physical phenomenon is such that the inclusion of other external influences is difficult to consider. Cell-DEVS allows us to easily include new rules, allowing the evolvability of the model. For instance, we can use CD++ Lattice Translator to convert this model into a hexagonal version, as shown in Figure 30.

As we can see, we use a different notation to represent each one of the six neighbors ([1 ... 6], in counterclockwise direction starting at 0°). In the hexagonal lattice, the distance between two neighboring cells is the same in every direction, so we use a distance of 15.24 m for all of the rules. Using a triangular lattice, we obtain the rules shown in Figure 31.



(a) (b)

Figure 29. (a) Fire propagation results. (b) A 2-h period (each zone represents 20 min)

```
[FireBehavior]
rule: {[5]+( 15.24/13.68048533 )} {( 15.24 / 13.68048533 )*60000} {[0]=0 and [5]!=? and [5]>0}
rule: {[6]+( 15.24/5.106976 )} {( 15.24 / 5.106976 )*60000} {[0]=0 and [6]!=? and [6]>0}
rule: {[4]+( 15.24/2.95036533 )} {( 15.24 / 2.95036533 )*60000} {[0]=0 and [4]!=? and [4]>0}
rule: {[1]+( 15.24/1.630070333 )} {( 15.24 / 1.630070333 )*60000} {[0]=0 and [1]!=? and [1]>0}
rule: {[3]+( 15.24/1.146091 )} {( 15.24 / 1.146091 )*60000} {[0]=0 and [3]!=? and [3]>0}
rule: {[2]+( 15.24/1.040346333 )} {( 15.24 / 1.040346333 )*60000} {[0]=0 and [2]!=? and [2]>0}
```

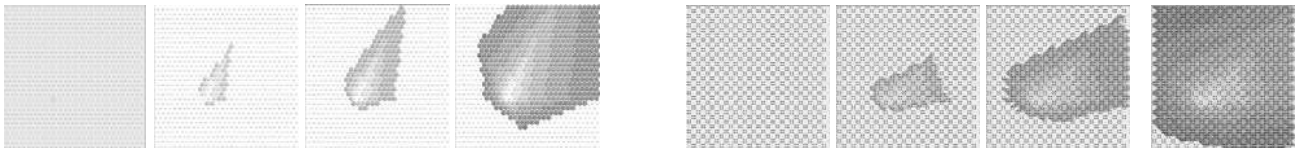
Figure 30. Rothermel's rules using hexagonal topology

```
[FireBehavior]
rule: {[3]+( 4.40 / 5.106976 )} {( 4.40 / 5.106976 )*60000} {[0]=0 and [3]!=? and [3]>0 and
odd(cellpos(0)+cellpos(1))}
rule: {[1]+( 4.40 / 2.95036 )} {( 4.40 / 2.950365 )*60000} {[0]=0 and [1]!=? and [1]>0 and
odd(cellpos(0)+cellpos(1))}
rule: {[2]+( 4.40 / 1.040346 )} {( 4.40 / 1.040346 )*60000} {[0]=0 and [2]!=? and [2]>0 and
odd(cellpos(0)+cellpos(1))}

rule: {[3]+( 4.40 / 8.57344 )} {( 4.40 / 8.57344)*60000} {[0]=0 and [3]!=? and [3]>0 and
even(cellpos(0)+cellpos(1)}
rule: {[2]+( 4.40 / 1.630070 )} {( 4.40 / 1.630070 )*60000} {[0]=0 and [2]!=? and [2]>0 and
even(cellpos(0)+cellpos(1)}
rule: {[1]+( 4.40 / 1.146091 )} {( 4.40 / 1.146091 )*60000} {[0]=0 and [1]!=? and [1]>0 and
even(cellpos(0)+cellpos(1)}
```

Figure 31. Rothermel's rules using triangular topology

In this case, there are six rules because we need to perform rules for even triangles and odd triangles. These models are translated into a square grid, as shown earlier in Figure 9. CD++ can display these topologies, as shown in Figure 32. The following figures show how to apply our new facilities in simulating Rothermel's model using different topologies. As we can see, the use of triangular and hexagonal topologies (which have much higher isotropy) produce very different results. Although triangular and hexagonal meshes provide better results in terms of the area covered, most researchers still use square lattices, as most GIS systems and existing modeling tools are based on square lattices only. The results presented in the Figure 32 provide a means of showing how environmental scientists can make use of these advanced topologies in order to improve the existing models using this basic feature.

