Applying Cellular Automata and DEVS Methodologies to Digital Games: A Survey

Gabriel Wainer, Qi Liu, Olivier Dalle and Bernard P. Zeigler

Simulation Gaming 2010 41: 796 originally published online 13 August 2010
DOI: 10.1177/1046878110378708

Simulation & Gaming
41(6) 796–823
© 2010 SAGE Publications
Reprints and permission: http://www.sagepub.com/journalsPermissions.nav
DOI: 10.1177/1046878110378708
http://sg.sagepub.com
SAGE

Applying Cellular Automata and DEVS Methodologies to Digital Games: A Survey

Gabriel Wainer[1], Qi Liu[1], Olivier Dalle[2,3], and Bernard P. Zeigler[4]

Abstract

Cellular automata were designed by John von Neumann in the 1940s, as a mathematical abstraction for modeling self-replicating algorithms. Since then, cellular automata have been widely studied theoretically and evolved into multiple variants. In the 1970s, Bernard P. Zeigler proposed a formalism rooted on systems theory principles, named DEVS (discrete-event systems specifications), which paved the way for component-based modeling and simulation and related methodologies. The purpose of this article is to survey how cellular automata and its variant, called cell-DEVS, may be used to implement computer simulations that can be used as digital serious games. The authors illustrate that implementation through some of the practical applications of such cellular automata. They show various serious game applications using real case studies: first, a simple bouncing ball and pinball game, a particle collision model, another on gossip propagation, and an application on human behavior at a metro station. Then, they show an application to social simulation using a voters game, a theoretical application (a model called Daisy World, which is derived from Gaia theory), and applications to physical phenomena such as a sandpile formation model or, finally, a three-dimensional model of a "virtual clay" that changes its shape when it is subject to pressure effects.

Keywords

CD++, cell-DEVS, cellular automata, cellular models, DEVS, digital games, discrete-event simulation, emergent behavior, modeling, parallel simulation, serious games, simulation, systems theory, visualization

[1]Carleton University, Ottawa, Ontario, Canada
[2]Université de Nice–Sophia Antipolis, Nice, France
[3]INRIA Sophia Antipolis (CRISAM), Sophia Antipolis, Nice, France
[4]University of Arizona, Tucson, AZ, USA

Corresponding Author:
Olivier Dalle, Dept. Informatique, UFR Sciences, 29 Ave. Joseph Vallot, 06000 Nice, France
Email: olivier.dalle@sophia.inria.fr

A cellular automaton is a very simple abstraction invented in the 1940s by John von Neumann to study self-reproducing systems. Since then, cellular automata (CA) have been widely used to reproduce and better understand the dynamics of complex phenomena found in the real world, especially when these phenomena only require very simple programming to be reproduced digitally. Life—as observed in the behavior of living entities—is one of the fascinating phenomena that have been extensively studied using such CA, for instance, by means of the famous GAME OF LIFE automaton. In its standard form, the GAME OF LIFE automaton is simply referred to as the B3/S23 rule by experts, which means that a new a cell is “born” (B) if it has exactly three neighbors, it stays alive (S) if it has two or three living neighbors, and it dies otherwise. Depending on the initial configuration of the game, this simple rule can result in a vast number of interesting phenomena and evolutionary patterns that have motivated serious studies as well as friendly applications (e.g., screen savers).

Despite their high ability to describe complex phenomena, however, CA still suffer from a low ability in supporting structural rules at various levels. In this article, we describe an extension of the CA that compensates for this missing structure by means of another simple but equally powerful mathematical abstraction called discrete-event systems specifications (DEVS). First, we give a comprehensive description of the resulting cell-DEVS formalism. Then, we demonstrate the ability of this formalism to express a wide range of serious games, by going through a series of examples of use case applications, ranging from the simplistic bouncing ball example to the more complex three-dimensional (3D) “virtual clay” model that changes its shape when it is subject to pressure effects. Each of these examples comes with a practical description of its implementation within the CD++ software simulator.

Overview

In recent years, computer games have been used in applications that go far beyond the traditional entertainment realm to support intuitive and critical training, efficient decision making, effective exploration of public policy, and cost-effective analysis of interpersonal communication and behavior, among others (Zyda, 2007). The technology exchange between computer games and simulation industries has created a new hybrid application field known as *serious games*, which, according to the Serious Games Initiative (Serious Games, 2008), refers to applying modeling and simulation (M&S) technologies for nonentertainment purposes in a variety of challenges facing the world today. Although several differences exist between M&S and serious games (Narayanasamy, Wong, Fung, & Rai, 2006), many already existing M&S methodologies can be applied in the serious game arena. Thus, for convenience, we use the current popular term *serious games* to refer to digital educational simulation/games.

In this article, we focus on the application of CA to serious games. Depicted in Figure 1, CA are formal models capable of describing dynamic systems as cell spaces, where complex behavior at the global (system) level emerges from local rules (Burks, 1970). CA were first proposed by von Neumann in the 1940s while working at Los

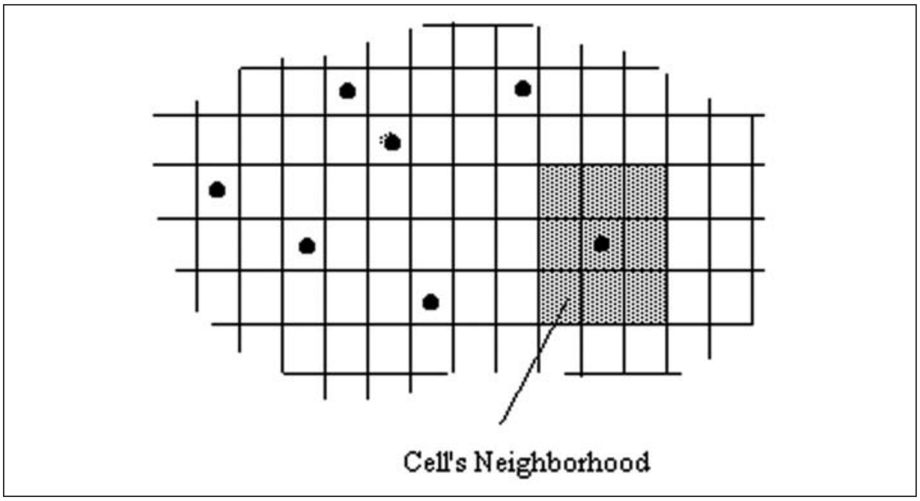


Figure 1. Sketch of a cellular automaton

Alamos on the problem of building self-replicating machines. Following a suggestion of his colleague Ulam, von Neumann designed a two-dimensional (2D) mathematical abstraction that was able to self-replicate algorithmically.

The fundamental idea behind this mathematical abstraction is to use a network of identical elements, named cells, each having its own individual state and sharing the same set of transition (evolutionary) rules. The transition rules decide the next state of a cell based on its current state and the current state of its neighboring cells in the network; rules are applied synchronously to every cell of the network. Hence, the resulting cellular automaton evolves step by step, and each such evolution is called a generation.

Practically, a CA simulation consists of computing a number of generations of a given CA, starting from an initial configuration. This initial configuration describes the state of each cell of the CA. Each cell in the cell space is evaluated synchronously, at discrete time steps, using the state values of neighboring cells.

A wide variety of studies relies on CA to model and simulate varied systems, thus leading to a new mathematical methodology. For example, CA have been used to study crowd behavior (Was, 2005; Yang, Fang, & Fan, 2003), urban traffic streams (Correia & Wehrle, 2006), biological systems and therapy (Reis, Santos, & Pinho, 2009; Slimi, EI Yacoubi, Dumonteil, & Gourbiere, 2009), market dynamics (Qiu, Kandhai, & Sloot, 2007), and so on. In spite of their widespread use and capabilities, however, CA models can be computationally inefficient. The synchronous nature of these models usually requires a discrete-time simulation approach, which can cause performance problems. The performance issues become even more prominent with the growing need to integrate CA simulations with hardware or human-in-the-loop applications (e.g., in serious games or advanced immersive virtual environments).

It has been shown that these performance problems can be mitigated using a discrete-event M&S approach, in which time is continuous but advances only when some events occur (Wainer & Giambiasi, 2001; Zeigler, Kim, & Praehofer, 2000). This is particularly useful in the case of cellular models, in which the number of inactive cells can be very large, where computing them only on the occurrence of an event can improve performance significantly.

Since the cell space is supposed to be of infinite size in each dimension, enumerating the states of each cell is a problem on its own. To ease this task, a particular state, called quiescent state, is distinguished, and cells in this state are called quiescent cells. The quiescent state has the following stability property: When the neighborhood of a quiescent cell exclusively contains quiescent cells, the state of that cell does not change in the next generation (i.e., it remains quiescent). Therefore, areas containing only quiescent cells can be neglected during the computation of the next generation, and only the nonquiescent cells need to be designated explicitly in a configuration.

In this work, we present an approach based on the DEVS formalism (Wainer & Giambiasi, 2001), a general modeling framework for DEVS. By decoupling the M&S concepts, DEVS offers several major benefits:

1. The same model can be executed with different DEVS-based simulators, allowing for portability and interoperability at a high level of abstraction;
2. A well-defined separation of concerns enables models and simulators to be verified independently and reused in later combinations with minimal reverification;
3. DEVS supports hierarchical composition of modular models, which can reduce the development and testing effort required for the construction of highly complex systems that are used commonly in serious games.

