Tools for Graphical Specification and Visualization of DEVS Models

Gabriel Wainer and Qi Liu

SIMULATION 2009; 85; 131
DOI: 10.1177/0037549708101182

Tools for Graphical Specification and Visualization of DEVS Models

Gabriel Wainer
Qi Liu
Department of Systems and Computer Engineering
Carleton University Centre on advance Visualization and Simulation (V-Sim)
Carleton University
1125 Colonel By Drive
Ottawa, ON K1S 5B6, Canada
gwainer@sce.carleton.ca

We introduce advanced graphical modeling and visualization facilities for Discrete Event System Specification (DEVS) modeling and simulation (M&S) in the CD++ environment. The objective is to provide general users with a variety of easy-to-use environments to facilitate the model analysis process and thereby promoting the adoption of M&S by a wider community of practitioners and researchers. CD++Modeler allows users without much experience in software development to construct rather complex DEVS models and to analyze simulation data using 2D graphics. We also introduce a graphical platform called MAPS designed specifically for urban traffic systems, and other advanced 3D animation tools (CD++/VRML, CD++/Maya, CD++DEVSView, and CD++/Blender) based on both commercial and open-source software packages. We elaborate on the design of these toolkits and demonstrate their capabilities as well as relative merits and limitations with realistic applications. Following a highly modular approach, the resulting architecture can be easily extended to incorporate other modeling and visualization techniques in future development. We show that these facilities can reduce the model development cost significantly, lower the learning curve for general users, and improve the comprehension of continuously evolving models, making them suitable for efficient decision making.

Keywords: virtual reality, graphics and animation, Web-based environments, DEVS methodology

1. Introduction

With the computing power and advanced software tools available today, modeling and simulation (M&S) allows for cost-effective and detailed analysis of natural and artificial systems where a mathematical approach is intractable. A methodology that has gained increasing popularity in recent years is the Discrete Event System Specification (DEVS) formalism [1]. By decoupling the model and simulation concepts, the DEVS framework offers two major benefits. First, the same model can be executed on different simulators, allowing for portability and interoperability at a high level of abstraction. Moreover, the well-defined separation of concerns permits models and simulators to be independently verified and reused in later combinations with minimal re-verification. Furthermore, DEVS supports hierarchical and modular construction of models, reducing the development and testing effort. The Cell-DEVS formalism [2] extends the DEVS theory to describe n-dimensional cell-spaces as discrete-event models. Both DEVS and Cell-DEVS formalisms are implemented in CD++ [3], which is an open-source M&S environment that supports standalone and parallel/distributed simulation on different platforms and has been used to solve a variety of sophisticated problems successfully [4–8].

In spite of its great power and capabilities, however, M&S technology is still viewed by many practitioners as an intimidating tool to use. The major reasons for this are twofold. First, M&S is a knowledge-intensive process that requires in-depth expertise not only in the underlying application domain but also in software development and system integration. Current simulation practice relies

SIMULATION, Vol. 85, Issue 3, March 2009 131–158

DOI: 10.1177/0037549708101182
Figures 5, 7, 9–16, 18, 19, 21, 22, 24–30, 33, 35–38 appear in color online: http://sim.sagepub.com

on close cooperation between application domain specialists and software engineers who are usually not experts in that specific domain. This cross-domain communication often leads to a fair amount of difficulties in model specification, validation, and verification. Second, M&S relies on human-computer interaction (HCI) and everything associated with HCI, including model presentation, representation, interface metaphors, interaction modalities, and evaluation techniques [9]. The lack of an adequate visualization mechanism that is generic and flexible enough to show different simulation results in an intuitive and user-friendly manner makes the analysis of complex system behavior time consuming, costly and error prone.