Fire suppression can be easily implemented. In Figure 33, we show the implementation of a rain front moving to the south-east, extinguishing the fire on burning cells. This behavior is implemented in the rules shown in Figure 34, which were added to the previous model. Negative values define the effects of the rain. A cell whose value is –1 is a

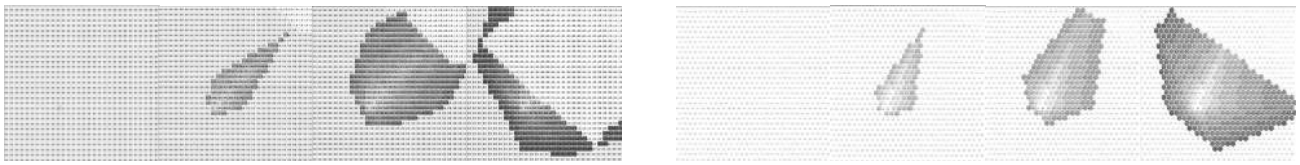


(a) (b)

Figure 32. Fire propagation with (a) hexagonal lattice and (b) triangular lattice

```
rule : -1 {60000*3}   {(0,0)=0 and ((-1,0)=-1 or (0,1)=-1 or (-1,0)=-2 or (0,1)=-2)}
rule : -2 {60000*3.5} {(0,0)>0 and ((-1,0)=-1 or (0,1)=-1 or (-1,0)=-2 or (0,1)=-2)
rule : -3 {60000*4.5} {(0,0)=-2}
rule : -4 {60000*5}   {(0,0)=-3}
```

Figure 33. Forest fire: rules defining rain



(a) (b)

Figure 34. Fire evolution with rain: (a) square lattice; (b) hexagonal mesh

wet cell where no fire was presented previously. A value of –2 or –3 indicates the cell was previously on fire and is now cooling down, and a value of –4 means the fire on that cell is extinguished. The first rule in Figure 33 defines rain spreading to the south-west. The second rule defines the cooling process on a burning cell, and the third and fourth rules represent an advance in the cooling process.

Figure 34 shows the execution of this model using two different topologies (square and hexagonal). The initial behavior is similar to that seen in Figures 29 and 32. We then observe the advance of rain, which cools the fire areas (light gray), and finally we can see how rain extinguishes fire areas. It is important to notice that if any of the cells are scheduled to start burning and become wet before the fire starts, these will not burn. This was easily defined by an inertial delay, which preempts any scheduled event if a new event from a neighboring cell occurs before the scheduled time and the present cell obtains a different value.

The extension in Figure 35 allows us to analyze fire suppression by firefighters. A negative value is still used for wet or cooling cells, and a positive value for burning cells, but the way in which the water is spread has been changed. In this case, firefighters move from north to south, spreading water on non-burning vegetation. Once they reach a burning cell, they hold their positions until the fire is extinguished, and then they move towards the south-west.

Figure 36 shows how firefighters spread coolant from north to south, and while the fire spreads (as in Figures 29 and 32), the firefighter zones cool down (light gray), while in some areas the fire has been extinguished.