Wainer and Giambiasi (2001, 2002) proposed the cell-DEVS formalism to describe cell spaces as discrete-event models where each cell is represented by a DEVS basic model component to combine the advantages of CA and DEVS methodologies in a systematic way. Using a modular interface, each DEVS basic model can communicate with its neighboring cells in the cell space, as well as other models outside of the cell space. In addition, a set of well-defined timing constructions (with varied semantics) provides a convenient way for defining both temporal and spatial relations between model components. Both DEVS and cell-DEVS formalisms are implemented in CD++, an open-source M&S environment that has been used to solve a variety of complex applications (see, e.g., Farooq, Wainer, & Balya, 2006; Wainer 2006, 2009).

In addition, the DEVS-based M&S framework adopts a hierarchical software architecture that consists of five distinct layers, as illustrated in Figure 2. At the bottom of the architecture lies the hardware platform layer, which can include commodity PC workstations as well as high-performance parallel computing systems or single-board computers. In the center of the figure, the simulators layer offers the capability of

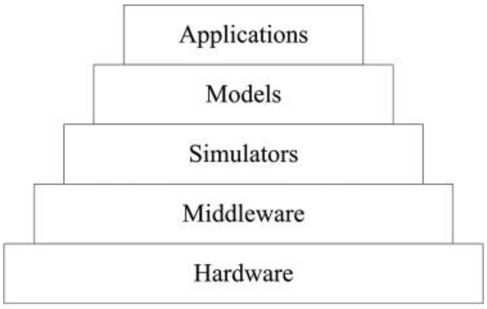


Figure 2. Hierarchical software architecture in DEVS-based M&S framework
NOTE: DEVS = discrete-event systems specifications; M&S = modeling and simulation.

running a simulation either in a standalone, real-time, or parallel and distributed modes. In between these two layers, the middleware layer offers standard alternatives for building the core of simulators, including, but not limited to, Message-Passing Interface (MPI), High-Level Architecture (HLA), Common Object Request Broker Architecture (CORBA), and Web Services (WS).

On the other side of the simulator layer, we find the Models layers. Such a separation of the M&S layers is a major attribute of the DEVS formalism. Thanks to this clear separation of concerns, models can be constructed and verified independently of the simulators and serve as reusable components for developing complex applications. While the *simulators* layer can be integrated with advanced database management systems as well as visualization and rendering facilities to provide much of the functionality of a game engine, the *models* layer incorporates many DEVS-based formalisms and techniques to allow for game content generation, complex behavior modeling, and game AI that are considered crucial components in serious games. The applications layer encompasses varied uses, including the serious games we introduce here. By taking advantage of this layered architecture, different technologies can be seamlessly integrated into the system with minimized impact on the other layers, allowing for the flexibility necessary to address many challenging problems in serious game development.

In the following sections, we focus on how to apply these cellular models to the domain of serious games development. We discuss the application potential of this methodology by providing different examples of serious games, by demonstrating how these methodologies address several technical challenges raised in developing these applications, and by showing how DEVS and cell-DEVS methodologies and tools can be used in this endeavor. We discuss some of the related work in this field and present several realistic examples to demonstrate the features and capabilities of such methodologies.

A DEVS Approach to the Development of Serious Games

Although serious games have been the topic of a number of studies (see, e.g., BinSubaih & Maddock, 2008; BinSubaih, Maddock, & Romano, 2006; Morgan, 2009; Prakash et al., 2009 Jain & McLean, 2008; Banerjee, Abukmail, & Kraeme, 2009) many technical issues still have to be resolved before their full potential can be realized in large-scale real-world applications. In this section, we review several key issues involved in serious game development and discuss how DEVS-based technologies could be used to address them.

Content Sharing and Interoperability

Many serious games involve a virtual environment where communication and content sharing between players serve as the central means of the gaming experience. This trend is also widely used in other gaming engines and online games, such as SECOND LIFE (Friedman, Steed, & Slater, 2007) and WORLD OF WARCRAFT (Ducheneaut, Yee, Nickell, & Moore, 2006). One of the most complex issues in integrating these applications is related to data sharing, given that content is often transmitted across different virtual environments provided by different vendors. This challenging problem requires nontrivial efforts (Merabti, El Rhalibi, Shaheed, Fergus, & Price, 2008), especially when we consider that current gaming engines are proprietary in nature and, in most cases, developed around specific gaming genres. This makes standardization and interoperability at the content level an even more difficult task (BinSubbaih et al., 2006).

Some studies suggest a middleware-oriented approach to alleviate this problem (e.g., Hsiao & Tuan, 2005). Nevertheless, without a solid common methodology applied at a higher level of abstraction, this middleware-based approach cannot, on its own, support describing and sharing game artifacts, content, and semantics in an integrated manner. Through DEVS, however, interoperability can be enhanced at the model (or content) level. Recent works, such as the DEVS Modeling Language (DEVSML; Mittal, Martin, & Zeigler, 2007), allow for semiautomatic translation between DEVS models and their XML representations, as well as simulation code generation from models defined in DEVSML. Efforts on DEVS standardization (e.g., Al-Zoubi & Wainer, 2008; Wainer et al., 2005; Wainer et al., in press; Zeigler, Mittal, & Hu, 2008) represent another step toward interoperability both at the semantic level (with emphasis on the standardization of model interfaces) and at the syntactic level (with emphasis on the standardization of simulation protocols).

Evolving Virtual World

To maintain player interest and provide a degree of continuity, new content and gaming scenarios need to be added incrementally into an existing virtual world. To this end, it is advantageous to create an evolving game environment based on dynamically extensible and semantically interoperable software architectures (Zyda, 2005). A common approach to this problem is to use a client/server architecture in which one updates the code at the server side (so that game content enhancement can be performed in a centralized and safe way). However, modifying an existing virtual environment at the code level requires significant programming and validation efforts and increases the cost of game development (BinSubaih et al., 2006).

On the other side of the spectrum, approaches that allow players to contribute new content to the virtual world (although at the expense of reduced simulation safety) also exist. Hence, a delicate balance between simulation safety and programming effort could be seen as the most effective approach.

The use of plug-in techniques is already a common practice in the game industry to extend game content and/or enhance application functionality (Kapolka, McGregor, & Capps, 2002). While this plug-in approach facilitates game development and allows game vendors to create new revenues when modules are sold, they do not provide a means of modifying existing modules. That said, plug-in techniques are suitable for adding new arenas to a virtual world but can add no new challenges that might increase player interest in modules that have already been sold. The hierarchical and modular approach of the DEVS methodology lends itself naturally to achieve this objective—in terms of simulation safety, module modification, player contribution, and development cost effectiveness.

First, model components that have already been validated in other similar game scenarios can be seamlessly incorporated into a DEVS-based virtual environment to achieve the well-defined and intended behavior without reverification of the whole system. This allows the use of an existing virtual world as a subcomponent of another larger system, reducing game development cost considerably. Second, rule-based high-level specification languages such as the one employed in CD++ for defining cell-DEVS models (Wainer, 2002, 2009) provide players with the means to specify model behavior and interaction in a simple way.

By separating the business logic from the application implementation, these rule-based tools allow for content enhancement without jeopardizing the underlying simulation system. Moreover, with the introduction of various graphical model specification techniques (e.g., Lavis, 2010; Wainer & Liu, 2009), defining new models and rules can be achieved efficiently. For instance, Figure 3 shows the integration of Distil Interactive serious game engine (Lavis, 2010) with a fire-spreading model in CD++.

Terrain Generation

One specific issue that attracts much interest from the serious games community is terrain generation. This is an important component in a variety of applications from landscape planning to advanced flight simulation (Livny, Sokolovsky, Grinshpoun, & El-Sana, 2008; Prakash et al., 2009). Two major issues involved in modeling and rendering large terrains are performance and quality. Traditionally, terrain generation has been achieved using different level-of-detail (LOD) algorithms (e.g., Dollner &

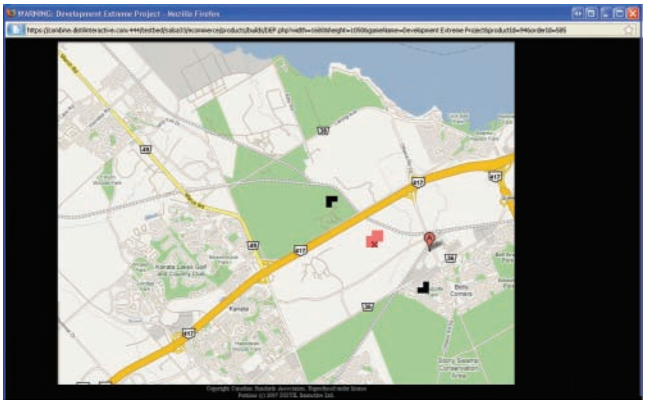


Figure 3. Integrating CD++ and Distil Interactive serious games platform

Buchholz, 2005; Levenberg, 2002), which dynamically reduce the number of triangles required to represent a terrain based on parameters such as distance from the current viewpoint and local height variations. However, LOD algorithms can be very computationally expensive and require a balanced approach to attain acceptable quality without degrading performance significantly. A similar issue also arises in modeling and rendering of large-scale cell spaces defined using cell-DEVS. Because of the discrete-event nature of DEVS-based systems, only active cells (i.e., those areas with changing characteristics such as elevation level or local bumpiness) are involved in the computation and rendering, which improves execution efficiency without a loss of accuracy. To enhance performance further, different techniques have been developed in the DEVS community, such as quantization (e.g., Kofman & Junco, 2001; MacLeod, Chreyh, & Wainer, 2006) and dynamic structure change (Hu, Zeigler, & Mittal, 2005). These techniques reduce the number of active cells while accounting for controllable errors. In turn, this allows the loading and simulating of a subset of the system that includes only active components. Thus, these techniques are especially suitable for generation of large terrain in serious games.