Although various DEVS-based toolkits have been developed for tackling complex problems in a broad array of domains, their adoption in the industry is still relatively limited. One reason is that the users are required to have rather extensive expertise in advanced programming techniques (e.g. object-oriented programming, parallel and distributed computing over heterogeneous platforms, and standardization and interoperability issues, etc.). Consequently, most successful applications involve multidisciplinary teams with a special group in charge of software development. This increases the M&S cost and makes it difficult for domain experts to explore their models at first hand. To promote the awareness of DEVS-based M&S techniques and assist general users in a wider community, this paper introduces a state-based graphical modeling paradigm, based on which a new toolkit (CD++Modeler) has been developed in our research that allows users lacking programming experience to construct rather complex DEVS models without the intervention of software engineers and to analyze simulation results in a natural visual way. This toolkit is integrated as a core module in the CD++Builder package, an integrated M&S platform delivered as an Eclipse plugin that has been used to support a model-centered methodology for developing real-time and embedded systems at reduced cost [10].

In order to present a general picture of the graphical modeling and visualization facilities currently available in the CD++ environment, we summarize and review the different mechanisms that have been employed to interface CD++ with sophisticated commercial and open-source 3D visualization and rendering techniques to fulfill the needs of different user communities. These techniques include Virtual Reality Modeling Language (VRML), Autodesk Maya [11], Open Graphics Library (OpenGL) [12], and Blender [13]. The result is a suite of toolkits, known as CD++/VRML, CD++/Maya, DEVSView, and CD++/Blender respectively, with varied capabilities to improve the comprehension of continuously evolving models and to facilitate the decision-making process. A special graphical toolkit called MAPS [14], designed exclusively for simulation of urban traffic, is also discussed in this paper to exemplify the potential of applying DEVS-based graphical modeling and visualization techniques in real-world settings. Although many of these tools have already been presented individually in our prior work, this paper highlights the design considerations behind these toolkits and compares their relative merits and limitations with new applications.

The rest of the paper is organized as follows. Section 2 recaps the background information about the formalisms and the CD++ environment. It also gives a brief survey of some of the existing graphical modeling and visualization tools, showing the popularity of this approach. A multidimensional taxonomy of the CD++ family of visualization toolkits is also presented in this section. Section 3 covers the graphical modeling paradigm and its realization in CD++Modeler. Section 4 discusses the integration between CD++ and VRML and shows how this technique can be used to improve urban traffic simulation. Section 5 is concerned with the integration of CD++ with other advanced 3D visualization techniques. Throughout the discussion, realistic applications are presented to demonstrate the features and capabilities of the integrated environment. Section 6 closes the paper with conclusion remarks and future work.

2. Background

Based on general dynamic systems theory, the DEVS formalism [1] provides a sound M&S framework for defining hierarchical discrete-event models in a modular way, where a system is described as a composition of behavioral (atomic) and structural (coupled) components. Unlike the discrete-time simulation approach, DEVS uses a continuous time base and allows for asynchronous model execution, improving the efficiency of the simulation without losing accuracy. Cellular Automata (CA) [15] are capable of producing a great variety of complex behavior of systems represented as cell spaces. Traditionally, CA models are implemented on a computer using a discrete-time approach. The behavior of a cell space depends on synchronous evaluation of local functions defined in the cells at discrete-time intervals. The Cell-DEVS formalism [2] combines the advantages of CA and DEVS to describe n-dimensional cell spaces as discrete-event models, where each cell is represented as a DEVS atomic model communicating with its neighboring cells and outside model components using a modular interface. It defines different timing semantics for each cell, allowing explicit timing specification, asynchronous model execution, and seamless integration with other types of models.