6. Conclusion

We have shown how to apply the Cell-DEVS formalism and the CD++ tool to the construction of advanced models in the field of environmental science. Cell-DEVS allows us to describe physical and natural systems using an n-dimensional cell-based formalism. Input/output port definitions allow the definition of multiple interconnections between Cell-DEVS and DEVS models. Complex timing behavior for the cells in the space can be defined using very simple constructions. CD++ simplifies the construction of complex cellular models by allowing simple and intuitive model specification. The CD++ logic rules facilitate the debugging phase and, consequently, reduce development time. Complex model modifications can now be easily integrated into the models, even by a non-computer specialist. Using different examples of applications in the field, we

```
rule : -1 60000     {(0,0)=0 and (-1,0)=-1}
rule : -2 {60000*7} {(0,0)>0 and ((-1,1)=-1 or (-1,1)=-4) }
rule : -3 {60000*9} {(0,0)=-2}
rule : -4 {60000*9} {(0,0)=-3}
```

Figure 35. Rules defining firefighter behavior

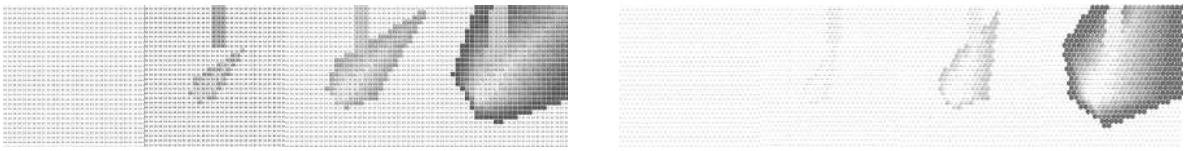


Figure 36. Fire evolution with firefighters: (a) square lattice; (b) hexagonal lattice

have shown how an environmental scientist can use the different facilities, showing how a team with minimum training can create complex applications without difficulties. The different examples presented here show the numerous new advanced facilities of CD++ and its application to the field of environmental sciences: execution of models with varied topologies, advanced rule definitions (with multiple state variables in each cell and multiple I/O ports to transfer information between submodels), integration of the results into advanced visualization environments, and seamless execution with high performance (including a parallel simulator and quantization algorithms). We have shown models on diffusion (viability of a population, pollution of the Venetian basin), hydrology and fire spreading, focusing on how our techniques can facilitate the task of the environmental modeler.

We have shown how these methods can improve the creation of ecological and environmental models, focusing on different aspects: how it allows the environmental specialists to apply their expertise while facilitating the change of paradigm, how to improve knowledge generation using a model-based approach, and how to execute with high performance. The experts can apply our advanced environment with traditional techniques, and then they can switch to the discrete-event based paradigm. If the local computing function is to be quantized, this requires a fundamental shift in thinking about the system as a whole, and also an alternative mechanism to collect experimental data and to define model equations: we must consider that instead of determining what value a dependent variable will have (its state) at a given time, we must determine at what time a dependent variable will enter a given state (therefore, the data collection must focus on the time for the state changes). The Cell-DEVS delay function provides a natural mechanism for implementing the quantization function. We have put into consideration two important issues: how to maintain the ability of CA to describe very complex systems using very simple rules (which is its main advantage), and how to bridge the gap between a continuous variable formalism. The independent simulation mechanisms permit these models to be executed interchangeably in single-processor, parallel or real-time simulators without any changes. This approach provides us with a technique that is easy to understand and to map into other existing techniques, while having the potential of evolving into more complex applications. The result is a set of models with the potential to evolve into more complex entities, which is unfeasible with the aforementioned approaches. We have also shown how different simulation engines can be activated at runtime, showing the use of quantization techniques for the execution of continuous variable Cell-DEVS. We have presented two different strategies for automatic updating of the quantum sizes in different cells. We have made important reductions in the error obtained, while maintaining the high speed of quantized DEVS models. Likewise, we can see that the introduction of QSS permits us to obtain a more controlled behavior, even for applications with cells executing with a nonlinear pattern.

7. Acknowledgments

This work has been partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canadian Foundation for Innovation, and the Ontario Innovation Fund. Different students collaborated in various stages of this project, including J. Ameghino, R. Chreyh, Esteban Fernandez Rojo, A. Khan, M. MacLeod and M. Polimeni.