Performance and Reuse

By decoupling the model and simulation concepts, the DEVS and cell-DEVS frameworks allow the same model to be executed on different simulators, allowing for portability and interoperability at a high level of abstraction. This allows for the elimination of sequential execution constraints, while enabling the exploitation of increased parallelism. The cell-DEVS formalism combines CA with DEVS theory to describe *n*-dimensional cell spaces as discrete-event models, where each cell is defined as a DEVS basic model with explicit timing constructions. The synchronization mechanism used in Parallel and Distributed Simulation (PADS) generally falls into two categories: conservative approaches that strictly avoid violating the local causality constraint and optimistic approaches that dynamically detect causality errors and provide mechanisms to recover from them at runtime. Although optimistic approaches can exploit a higher degree of parallelism available in the simulation, the overhead of checkpointing and rollback operations incurred in optimistic simulations constitutes the primary bottleneck that may result in unstable and degraded performance.

DEVS tools, including CD++, make use of the DEVS decoupling of the M&S concepts. CD++, for instance, provides two separate frameworks: a modeling framework that allows users to define the behavior of atomic and coupled models and a simulation framework that creates an executive entity for each component in the model hierarchy to carry out the simulation (using sequential, parallel, or real-time algorithms). This independency between M&S mechanisms permits DEVS models to be executed interchangeably in single-processor, parallel, or real-time engines without any changes.

Likewise, the use of DEVS as the basic formal specification mechanism enables one to define interactions with models in other formalisms (i.e., PDE, CA, finite elements, finite differences, etc.), meaning that integration between these formalisms could then be easily used to defining complex models with diverse methods (Mayer & Sarjoughian, 2009; Wainer, 2009). This approach provides evolvability of the models through a technique that is easy to understand and can be combined with other techniques. DEVS enables high performance execution. The discrete-event nature of DEVS can improve execution times in several orders of magnitude, including the application for continuous models.

Finally, DEVS and cell-DEVS (as CA as well) provide the advantage of a formal approach: a formal conceptual model can be validated and improves the error detection process, while reducing testing time (thus improving the quality and development costs of a simulation). DEVS allows for the translation of formal specifications into executable models, facilitating the verification of the simulator and validation against the real system as well as the controlled investigation of performance through alternative models and simulators (Shiginah, 2006).

Applying DEVS and CA in Serious Games

As shown previously, cellular models and DEVS can provide enhanced capabilities in the development of serious games applications. In this section, we give different examples in this field. The idea of this section is to show how CA and DEVS models can be used to create varied serious games samples, focusing on the methods and techniques employed in developing those models.

These methods and techniques may be combined and integrated in various simulation engines. For instance, Figure 3 shows an integration of CD++ cell-DEVS models with Distil Interactive's engine and Figure 4 shows a fire-spreading simulation example in which the fire-spreading scenario running in CD++ is integrated with a Google Map environment (Harzallah, Michel, Liu, & Wainer, 2008). Likewise, Figure 5 shows an integration of a CD++ simulation with BLENDER (Wainer & Liu, 2009), a game

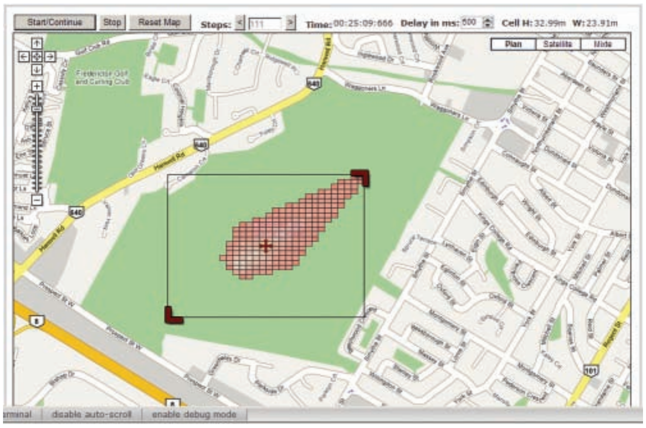


Figure 4. Integrating a fire-spreading cellular model with Google Maps

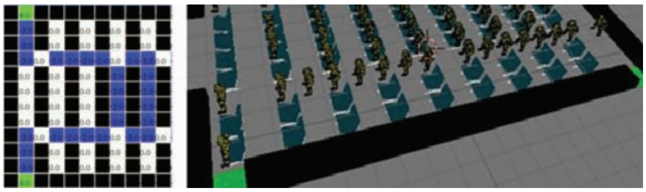


Figure 5. Airbus aircraft evacuation in 2D and 3D

that shows a visual model of an emergency evacuation (used to demonstrate the effect of delays caused by passenger's hesitation at the exit door). As we can see, the 2D model on the left can be also visualized in a 3D environment without difficulty.

The issues to be discussed are the following: How do we create the fire model in the figures above? How is the behavior of the game (fire spreading, plane evacuation) defined? If a developer needs to upgrade or improve the model: How is it done? In this section, we will introduce different cellular models applications with the goal of answering the questions above. These varied models have been developed and simulated with the CD++ environment, and they illustrate the potential of applying CA- and DEVS-based methodologies in serious game development. All these examples are available to the public and can be downloaded from http://cell-devs.sce.carleton.ca/.

A Simple Model of Bouncing Balls

The CD++ environment includes facilities to build DEVS and Cell-DEVS models. Cellular models are defined using a built-in language based on the formal specifications

```
[top]
type : cell
dim : (20,20)
delay : transport
border : nowrapped
neighbors : (-1,-1)  (-1,1)  (0,0)  (1,-1)  (1,1)
localtransition : move-rule

zone  : UpperLeft-rule {  (0,0)  } %  zone  definition  for  northwest  corner
cell
...                                % Other similar rules skipped here

[move-rule]
rule : 1 100 { (-1,-1) = 1 }       %move towards southeast
rule : 2 100 { (1,-1) = 2 }        %move towards northeast
rule : 3 100 { (-1,1) = 3 }        %move towards southwest
rule : 4 100 { (1,1) = 4 }         %move towards northwest
rule : 0 100 { t }
...
```

Figure 6. Bouncing-ball model definition in CD++

of Cell-DEVS, which includes the definition of the size and dimension of the cell space, the shape of the neighborhood, and the type of the cells that represent the borders of the bouncing area. The cell's local computing function is defined using a set of rules in the following format.

```
POSTCONDITION    DELAY    { PRECONDITION }
```

That is, when the PRECONDITION is met, the state of the cell will change to the designated POSTCONDITION after the duration specified by DELAY. If the precondition is not met, then the next rule is evaluated until a rule is satisfied or no more rules are available. As discussed in (Wainer, 2006; Wainer & Giambiasi, 2001), this formalism allows for improved security and lowered development cost for the simulations.

In this section, we show the basic definitions of Cell-DEVS models by defining the behavior of balls bouncing within a container. The model is specified by a 20 × 20 cell space using the CD++ specification language. The model defines nine sets of rules to control the movement of balls, including four sets of rules for the corner and border cells each as well as a set of rules for the rest of the cell space. Figure 6 shows a snippet of the model definition in CD++.

The model starts by defining a cell space of 20 × 20 cells, in which we use special rules for the border cells (nowrapped). Each cell has a neighborhood that includes the surrounding cells at the four possible 2D directions. Each cell in the model has a neighborhood definition (using the relative positions to the central cell itself) that includes the central cell (0, 0), northwest (NW) cell (−1, −1), northeast (NE) cell (−1, 1), southwest (SW) cell (1, −1), and southeast (SE) cell (1, 1). Each set of rules in the figure is applied to its corresponding zone defined in the cell space. For instance, the

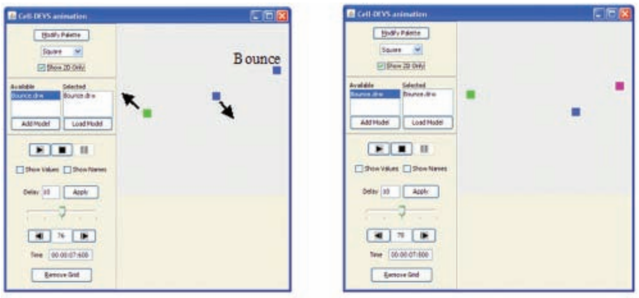


Figure 7. Simulation of bouncing balls in CD++

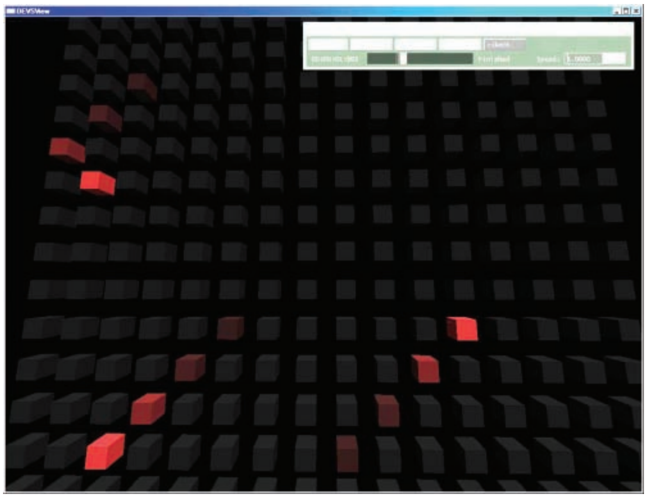


Figure 8. 3D visualization of bouncing balls in CD++

northwest corner cell of the cell space is defined as a zone of a single cell with an associated local transition rule called UpperLeft-rule. Four possible moving directions are encoded by integer cell values (i.e., 1 for SE, 2 for NE, 3 for SW, and 4 for NW). A ball changing its direction when it bounces into a wall or other balls is simply modeled by a change of cell value during rule evaluation.