CD++ [3] is an open-source environment that supports standalone and parallel/distributed simulation of DEVS and Cell-DEVS models. Over the years, it has been developed as a set of software toolkits running on many mainstream operating systems. The environment provides two major frameworks: a modeling framework that allows users to define the behavior of atomic and coupled models using a built-in graph-based specification language or C++; and a simulation framework that creates an executive

```
I / 00:00:00:000 / top(38) / life(01)
D / 00:00:00:000 / life(2,2)(16) / 00:00:00:000 / life(01)
@ / 00:00:00:000 / top(38) / life(01)
Y / 00:00:00:000 / life(2,2)(16) / out / 0.00000 / life(01)
X / 00:00:00:000 / life(01) / neighborchange / 1.00000 / life(2,2)(16)
D / 00:00:00:000 / life(2,2)(16) / 00:00:00:000 / life(01)
* / 00:00:00:000 / top(38) / life(01)
D / 00:00:00:000 / life(2,2)(16) / 00:00:00:100 / life(01)
@ / 00:00:00:100 / top(38) / life(01)
Y / 00:00:00:100 / life(2,2)(16) / out / 1.00000 / life(01)
D / 00:00:00:100 / life(2,2)(16) / 00:00:00:000 / life(01)
```

Figure 1. An excerpt of a log file created in CD++

```
Line : 1 - Time: 00:00:00:000
   0 1 2 3 4 5
  +-----------+
0|           |
1|  1 1      |
2|  1        |
3|        1  |
4|      1 1  |
5|           |
  +-----------+

Line : 2 - Time: 00:00:00:100
   0 1 2 3 4 5
  +-----------+
0|           |
1|  1 1      |
2|  1 1      |
3|      1 1  |
4|      1 1  |
5|           |
  +-----------+

Line : 3 - Time: 00:00:00:200
   0 1 2 3 4 5
  +-----------+
0|           |
1|  1 1      |
2|  1        |
3|        1  |
4|      1 1  |
5|           |
  +-----------+
```

Figure 2. Textual representation of the Life game

entity for each component in the model hierarchy to carry out the simulation in line with the formalisms. Cell-DEVS models can be coded in simple rules with a few parameters using predefined operators, functions and constants.

The simulation is executed in a message-driven fashion. CD++ messages fall into two categories: *content messages* include the external message (X) and output message (Y) that encode the actual data transmitted between the models, while *control messages* include the initialization message (I), collect message (@), internal message (*), and done message (D) that are used to synchronize the simulation. During the simulation, all messages exchanged between the models are recorded in log files, which can then be used for debugging and visualization purposes. Figure 1 shows an excerpt of a log file created for the Game of Life [16], where the output messages are highlighted. Each Y message contains information regarding the time of the event, source model, output port, value of the data, and destination model.

As we can see, it would be very difficult and error prone to obtain a high-level depiction of the model behavior from the log files and extensive interpretation and reconstruction are required to have a clear understanding of what occurred in the simulation. To ease the analysis of simulation data, a utility tool is provided to translate the log files into a textual representation of the simulated model, as seen in Figure 2.

However, this textual representation is only available for presenting Cell-DEVS models, as there is no convenient way to draw the input and output trajectories of a general DEVS model. In addition, three or higher dimensional cell spaces can be represented only as a series of individual slices (each for a 2D plane), further reducing the expressive power of the textual representation.

Many graphical M&S techniques have been proposed in the literature. The following is a non-comprehensive list of recent efforts, showing the popularity of this approach.

- Modelica [17] is an object-oriented language for modeling complex and heterogeneous physical systems and controllers. Models are described using differential, algebraic and discrete equations. Dymola [18] is a commercial tool that uses Modelica for real-time visual simulation of electromechanical, automotive, aerospace, and robotic systems. 3D animation and plotting facilities are included in the tool to help analyze the simulated models.

- VisualSense [19] is an M&S framework for wireless sensor networks that supports component-oriented construction of models based on the Ptolemy II modeling and visualization environment [20]. A Java-based graphical user interface is provided in Ptolemy II that allows users to develop continuous-time models as block diagrams, and hybrid system models are constructed by combining finite state machines with continuous-time models. It also includes a 2D Java plotter and a 3D animation module that uses Java 3D library to render simulation results.