8. References

[1] Toffoli, T., and N. Margolus. 1987. *Cellular Automata Machines*. Cambridge, MA: MIT Press.
[2] Wolfram, S. 2002. *A New Kind of Science*. Champaign, IL: Wolfram Media.
[3] Avolio, M. 2004. A cellular "blocks" model for large surface flows and

applications to lava flows. In *Proceedings of the 6th International Conference on Cellular Automata for Research and Industry*, Amsterdam, the Netherlands. *Lecture Notes in Computer Science* Vol. 3305. Berlin: Springer.

[4] Koh, A. 2002. Dramatic landscapes: Cellular automata modeling of landscape phenomena. Ph.D. Thesis, Monash University. Available online at http://www.csse.monash.edu.au/hons/se-projects/2002/cckoh1/dramatic_landscapes.pdf.

[5] El Yacoubi, S., A. El Jai, P. Jacewicz, and J. G. Pausas. 2003. LUCAS: an original tool for landscape modeling. *Environmental Modelling and Software* 18(5):429–37.

[6] Pukkala, T. 1988. Effect of spatial distribution of trees on the volume increment of a young Scots pine stand. *Silva Fennica* 22(1):1–17.

[7] Colasanti, R., R. Hunt, and L. Watrud. 2004. The use of cellular automata modeling approaches to understand potential impacts of genetically modified plants on plant communities. In *NKS 2004*, Boston, MA. Available online at http://www.wolframscience.com/conference/2004/presentations/HTMLLinks/index_20.html.

[8] Bagnoli, F. 2004. Sympatric speciation through assortative mating in a long-range cellular automaton. In *Proceedings of the 6th International Conference on Cellular Automata for Research and Industry*, Amsterdam, the Netherlands. *Lecture Notes in Computer Science*, Vol. 3305. Berlin: Springer.

[9] Auger, P., and B. Faivre. 1993. Cellular automata models applied to competition: The case of the sibling birds species of Hippolais in Burgundy. *Acta Oecologica* 14(6):781–806.

[10] Molofsky, J., and J. Bever. 2004. A new kind of ecology? *Bioscience* 54(5):440–6.

[11] Dzwinel, W. 2004. A cellular automata model of population infected by periodic plagues. In *Proceedings of the 6th International Conference on Cellular Automata for Research and Industry*, Amsterdam, the Netherlands. *Lecture Notes in Computer Science* Vol. 3305. Berlin: Springer.

[12] Darwen, P., and D. Green. 1995. Viability of populations in a landscape. *Ecological Modelling* 85(2–3):165–71.

[13] Bianchini, A., F. Indovina, and E. Rinaldi. 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 the 6th International Conference on Computers in Urban Planning and Urban Management*, Venice, Italy.

[14] Bandini, S., and G. Pavesi. 2002. Simulation of vegetable population dynamics based on cellular automata. In *Proceedings of the 5th International Conference on Cellular Automata for Research and Industry*, Geneva, Switzerland. *Lecture Notes in Computer Science* Vol. 2493. Berlin: Springer.

[15] Spencer, M. 1997. The effects of habitat size and energy on food web structure: An individual-based cellular automata model. *Ecological Modelling* 94:299–316.

[16] Balbi, J., P. Santoni, and J. Dupuy. 1999. Dynamic modelling of fire spread across a fuel bed. *International Journal of Wasteland Fire* 9:275–84.

[17] Barros, F., and G. L. Ball. 1998. Fire modelling using dynamic structure cellular automata. In *Proceedings of the 3rd International Conference on Forest Fire Research and 14th Conference on Fire and Forest Meteorology*, Luso, Portugal.

[18] Trunfio, G. 2004. Predicting wildfire spreading through a hexagonal cellular automata model. In *Proceedings of the 6th International Conference on Cellular Automata for Research and Industry*, Amsterdam, the Netherlands. *Lecture Notes in Computer Science* Vol. 3305. Berlin: Springer.

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

[20] Dunn, A. 2004. Modelling wildfire dynamics via interacting automata. In *Proceedings of the 6th International Conference on Cellular Automata for Research and Industry*, Amsterdam, the Netherlands. *Lecture Notes in Computer Science* Vol. 3305. Berlin: Springer.