Figure 7 shows a snapshot of the bouncing ball simulation results. As we can see, three balls are moving in different directions, and one of them bounces against the wall. The simulation in Figure 8 shows three balls contained in a 3D grid, which bounce on the walls, and a 3D visualization during playback, built using CD++ visualization engines (Wainer & Liu, 2009).

This model was extended to define a 2D PINBALL GAME. This 19 × 19 cellular model simulates a single pinball bouncing around in a container with predefined obstacles (unlike the pinball game, the action of the paddles is not included in this example). The cell space uses a Moore's neighborhood of range 1 (i.e., the origin cell and its eight near neighbors). In this case, we extended the model's behavior to include obstacles. Similar to the previous example, the pinball's movement is controlled by a set of local transition rules, which now have to detect obstacles. For instance, the following set of rules defines the behavior of a ball moving from the SE to the NE:

```
rule: 2 100 { (0, 0) = 0 and (1, 0) = 2 }
rule: 5 100 { (0, 0) = 0 and (0, -1)= 2 and (-1, -1)= 9}
rule: 7 100 { (0, 0) = 2 and (-1, 0)= 9 and (0, 1) = 9 }
```

As we can see, the rules now include a new state (value = 9) to define the obstacles. In this case, the first rule states that, if at present the cell is empty and a ball is coming from the S with direction N (value = 2), then, the current cell will become occupied by that ball after a delay of 100 ms. The second rule checks the cell to the W, in order to see if it contains a ball with direction N (defined by the expression (0, −1) = 2) when the cell to the NW is blocked by an obstacle (i.e., (−1, −1) = 9). In this case, the ball bounces against the obstacle, and moves to the current cell, changing direction (i.e., value = 5, which represents direction WE). The third rule represents the case in which the current cell contains a ball moving N, and the N cell is blocked, as well as the E cell. In this case, the ball bounces to the W (i.e., value = 7, which represents the direction EW).

As we can see, modifying these rules can be done on the fly, and adding behaviors that are more advanced is straightforward. The resulting model can be executed in different engines (standalone, distributed, parallel, or real-time) without any modifications. Figure 9 shows a snapshot of the simulation results.

Particle Collision

This model uses a lattice gas to define a 2D cellular model of the collision of particles. This kind of model can be used to study the particle collision of gas fluids, where different types of fluids are mixed together in a container. Each cell uses a double value that consists of an integer part and a fraction part. The integer part represents four possible moving directions of a particle (1:E, 2:N, 3:W, 4: S), whereas the fraction part represents the purity of the fluid. The collision of two particles modifies not only their direction but also the degree of purity of the corresponding fluids. The following figure shows a sample rule for this model, in which we can see that if a particle is moving in direction E and another one in front of it is moving in direction W, a new particle is created in direction N (and their purities combined). The second rule shows the symmetric behavior to create the second particle moving to the S.

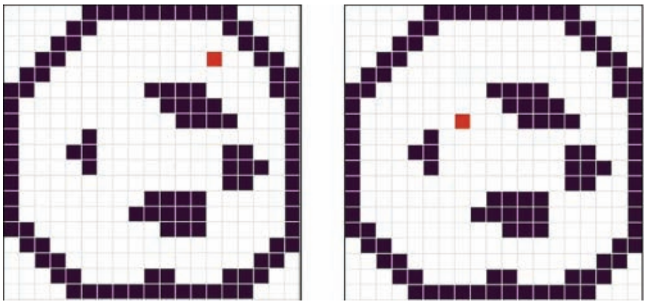
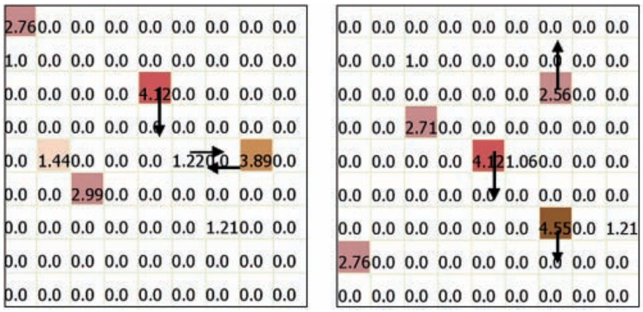


Figure 9. Pinball game simulation results



2.76	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	4.12	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.44	0.0	0.0	0.0	1.22	0.0	3.89	0.0
0.0	0.0	2.99	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	1.21	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	2.56	0.0	0.0
0.0	0.0	2.71	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	4.12	1.06	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	4.55	0.0	1.21
2.76	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 10. A particle collision cellular model

```
rule: { 2 + ( fractional((1, 0)) + fractional((1, 1)) )/2 }
      100 { trunc((1, 0)) = 1 and trunc((1, 1)) = 3 }
rule: { 4 + ( fractional((-1, -1)) + fractional((-1, 0)) )/2 }
      100 { trunc((-1, -1)) = 1 and trunc((-1, 0)) = 3 }
```

The simulation results are shown in Figure 10. At simulated Time 100, we can see a particle moving with direction E (value 1.0 in the second row) and another moving to the S (value 4.12 in the third row). We can also see a collision (in Row 5), where a cell moving to the E with purity 22 collides with another one with purity 89. As a result of the collision, the particles change direction and purity (2.56 and 4.55, respectively, to the N with purity 56 and to the S with purity 55).

Gossip Propagation

This model (based on an original definition by Klaus Troitzsch at the University of Koblenz) represents the propagation of a piece of gossip (or information) by a group of people, using the flow of information from a single source to a group of sinks (i.e., receivers). The information only propagates to interested parties from someone who has already heard the gossip. On the other hand, a person who knows the information may also choose to not spread it to others. This kind of information flow can be used as a means of interest management in serious games to express how interest and influence of a party in a virtual world is exerted on other participants.

The Cell-DEVS model of this gossip propagation uses two bidimensional square grids. The bottom layer grid represents the people, who will be either active or passive based on whether or not they know the gossip. The second layer is used to hold the probabilities of spreading the gossip. The cell neighborhood can be either a Moore or a von Neumann neighborhood (i.e., the cells to the N/S/E/W). Each cell in the inverse neighborhood (i.e., those cells obtained by negating the neighborhood values) has a random chance of being told the gossip from a cell that knows it. This prevents the model from evolving the same way in every execution. This random number is obtained from the layer above.

This model is defined by the three following rules for cells:

1. If a cell is passive, and cells are active in its inverse neighborhood, the cell may become active.
2. If a cell is active, it will remain active.
3. A cell that knows the gossip is unlikely to forget it.

Some of the rules are presented in Figure 11.

As we can see in Figure 11, the bottom layer (i.e., the one in which the precondition *cellpos (2) = 0* is true) represents the people (with or without information). The top layer (i.e., the one where *cellpos (2) = 1*) holds the probabilities of spreading the information for each person. Different types of neighborhoods can be used to define the interrelationship among the people (and it can be easily changed by modifying the *neighbors* clause in the model). Initially, the gossip holder is located at the center of the cell space. The model is tested with varying probabilities for passing on the gossip or forgetting it. Figure 12 shows an animation of the simulation results where an orange (gray) cell stands for a person who possesses the information.

The tests run were with probabilities of the gossip being spread at 25%, 50%, and 75%. In all these cases, the forgetting probability was 10%. The model was also tested with a probability of 50% and forgetting probability of 5%. When analyzing the results of this game, the change in probability of forgetting the gossip showed little effect on the rest of the model (as long as the probability of passing the gossip was greater than forgetting it, any cells forgetting the gossip would quickly learn it again). The model appears to spread outward for all the probability levels tested (with smaller probability of spreading the gossip, the model spreads slower). The spread pattern looks slightly different with different neighborhoods, as one can expect.

```
[gossip]
type  : cell                    dim : (10, 10, 2)
delay : transport            border : wrapped
neighbors : (-1,0,0)  (0,-1,0)  (0,0,0)   (0,1,0)   (1,0,0)   (-1,0,1)
neighbors : (0,-1,1)  (0,0,1)   (0,1,1)   (1,0,1)
zone : people { (0,0,0)..(9,9,0) }
zone : problty { (0,0,1)..(9,9,1) }

[people]
rule : { if ((if(((-1,0,0) = 1 and (-1,0,1) > 0.25),1,0) +
             if(((0,-1,0) = 1 and (0,-1,1) > 0.25),1,0) +
             if(((0,1,0) = 1 and (0,1,1) > 0.25),1,0) +
             if(((1,0,0) = 1 and (1,0,1) > 0.25),1,0)) >= 1, 1, 0) }
        1000
        { cellPos(2) = 0 and (0,0,0) = 0 and stateCount(1) > 0 }
rule : { if (random < 0.10,0,1) }
       1000
       { cellPos(2) = 0 and (0,0,0) = 1 }

[problty]
rule : { random } 1000 { cellPos(2) = 1 and (0,0,1) = 1 }
```

Figure 11. Cell-DEVS definition of the gossip propagation model
NOTE: DEVS = discrete-event systems specifications.