Table 1. A taxonomy of graphical modeling and visualization toolkits in CD++

Toolkits / Criteria	CD++ Modeler	CD++/ VRML	MAPS	CD++/ Maya	DEVSView	CD++/ Blender
Cost	Open-source	Open-source	Open-source	Commercial	Open-source	Open-source
Capability	Graphical modeling and visualization	Visualization only	Visualization only	Visualization only	Visualization only	Visualization only
Applicability	General DEVS and Cell-DEVS models	General DEVS and Cell-DEVS models	Urban traffic M&S only	General DEVS and Cell-DEVS models	General DEVS and Cell-DEVS models	General DEVS and Cell-DEVS models
Visual effect	2D	3D	3D	3D	3D	3D
Dependency	Java-based component integrated with CD++Builder	Web-based Standalone toolkit using VRML	VRML-based Standalone toolkit	Standalone toolkit using Maya	OpenGL-based Standalone toolkit	OpenGL-based Standalone toolkit
Installation footprint	Small	Small	Medium	Large	Medium	Large

- RUBE [21] is an XML-based framework that decouples model specification from model presentation so that dynamic models can be formally specified and then presented in customized 2D or 3D visualization. Users must manually associate an icon with its semantic meaning during the model authoring procedure, and need to program the semantic functions for each model component. The framework provides a graphical user interface (GUI) to generate the model representation that consists of a scene file and a model behavior file using open-source computer graphics software such as Sodipodi [22] and Blender [13].

- Trend/jTrend [23] is a general purpose 2D CA environment with an integrated simulator and a compiler. It provides a GUI for defining CA rules and dynamically changing the attributes of the cellular spaces, and supports 2D text-based animation of CA models.

- BioSim [24] is an interactive and visual problem-solving environment for the biomedical domain that uses CA to model biological behaviors. It provides a 3D virtual world model in which users can explore biological interactions either as an observer or by immersive role playing. Photorealistic 3D models of components of the system are created with Autodesk 3ds Max and exported to 3D Game Studio, which is then used to construct the game scenes, bio-morphed characters and the integration of the world/character dynamics and interactions.

- JDEVS [25] is a DEVS-based M&S environment that enables object-oriented, general purpose, component-based, and Geographical Information System (GIS) connected visual simulation model development and execution. It includes easy-to-use 2D and 3D visualization tools to render the simulated phenomena.

Although graphical modeling and visualization technique has been the topic of a great number of studies, its application to DEVS and Cell-DEVS models is only rarely explored in the literature [25]. In the following sections, we will present a suite of toolkits developed around the CD++ environment based on various advanced visualization and rendering techniques to assist in the analysis of massive simulation data for complex models. To improve comprehension and emphasize distinctions between these toolkits, Table 1 gives a multi-dimensional taxonomy of the CD++ family of visualization toolkits using different criteria.

3. DEVS-based Graphical Modeling and Animation Framework

To assist users who are not necessarily computer software specialists in the modeling process, the DEVS Graphs notation originally described in [26] has been included in CD++ for defining the behavior of DEVS atomic models in a natural visual way [10, 27]. With DEVS Graphs, users can think about the problem at hand in a more abstract way and develop models as state machines based on formal DEVS specifications. Using DEVS Graphs, atomic models are defined as a set of symbolic objects, each with several attributes, to represent the states and state transitions. In order to be simulated in CD++, DEVS Graphs are translated into a machine-processable representation called GADscript (GrAphical DEVS script). Figure 3 shows the graphical and GADscript representations.