[21] Muzy, A., E. Innocenti, A. Aiello, J.-F. Santucci, P. Santoni, and D. Hill. 2005. Modelling and simulation of ecological propagation processes: Application to fire spread. *Environmental Modelling and Software* 20:827–42.

[22] Wainer, G., and N. Giambiasi. 2001. Application of the Cell-DEVS paradigm for cell spaces modeling and simulation. *Simulation* 76(1):22–39.

[23] Zeigler, B., T. Kim, and H. Praehofer. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. New York: Academic Press.

[24] Chiari, F., M. Delhom, J.-B. Filippi, and J.-F. Santucci. 2000. A GIS based methodology for the modeling and the simulation of watersheds. In *Proceedings of the Advanced Technology Workshop (ATW) 2000 Conference*, Corsica, France.

[25] Zeigler, B., Y. Moon, D. Kim, and G. Ball. 1997. The DEVS environment for high-performance modeling and simulation. *IEEE Computational Science and Engineering* 4(3):61–71.

[26] Hill, D., T. Thibault, and P. Coquillard. 2002. Predicting invasive species expansion using GIS and simulation coupling. *Modeling and Simulation* 1(1):30–5.

[27] Hill, D., P. Coquillard, J. De Vaugelas, and A. Meinesz. 1998. An algorithmic model for invasive species application to Caulerpa taxifolia (Vahl) C. Agardh development in the North-Western Mediterranean Sea. *Ecological Modelling* 109:251–265.

[28] Muzy, A., G. Wainer, E. Innocenti, A. Aiello, and J.-F. Santucci. 2002. Dynamic and discrete quantization for simulation time improvement: fire spreading application using the CD++ tool. In *Proceedings of the 2002 Winter Simulation Conference*, San Diego, CA.

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

[30] Ntaimo, L., B. Khargharia, B. Zeigler, and M. Vasconcelos. 2004. Forest fire spread and suppression in DEVS. *Simulation* 80(10):479–500.

[31] Muzy, A., E. Innocenti, D. Hill, and J.-F. Santucci. 2003. Optimisation of cell spaces simulation for the modelling of fire spreading. In *Proceedings of the 36th Annual Simulation Symposium*, Orlando, FL.

[32] Vasconcelos, M., A. Gonçalves, and F. Barros. 2000. Dynamic maps. In *Proceedings of Artificial Intelligence, Simulation and Planning in High Autonomy Systems (AIS 2000)*, Tucson, AZ.

[33] Hu, X., and B. P. Zeigler. 2004. A high performance simulation engine for large-scale cellular DEVS models. *High Performance Computing Symposium (HPC'04), Advanced Simulation Technologies Conference*, Arlington, VA.

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

[35] Zeigler, B. 1998. DEVS. Theory of Quantization. DARPA Contract N6133997K-007, ECE Department, University of Arizona, Tucson, AZ.

[36] Kofman, E., and S. Junco. 2001. Quantized state systems: A DEVS approach for continuous system simulation. *Transactions of the SCS* 18(3):123–32.

[37] Wainer, G., and B. Zeigler. 2000. Experimental results of timed Cell-DEVS quantization, AI and simulation. In *Proceedings of Artificial Intelligence, Simulation and Planning in High Autonomy Systems (AIS 2000)*, Tucson, AZ.

[38] Wainer, G. 2004. Performance analysis of continuous Cell-DEVS models. In *Proceedings of High Performance Computing and Simulation (HPC&S) Conference; 18th European Simulation Multiconference*, Magdeburg, Germany.

[39] Giambiasi, N., B. Escude, and S. Ghosh. 2000. GDEVS: A generalized discrete event specification for accurate modeling of dynamic systems. *Transactions of the SCS* 17(3):120–34.

[40] Eckart, D. 1991. A cellular automata simulation system. *SIGPLAN Notices* 26(8):80–5.