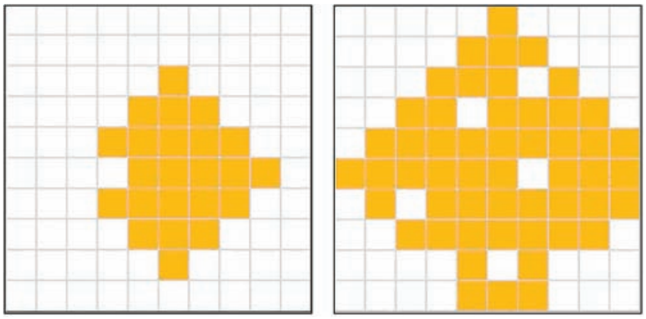


Figure 12. Gossip propagation with Von Neumann's neighborhood and information loss probability of 5%

Human Behavior at a Metro Station

This model represents the movement of persons waiting for a train in a metro station (Wainer, 2009). The following figure shows the simulation results of a CA that mimics people arriving at a train station. Two cells located on the right side of each slide represent the platform entrance (yellow cells), and people move from those cells to the

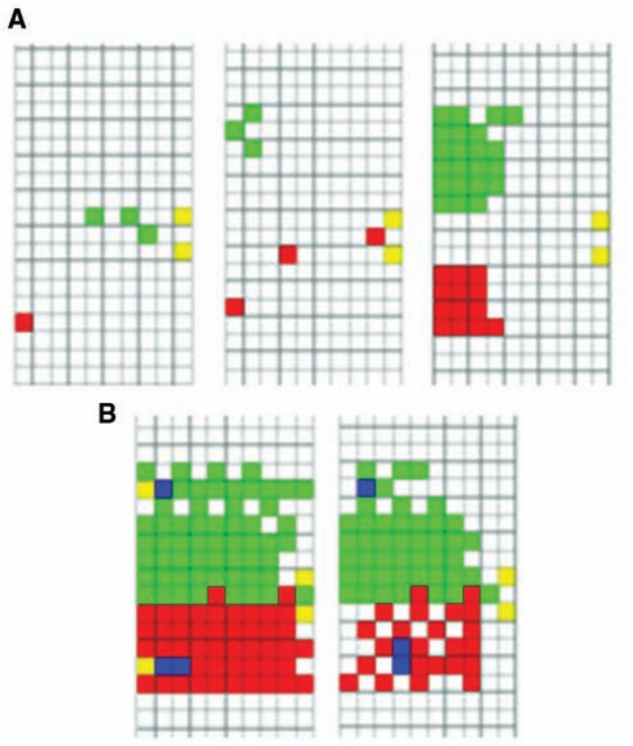


Figure 13. Simulation of human behavior at a metro: (A) standard pattern; (B) crowded station as a cellular model

left, where the train arrives. The green (light gray) cells represent people who want to get in the train using the door placed in the upper part of the grid. The red (dark gray) cells represent people who want to get in the train using the door placed in the lower part of the grid. As more people arrive, they tend to concentrate at the two boarding doors. As shown in the rightmost slide in Figure 13A, two groups of people are formed around the boarding doors at the border of the platform waiting for the doors to open.

When the train arrives, the people leaving the train have a higher priority than those trying to get into the train. Therefore, if someone is leaving the train, the people waiting at the platform start to move to let them leave. In the same way, collision detection is used such that if people try to get into the train at the same time, only one of them gets the access. Figure 13B shows the case in which the metro station is crowded, and in this case, we can see that when a train leaves, the train station still has a large number of people waiting for the next train to come. As we can see, only a few people could get into the train in the scenario, leaving the station still crowded.

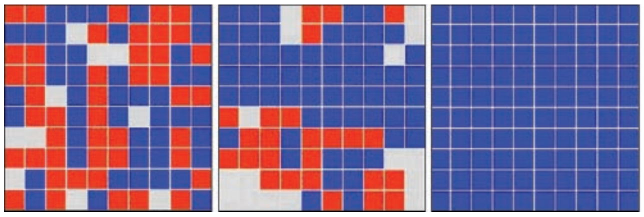


Figure 14. Modeling voter behavior as a cellular model

Voters

In this game, we represent how voters' political preferences are influenced by others in a constrained environment using a 2D cellular model. The voters can either stick to their current political preferences or change their preferences based on the influence of their neighbors. Each cell stands for a voter who is under the influence of four neighbors as defined by a Von Neumann's neighborhood. The rules that control future political preference have two parts:

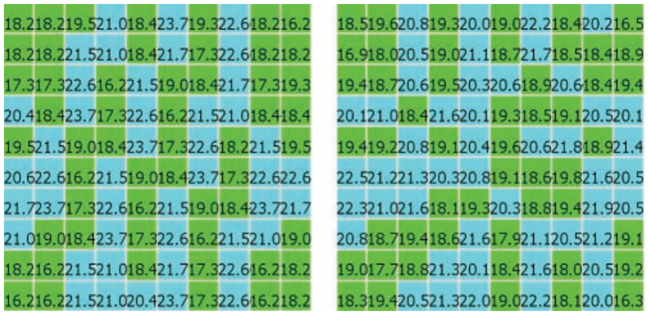
(a) voters stick to their previous political preference according to a global probability parameter (e.g., 50%) and
(b) if the voter decides to change the political preference, one of the four adjacent neighbors is picked randomly with equal probability (i.e., 25%) and this neighbor's current preference value is assigned to the voter.

In this example, each voter has a 50% chance to keep his or her previous preference and a 12.5% chance for a change.

The model begins with a board of randomly generated preferences, shown in Figure 14A. In this example, 55% prefer the Blue Party (dark cells), 35% prefer the Red Party (light gray cells), and 10% are independent voters (white cells). Figure 14 shows the convergence of opinion. After 200 votes, the minor political preference (i.e., those with no opinion) disappear, and the major preference (i.e., Blue Party) prevails.

The Daisy World

The Daisy World model is a game that can be used to study the principle of self-regulation and the effect of planetary life on climate change. An imaginary planet, called the Daisy World, is populated solely by black and white daisies (Wu, Wu, & Wainer, 2004; Lovelock, J., 1992, September). Black daisies warm up the planet by absorbing heat, while white daisies cool it by reflecting heat. As the planet's sun warms up, the daisies maintain a moderate temperature by altering the balance between the black and white. Eventually, as the sun gets too hot, no daisy can survive and all life dies out on the planet.



```
18.218.219.521.018.423.719.322.618.216.2
18.218.221.521.018.421.717.322.618.218.2
17.317.322.616.221.519.018.421.717.319.3
20.418.423.717.322.616.221.521.018.418.4
19.521.519.018.423.717.322.618.221.519.5
20.622.616.221.519.018.423.717.322.622.6
21.723.717.322.616.221.519.018.423.721.7
21.019.018.423.717.322.616.221.521.019.0
18.216.221.521.018.421.717.322.616.218.2
16.216.221.521.020.423.717.322.616.218.2
```

```
18.519.620.819.320.019.022.218.420.216.5
16.918.020.519.021.118.721.718.518.418.9
19.418.720.619.520.320.618.920.618.419.4
20.121.018.421.620.119.318.519.120.520.1
19.419.220.819.120.419.620.621.818.921.4
22.521.221.320.320.819.118.619.821.620.5
22.321.021.618.119.320.318.819.421.920.5
20.818.719.418.621.617.921.120.521.219.1
19.017.718.821.320.118.421.618.020.519.2
18.319.420.521.322.019.022.218.120.016.3
```

Figure 15. A Daisy World define as a cellular model

The Daisy rules are based on the following assumptions:

1. Black daisies live in temperatures below 25 degrees, and they absorb heat and raise the ambient temperature by 1 degree per time unit;
2. White daisies live in temperatures above 15 degrees, and they reflect heat and lower the ambient temperature by 1 degree per time unit.

The model uses a Moore's neighborhood of range 1 to calculate a cell's current temperature, which is the average temperature of the neighboring cells. When the temperature in a cell is between 15 and 25 degrees, the daisy could be either black or white. However, if the temperature drops below 15 degrees, then the daisy in that cell must be black; and if the temperature rises above 25 degrees, then the daisy must be white. It is possible to infer the daisy color for temperatures between 15 and 25 degrees by tracking the color changes below 15 degrees and above 25 degrees (i.e., if the temperature in a cell drops below 15 degrees, then the daisy will be black and it will remain black as long as the temperature does not rise past 25 degrees. Conversely, if the temperature in a cell rises above 25 degrees, then the daisy will be white and it will remain white as long as the temperature does not drop below 15 degrees).

The simulation results in Figure 15 show a self-regulated Daisy World (dark cells represent black daises, light cells represent white daisies).

Sandpile Modeling

CA models have been used to study the formation of sand piles in the event of landslides caused by major earthquakes (D'Ambrosio, D. et all., 2003; Malamud, B. D., & Turcotte, D. L.; 2000). Here, we show the definition of one of such CA proposed in (Malamud and Turcotte, 2000), built as a coupled model consisting of a cellular model component (representing a sandpile) and a DEVS atomic component that serves as a sand generator.