Figure 3(a) shows a state, defined as a bubble with an identifier and a lifetime (translated to GADscript state keyword; the lifetime of a state is then assigned to the corresponding stateId in a separate statement). Figure 3(b) shows the definition of internal transitions, represented as

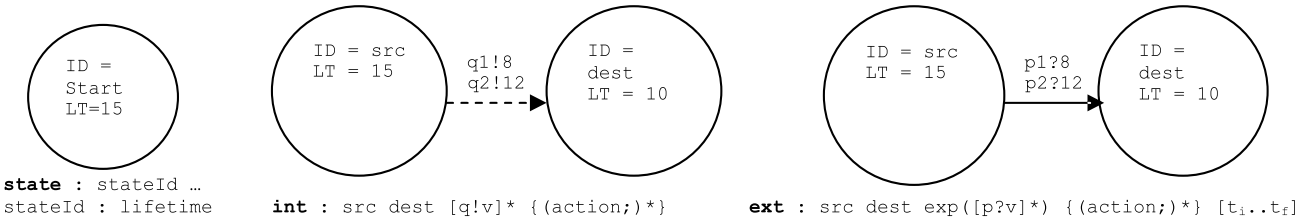

Figure 3. Graphical and GADscript notation of states and transitions

a dotted arrow between a source and a destination state. In GADscript the keyword int is used, and the output function is defined as a set of events denoted as q!v (i.e., sending value v through port q). Finally, Figure 3(c) depicts an external transition as a solid arrow between a source and a destination state. GADscript uses ext to define an external transition, and an input event is denoted as p?v (i.e., arrival of an input value v at port p). The external transition happens only if the required triggering condition is hold, which is specified using a logical expression that involves the input events. The time values [t1..tf] provide a mechanism to verify the timing properties of the simulated system.

However, this approach also has some limitations in building sophisticated systems. In most cases, it can be difficult to represent complex model behavior by looking only to state changes. Although we can attack such complexity by reducing model granularity (i.e., using DEVS hierarchical decomposition), this could result in a model with artificially complex structure (for instance, we could write a state machine to approximate the computation of a continuous function; the resulting model is much more complex than the definition of such a model using a high-level language). To deal with these cases, CD++ includes constructions (actions) that allow modelers to invoke external functions during state transitions, providing a flexible mechanism for defining complex model behavior while still maintaining the simplicity and intuitiveness of the graphical approach. The list of actions manipulates temporary variables defined in the model. Actions can be simple mathematical expressions, or they can be implemented in user-defined C++ functions. Thanks to this mechanism, users can now combine models written in DEVS Graphs with others defined only using C++, or a mix of them (DEVS Graphs with actions).

DEVS atomic models interact with other models through its input/output ports. DEVS Graphs represent them as arrowheads attached to a model definition, as shown in Figure 4. A port is specified in GADscript by its name and data type (e.g., port p2 only accepts input data of type float). The default data type of a port is integer (e.g., both input port p1 and output port q1 are integer). Using these notations, a DEVS atomic model can be specified graphically. Figure 5 gives a DEVS Graph definition of an atomic model called sender. The corresponding GAD-

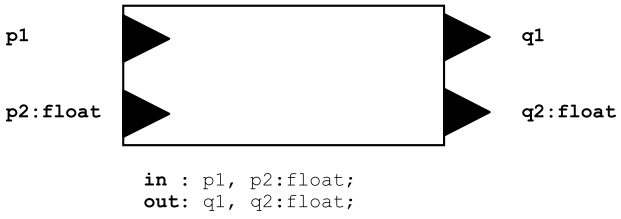

Figure 4. Graphical and GADscript notation of input and output ports

script specification is shown in Figure 6. As we can see, the atomic model declares 11 states (start is designated as the initial state). Passive and transient states are allowed. For example, the start is a passive state with an infinite lifetime (line 26); sendingm0 is a transient state with lifetime 0 (line 34). The state transitions use four temporary variables, which are declared (line 4) and initialized (line 37 to 40) in the specification. The internal transition on line 11, for example, occurs when the lifetime of sendingm1 is consumed. In this case, it is a transient state, so the transition is fired immediately after sending the value of msg1 through port m1. The external transition on line 21 is triggered if the current state is waitmsg1 and a valid input event is received at port i. The model changes to state m0ok, and updates the variables msg1 and flag1 in the corresponding action.