[41] Wainer, G., and N. Giambiasi. 2001. Timed Cell-DEVS: Modelling and simulation of cell spaces. In *Discrete Event Modeling and Simulation: Enabling Future Technologies*, H. Sarjoughian, F. Cellier, eds. Berlin: Springer.
[42] Wainer, G., and N. Giambiasi. 2002. N-dimensional Cell-DEVS. *Discrete Events Systems: Theory and Applications*, Vol. 12, No. 1, pp. 135–57. Dordrecht: Kluwer.
[43] Wainer, G. 2002. CD++: A toolkit to define discrete-event models. *Software, Practice and Experience*, Vol. 32, No. 3, pp. 1261–306. New York: Wiley.
[44] Wainer, G., and W. Chen. 2003. A framework for remote execution and visualization of Cell-DEVS models. *Simulation* 79:626–47.
[45] Troccoli, A., and G. Wainer. 2003. Implementing parallel Cell-DEVS. In *Proceedings of the 36th IEEE/SCS Annual Simulation Symposium*, Orlando, FL.
[46] Muzy, A., G. Wainer, E. Innocenti, A. Aiello, and J.-F. Santucci. 2005. Cellular discrete-event modeling and simulation of fire spreading across a fuel bed. *Simulation* 81(2):103–17.
[47] López, A., and G. Wainer. 2004. Improved Cell-DEVS model definition in CD++. In *Proceedings of the 6th International Conference on Cellular Automata for Research and Industry*, Amsterdam, the Netherlands. *Lecture Notes in Computer Science* Vol. 3305. Berlin: Springer.
[48] Ameghino, J., A. Troccoli, and G. Wainer. 2001. Modeling and simulation of complex physical systems using Cell-DEVS. In *Proceedings of the 34th IEEE/SCS Annual Simulation Symposium*, Seattle, WA.
[49] Toffoli, T. 1994. Occam, Turing, von Neumann, Jaynes: How much can you get for how little? (A conceptual introduction to cellular automata). In *Proceedings of the International Conference on Cellular Automata for Research and Industry*, Rende, Italy.
[50] Khan, A., W. Venhola, G. Wainer, and M. Jemtrud. 2005. Advanced DEVS model visualization. In *Proceedings of the IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation*, Paris, France.
[51] Gutowitz, H. 1995. Cellular automata and the sciences of complexity. Parts I and II. *Complexity* 1(5):16 and 1(6):29.
[52] Ameghino, J., and G. Wainer. 2000. Application of the Cell-DEVS paradigm using CD++. In *Proceedings of the 32nd SCS Summer Computer Simulation Conference*, Vancouver, Canada.
[53] Gaylord, R., and K. Nishidate. 1996. *Modeling Nature: Cellular Automata Simulations with Mathematica*. Berlin: Springer.
[54] Ameghino, J., A. Troccoli, and G. Wainer. 2003. Applying Cell-DEVS models of complex systems. In *Proceedings of the Summer Simulation Multiconference*, Montreal, QC, Canada.
[55] Moon, Y., B. Zeigler, G. Ball, and D. P. Guertin. 1996. DEVS representation of spatially distributed systems: Validity, complexity reduction. *Proceedings of Artificial Intelligence, Simulation, and Planning in High Autonomy Systems*, La Jolla, CA, 288–96.
[56] Rothermel, R. C. 1972. A mathematical model for predicting fire spread in wasteland fuels. USDA Forestry Service Research Paper, INT-115.

Gabriel Wainer *received the M.Sc. (1993) and Ph.D. degrees (1998, with highest honors) of the Universidad de Buenos Aires, Argentina, and Université d'Aix-Marseille III, France. In July 2000, he joined the Department of Systems and Computer Engineering, Carleton University (Ottawa, ON, Canada), where he is now an Associate Professor. He is the author of two books and over 130 research articles. He is one of the investigators in Carleton University Centre for advanced Simulation and Visualization (V-Sim). He is Associate Editor of the Transactions of the SCS, and the International Journal of Simulation and Process Modeling. He is a chair of the DEVS standardization study group (SISO), and Associate Director of the Ottawa Center of The McLeod Institute of Simulation Sciences and chair of the Ottawa M&SNet. His current research interests is related with modelling methodologies and tools, parallel/distributed simulation and real-time systems.*