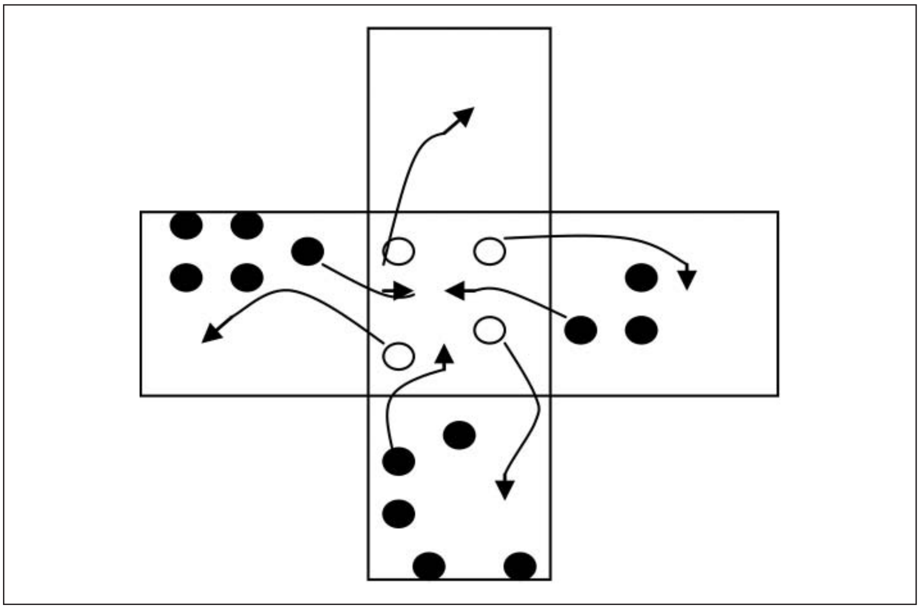


Figure 16. Redistribution of sand particles in the cell space

The following rules are applied to the sandpile model:

1. Initially, the cell space is loaded with some sand particles in some of its cells (while others are empty).
2. Sand particles generated by the DEVS component arrive randomly in the central cell in the cell space.
3. Each cell has a maximum capacity of four particles. When a cell reaches this capacity, a redistribution operation will empty the cell and add one particle to each of the nondiagonal neighboring cell (consequently, a cascaded redistribution might occur in the neighboring cells).
4. Cells on the border of the cell space will lose particles to the environment at sides when their capacity is reached.
5. The cascaded redistribution could result in an avalanche in the cell space and the intensity of the avalanche can be measured either by number of cells participating in the redistribution, or by the number of particles lost from border cells.

Cells can have one integer value ranging from 0 to 7. A value of 0 represents an empty cell; values 1 to 3 represent the number of sand particles. When the value of a cell goes beyond 3, redistribution occurs. A cell with value 4 becomes empty and redistributes all four particles to its neighboring cells. Similarly, a cell with value 5, 6, and 7 will distribute four particles to the neighbors and hold the remaining particles. This redistribution of particles is illustrated in Figure 16, while the simulation results are animated in Figure 17.

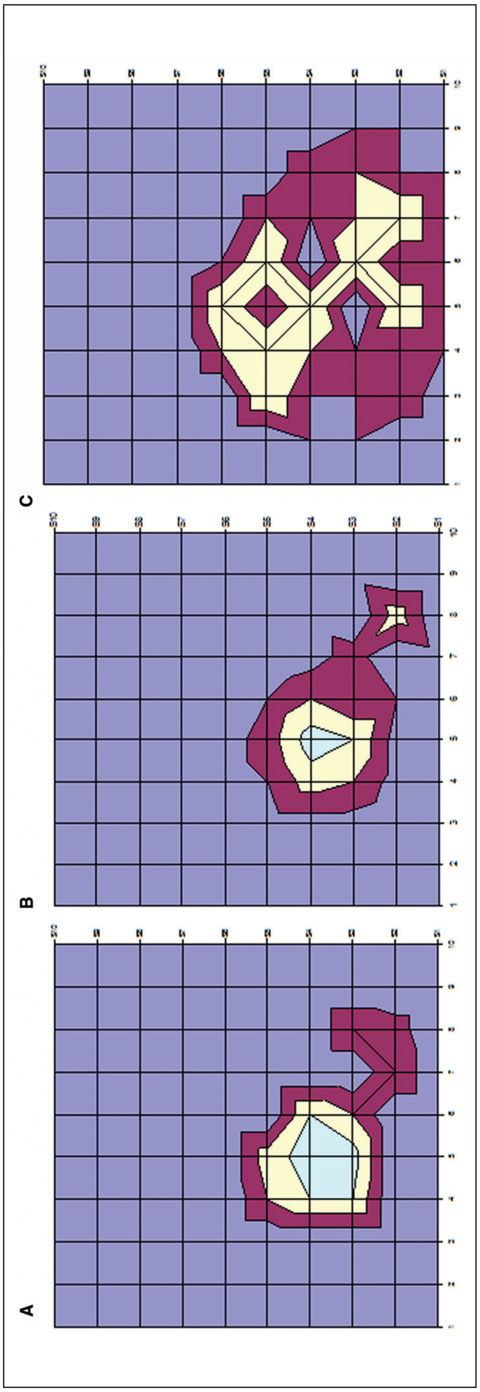


Figure 17. Sandpile formation

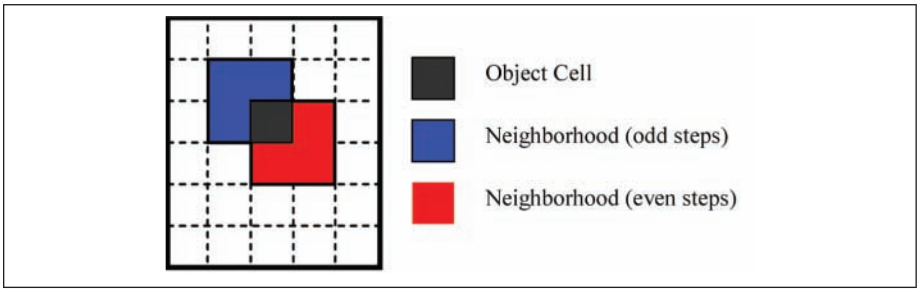


Figure 18. Margolus neighborhood definition

3D Free-Form Shape Modeling

3D free-form shape modeling has been used in various applications such as computer-aided design systems and mechanical engineering. However, modeling free-form shapes in 3D space involves considerable computation based on simplified physical laws. In this section, we show how to apply cellular models for the compression (from outside) and deformation (from inside) of a virtual clay model (Wu et al., 2004).

The deformation process is the result of clay transportation from high-density areas to low-density areas. In our cellular model, each cell has a positive value corresponding to the mass of clay contained in the cell. A cell with value above a predefined threshold will transfer clay to its neighboring cells. The model defines a Margolus neighborhood for the cells. Figure 18 shows the Margolus neighborhood definition in 2D and 3D spaces.

As we can see, the 2D Margolus neighborhood is a block that includes the nearest four cells. Cells belonging to the same block will perform state transitions together at each step. In addition, block boundaries change depending on whether the transition is performed at an odd or even step. Hence, the neighborhood of each cell alternates between its odd and even configuration at every transition step.

Each cell has a binary state reflecting whether it has a mass of clay above or under the threshold, creating four different block patterns of clay transitions in the 2D neighborhood, as illustrated in Figure 19. The block patterns are more complex in the 3D Margolus neighborhood, including 21 different transition rules.

As we can see in Figure 19, the deformation process is based on a push operation, where clay is transferred from a cell into the adjacent cell along the push direction. The surface of clay can be pushed at most one cell in depth per step. When some cells become overloaded as the result of a push operation, clay is redistributed based on the transition rules. The process repeats until no cell in the space is overloaded, reaching a steady state in the cell space.

The compression and deformation processes are realized as two distinct stages in the cellular model, as follows:

1. Compression: moving plate exerts a push operation on some cells, which transfer clay particles to the adjacent cells along the push direction;

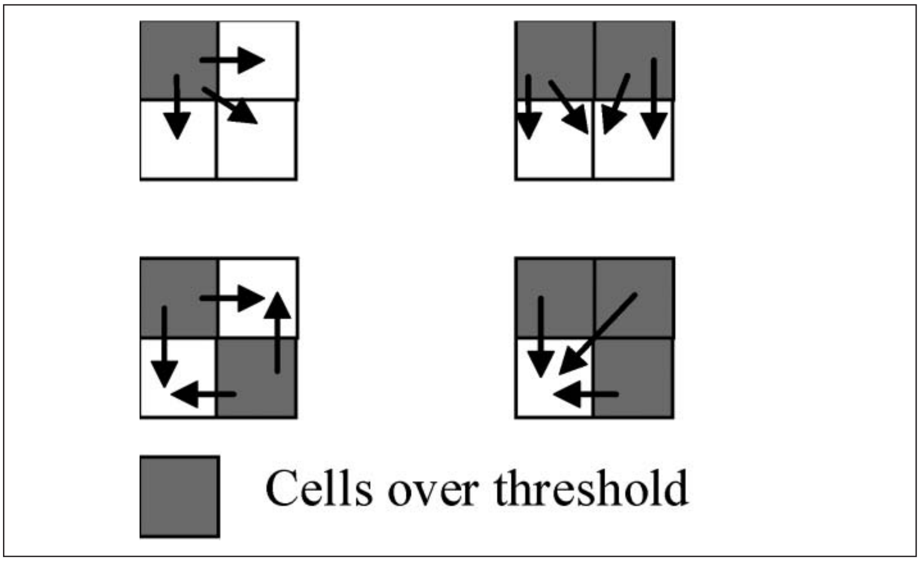


Figure 19. 2D block patterns and transition rules within a Margolus neighborhood

2. Deformation: Initially, all cells within the free-form shape have a number of particles below the threshold. Once the number of particles is above the threshold (because of a push operation), the deformation process begins, and it lasts until the cell space returns to a steady state again.