Once the atomic models are defined, they can be combined into a coupled model using graphical notations as well. A coupled model may include input/output ports, atomic or coupled models that have already been defined, and the coupling links between them. Figure 7 and 8 show the DEVS Graph and GADscript specification of a coupled model called network. The input and output ports of the coupled model are represented as arrowheads. Atomic models are shown as bubbles, while coupled models are depicted as squares. The solid arrows represent the coupling links between the model components and input/output ports. The links are defined using keyword link in GADscript. The coupled model in Figure 7 has two atomic components and one coupled component, as specified at line 2 in Figure 8 with keyword components. Each atomic or coupled model is an

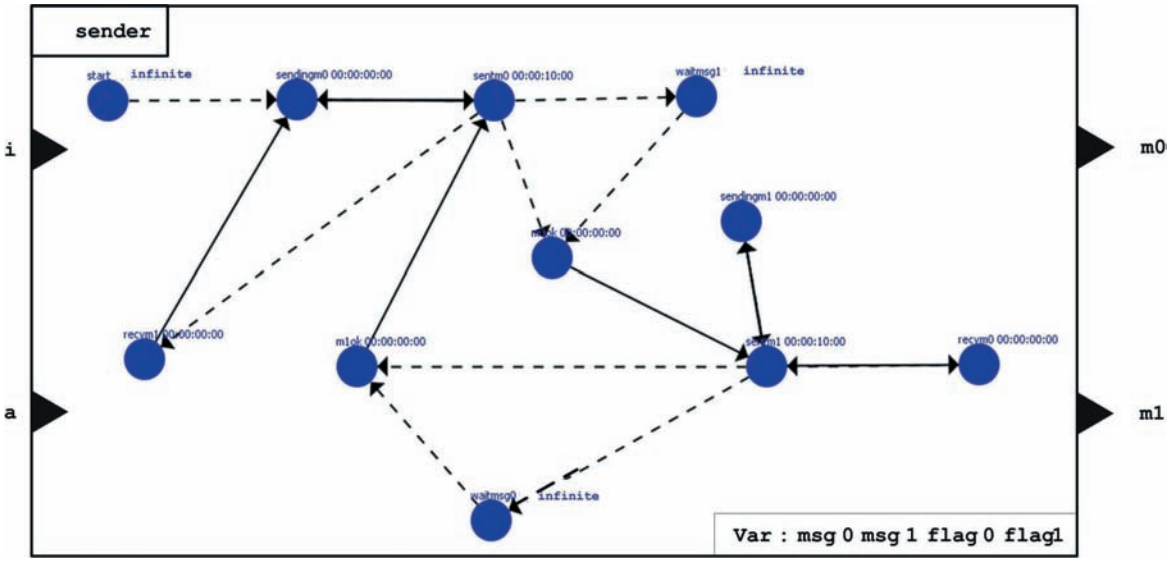

Figure 5. DEVS Graph specification of an atomic model

```
1.  [sender]
2.  in: i a
3.  out: m0 m1
4.  var: msg0 msg1 flag0 flag1
5.  state: start sentm0 m0ok m1ok sentm1 recvm0 recvm1 waitmsg1 sendingm0 waitmsg0 sendingm1
6.  initial: start
7.  int: sendingm0 sentm0 m0!msg0
8.  int: recvm1 sentm0
9.  int: m1ok sentm0 m0!msg0
10. int: m0ok sentm1 m1!msg1
11. int: sendingm1 sentm1 m1!msg1
12. int: recvm0 sentm1
13. int: sentm0 sendingm0