Figure 20 shows an animation of the 3D free-form compression and deformation process.

Conclusion

In this article, we surveyed how DEVS and its cell-DEVS variant can be used to implement CA in computer simulations. In particular, we emphasized the multiple benefits of using a DEVS approach for the development of serious games. Because of its rigorous formalization and continuing standardization, DEVS favors content sharing and interoperability in the development of serious game applications. In addition, DEVS also makes it easier to realize virtual world evolution in the simulations, thanks to its modular model construction capability and explicit separation of business logic (models) and simulation implementation. As DEVS is based on the discrete-event simulation paradigm, it allows for improving the efficiency of terrain generation without losing accuracy. Finally, DEVS-based simulations can be transparently executed in parallel and distributed computing environments by taking advantage of advanced PADS techniques; it can help reduce the simulation time significantly at reduced cost.

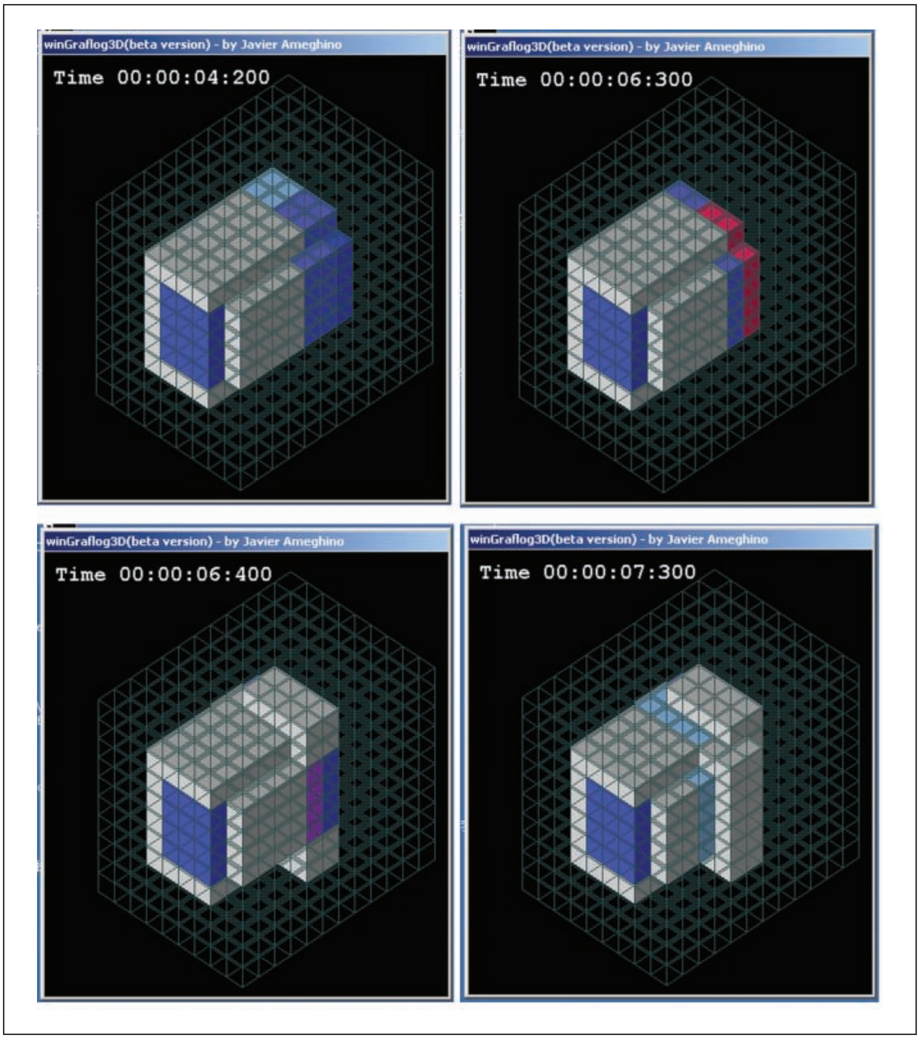

winGraflog3D(beta version) - by Javier Ameghino
Time 00:00:04:200
winGraflog3D(beta version) - by Javier Ameghino
Time 00:00:06:300
winGraflog3D(beta version) - by Javier Ameghino
Time 00:00:06:400
winGraflog3D(beta version) - by Javier Ameghino
Time 00:00:07:300

Figure 20. 3D free-form shape compression and deformation

Also, in this article, we illustrated some of the practical examples that make use of such cellular models in Serious Games applications, ranging from simple ones such as the bouncing ball to more sophisticated ones such as crowd behavior and 3D shape deformation. The variety of these use cases demonstrates the potential of using DEVS-based cellular M&S for studying real-world systems, situations, or phenomena. It also demonstrates how such models can be easily expressed using simple constructions and rules. The ability to express a given model at ease is a clear invitation to experiment with new models and to discover that simple local rules are often sufficient to explain the most complex and sometimes counterintuitive phenomenon observed in the real world.

Authors' Note

Numerous students participated in the definition of the models presented here, including Javier Ameghino, Wilson Venhola, David Tudino, Leandro Fal, Guillermo Vasconcelos, Tao Zheng, Dorin Petriu, Pengfei Wu, Xiuqing Wu, and Hesham Saadawi. The models authored by these students (and the complete reports written by them) can be found at http://www.sce.carleton.ca/faculty/wainer/wbgraf/doku.php. Also, Tristan Roscoe and Ghodrat Azizi participated in the interfacing with Distil's graphical interface.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the authorship and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research and/or authorship of this article:

This work has been partially funded by Natural Sciences and Engineering Research Council of Canada (NSERC).

It has also been supported by University of Nice Sophia Antipolis and INRIA Sophia Antipolis

References

Al-Zoubi, K., & Wainer, G. (2008). Interfacing and coordination for a DEVS simulation protocol standard. In *Proceedings of IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications* (pp. 300-307). New York, NY: ACM Press.

BinSubaih, A., & Maddock, S. (2008). Game portability using a service-oriented approach. *International Journal of Computer Games Technology, 8*, 1-7. doi:10.1155/2008/378485

BinSubaih, A., Maddock, S., & Romano, D. M. (2006). A serious game for traffic accident investigators. *International Journal of Interactive Technology and Smart Education*, *3*, 329-346.

Burks, A. W. (1970). Von Neumann's self-reproducing automata. In A. W. Burks (Ed.), *Essays on cellular automata* (pp. 3-64). Champaign: University of Illinois Press.

Components of an Incident Management Simulation and Gaming Framework and Related Developments Sanjay Jain and Charles R. McLean. SIMULATION, January 2008; vol. 84, 1: pp. 3-25.

Correia, L., & Wehrle, T. (2006). Beyond cellular automata: Towards more realistic traffic simulators. In *Proceedings of the 7th International Conference on Cellular Automata for Research and Industry* (LNCS Vol. 4173, pp. 690-693). New York, NY: ACM Press.

D'Ambrosio, D., Di Gregorio, S., & Iovine, G. (2003). Simulating debris flows through a hexagonal cellular automata: SCIDDICA S3-hex. *Natural Hazards and Earth System Sciences, 3*, 545-559.

Dollner, J., & Buchholz, H. (2005). Continuous level-of-detail modeling of buildings in 3D city models. In *Proceedings of the 13th annual ACM International Workshop on Geographic Information Systems* (pp. 173-181). New York, NY: ACM Press.

Ducheneaut, N., Yee, N., Nickell, E., & Moore, R. J. (2006). Building an MMO with mass appeal: A look at game-play in World of Warcraft. *Games and Culture—Journal of Interactive Media, 1*, 281-317.

Farooq, U., Wainer, G., & Balya, B. (2006). DEVS modeling of mobile wireless ad hoc networks. *Simulation Modeling Practice and Theory, 15*, 285-314.

Friedman, D., Steed, A., & Slater, M. (2007). Spatial social behavior in SECOND LIFE. In *Proceedings of the 7th International Conference on Intelligent Virtual Agents* (LNCS Vol. 4722, pp. 252-263). Berlin, Germany: Springer.

Harzallah, Y., Michel, V., Liu, Q., & Wainer, G. (2008). Distributed simulation and web map mash-up for forest fire spread. In *Proceedings of the 2008 IEEE International Conference on Web Services* (pp. 176-183). New York, NY: ACM Press.

Hsiao, T. Y., & Yuan, S. M. (2005). Practical middleware for massively multiplayer online games. *IEEE Internet Computing, 9*(5), 47-54.

Hu, X. L., Zeigler, B. P., & Mittal, S. (2005). Variable structure in DEVS component-based modeling and simulation. *Simulation: Transactions of the Society for Computer Simulation International, 81*, 91-102.

Kapolka, A., McGregor, D., & Capps, M. (2002). A unified component framework for dynamically extensible virtual environments. In *Proceedings of the 4th International Conference on Collaborative Virtual Environments* (pp. 64-71). New York, NY: ACM Press.

Kofman, E., & Junco, S. (2001). Quantized-state systems: A DEVS approach for continuous system simulation. *Simulation: Transactions of the Society for Computer Simulation International, 18*, 123-132.

Lavis, C. (2010). *Serious games: Transforming learning through well-designed play*. Retrieved from http://www.distilinteractive.com/library/index.php

Layered Intelligence for Agent-based Crowd Simulation. Bikramjit Banerjee, Ahmed Abukmail, and Landon Kraemer. SIMULATION, October 2009; vol. 85, 10: pp. 621-633.

Levenberg, J. (2002). Fast view-dependent level-of-detail rendering using cached geometry. In *Proceedings of the 2002 IEEE Visualization Conference* (pp. 259-265). New York, NY: ACM Press.

Livny, Y., Sokolovsky, N., Grinshpoun, T., & El-Sana, H. (2008). A GPU persistent grid mapping for terrain rendering. *The Visual Computer, 24*, 139-153.

Lovelock, J. (1992, September). *The evolving Gaia theory*. Paper presented at the United Nations University, Tokyo, Japan. Retrieved from http://www.unu.edu/unupress/lecture1.html

MacLeod, M., Chreyh, R., & Wainer, G. (2006). Improved cell-DEVS models for fire spreading analysis. In *Proceedings of ACRI 2006* (LNCS Vol. 4173, pp. 472-481). Berlin, Germany: Springer.

Malamud, B. D., & Turcotte, D. L. (2000). Cellular-automata models applied to nature hazards. *IEEE Computing in Science & Engineering, 2*(3), 42-51.

Composable Cellular Automata. Gary R. Mayer and Hessam S. Sarjoughian. SIMULATION, November 2009; vol. 85, 11-12: pp. 735-749., first published on July 17, 2009

Merabti, M., El Rhalibi, A., Shaheed, A., Fergus, P., & Price, M. (2008). Virtual environments with content sharing. In *Proceedings of the Third International Conference on Technologies for E-Learning and Digital Entertainment* (LNCS Vol. 5093, pp. 328-342). Berlin, Germany: Springer.

Mittal, S., Martin, J. L. R., & Zeigler, B. P. (2007). DEVSML: Automating DEVS execution over SOA towards transparent simulators. In *Proceedings of the DEVS Integrative M&S Symposium, part of the 2007 Spring Simulation Multi-Conference, SpringSim'07* (pp. 287-295). San Diego, CA: SCS International.

Morgan, G. (2009). Challenges of online game development: A review. *Simulation & Gaming: An Interdisciplinary Journal, 40*, 688-710.

Narayanasamy, V., Wong, K. W., Fung, C. C. & Rai, S. (2006). Distinguishing games and simulation games from simulators. *Computers in Entertainment, 4*(2), Article No. 9.

Prakash, E., Brindle, G., Jones, K., Zhou, S., Chaudhari, N., & Wong, K.-W. (2009). Advances in games technology: Software, models, and intelligence. *Simulation & Gaming: An Interdisciplinary Journal, 40*, 752-801.

Qiu, G., Kandhai, D., & Sloot, P. M. A. (2007). Understanding the complex dynamics of stock markets through cellular automata. *Physical Review E*, *75*(4). doi:10.1103/PhysRevE.75.046116

Reis, E. A., Santos, L. B. L., & Pinho, S. T. R. (2009). A cellular automata model for avascular solid tumor growth under the effect of therapy. *Physica A: Statistical Mechanics and its Applications, 388*, 1303-1314.

Serious Games. (2008). *Serious games initiative*. Retrieved from http://www.seriousgames.org

Shiginah, F. A. S. B. (2006). *Multi-layer cellular DEVS formalism for faster model development and simulator efficiency* (Unpublished doctoral thesis). University of Arizona, Tucson.

Slimi, R., EI Yacoubi, S., Dumonteil, E., & Gourbiere, S. (2009). A cellular automata model for Chagas disease. *Applied Mathematical Modelling, 33*, 1072-1085.

Wainer, G. (2002). CD++: A toolkit to develop DEVS models. *Software: Practice and Experience, 32*, 1261-1306.

Wainer, G. (2006). Applying Cell-DEVS methodology for modeling the environment. *Simulation: Transactions of the Society for Modeling and Simulation International, 82*, 635-660.

Wainer, G. (2009). *Discrete-event modeling and simulation: A practitioner's approach*. New York, NY: CRC Press.

"Standardizing DEVS Simulation Middleware". G. Wainer, K. Al-Zoubi, O. Dalle, S. Mittal, J. L. Risco Martín, H. Sarjoughian, L. Touraille, Mamadou K. Traoré, Bernard P. Zeigler. To be included in Discrete-Event Modeling and Simulation: Theory and Applications. G. Wainer, P. Mosterman Eds. Taylor and Francis (submitted on February 2009).

Wainer, G., & Giambiasi, N. (2001). Application of the Cell-DEVS paradigm for cell spaces modeling and simulation. *Simulation, 71*, 22-39.

Wainer, G., & Giambiasi, N. (2002). N-dimensional Cell-DEVS models. *Discrete Event Dynamic Systems, 12*, 135-157.

Wainer, G., & Liu, Q. (2009). Tools for graphical specification and visualization of DEVS models. *Simulation: Transactions of the Society for Modeling and Simulation International, 85*, 131-158.

"An Introduction to DEVS Standardization". G. Wainer, K. Al-Zoubi, S. Mittal, J.L. Risco Martín, H. Sarjoughian, B. P. Zeigler. To be included in Discrete-Event Modeling and Simulation: Theory and Applications. G. Wainer, P. Mosterman Eds. Taylor and Francis (submitted on February 2009).

Was, J. (2005). Cellular automata model of pedestrian dynamics for normal and evacuation conditions. In *Proceedings of the Third International Conference on Intelligent Systems Design and Applications* (pp. 154-159). New York, NY: Springer.

Wu, P., Wu, X., & Wainer, G. (2004). Applying Cell-DEVS in 3D free-form shape modeling. In P. Sloot, B. Chopard, & A. Hoekstra (Eds.), *Proceedings of ACRI 2004* (LNCS Vol. 3305, pp. 81-90). New York, NY: Springer.

Yang, L. Z., Fang, W. F., & Fan, W. C. (2003). Modeling occupant evacuation using cellular automata—Effect of human behavior and building characteristics on evacuation. *Journal of Fire Sciences, 21*, 227-240.

Zeigler, B. P., Kim, T., & Praehofer, H. (2000). *Theory of modeling and simulation: Integrating discrete event and continuous complex dynamic systems*. New York, NY: Academic Press.

Zeigler, B. P., Mittal, S., & Hu, X. L. (2008, May). *Towards a formal standard for interoperability in M&S/system of systems integration*. Paper presented at the proceedings of the GMU-AFCEA Symposium on Critical Issues in C4I, George Mason University, Fairfax, VA.

Zyda, M. (2005). From visual simulation to virtual reality to games. *IEEE Computer, 38*(9), 25-32.

Zyda, M. (2007). Creating a science of games. *Communications of the ACM, 50*(7), 26-29.

Bios

Gabriel Wainer, SMSCS, SMIEEE, received his MSc degree from the University of Buenos Aires, Argentina, and his PhD degree (1998, with highest honors) from Université d'Aix-Marseille III, France. In 2000, he joined the Department of Systems and Computer Engineering, Carleton University, where he is now an associate professor. He held positions at the Computer Science Department of the University of Buenos Aires and visiting positions at the University of Arizona, LSIS (CNRS), University of Nice, and INRIA Sophia Antipolis (France). He is a recipient of the First DEVS M&S Award, Carleton University Research Achievement Award, IBM Eclipse Innovation, and various best paper awards. Contact: gwainer@sce.carleton.ca.

Qi Liu received his BEng from the Huazhong University of Science and Technology, China, in 1993 and MSc degree from the Carleton University, Canada, in 2006. He was a recipient of a Senate Medal for Outstanding Academic Achievement for his research work and a variety of distinguished scholarships. He is currently a PhD candidate in the Department of Systems and Computer Engineering at Carleton University, Ottawa, Ontario, Canada. Contact: liuqi@sce.carleton.ca.

Olivier Dalle is *Maître de Conférences* (associate professor) in the Informatics Department of the Faculty of Sciences at University of Nice–Sophia Antipolis (UNS). He received his BSc from University of Bordeaux 1 and his MSc and PhD from UNS. From 1999 to 2000, he was a postdoctoral fellow at the French Space Agency Center in Toulouse (CNES-CST). In 2000, he joined UNS and the MASCOTTE research group, a joint team of UNS, CNRS, and INRIA. Contact: olivier.dalle@sophia.inria.fr.

Bernard P. Zeigler is a professor of electrical and computer engineering at the University of Arizona, Tucson, and director of the Arizona Center for Integrative Modeling and Simulation. He is internationally known for his 1976 foundational text *Theory of Modeling and Simulation*. He has published numerous books and research publications on the Discrete-Event System Specification formalism. He was named Fellow of the IEEE (1995) and Society for Computer Simulation (SCS; 2006). He received numerous awards, including McLeod Founder's Award (2000) by the SCS (its highest recognition), and was inducted to the SCS M&S Hall of Fame in 2010. Contact: zeigler@ece.arizona.edu.