14. int: sentm1 sendingm1
15. ext: sentm0 recvm1 and( any(i), not( flag1) ) ? 1 {msg1 = i;flag1 = 1;}
16. ext: sentm0 m0ok and(equal(a,0), equal(flag1,1) ) ? 1 {flag0 = 0;}
17. ext: sentm0 waitmsg1 and(equal(a,0), notequal(flag1,1) ) ? 1 {flag0 = 0;}
18. ext: start sendingm0 any(i)?1 {msg0 = i;flag0 = 1;}
19. ext: sentm0 sendingm0 Value(a)?1
20. ext: sentm1 m1ok and(equal(a,1),equal(flag0,1)) ? 1 {flag0 = 0;}
21. ext: waitmsg1 m0ok any(i)?1 {msg1 = i;flag1 = 1;}
22. ext: sentm1 recvm0 and( any(i), not( flag0) ) ? 1 {msg0 = i;flag0 = 1;}
23. ext: sentm1 waitMsg0 and(equal(a,1),notequal(flag0,1)) ? 1 {flag1 = 0;}
24. ext: waitmsg0 m1ok any(i)?1 {msg0 = i;flag0 = 1;}
25. ext: sentm1 sendingm1 a?0
26. start: infinite
27. sentm0: 0:0:10:0
28. m0ok: 0:0:0:0
29. m1ok: 0:0:0:0
30. sentm1: 0:0:10:0
31. recvm0: 0:0:0:0
32. recvm1: 0:0:0:0
33. waitmsg1: infinite
34. sendingm0: 0:0:0:0
35. waitmsg0: infinite
36. sendingm1: 0:0:0:0
37. msg0: 0
38. msg1: 0
39. flag0: 0
40. flag1: 22
```

Figure 6. GADscript representation of an atomic model

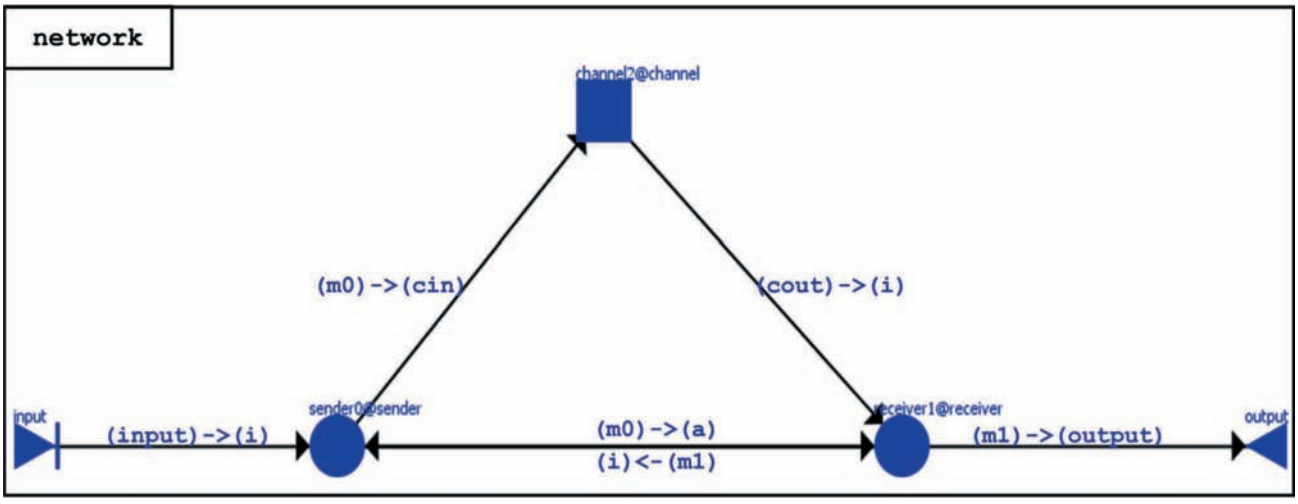

Figure 7. DEVS Graph specification of a coupled model

```
1.  [network]
2.  components : sender0@GGAD receiver1@GGAD channel2@channel
3.  out :  output
4.  in :   input
5.  Link : input i@sender0
6.  Link : m0@sender0 a@receiver1
7.  Link : m1@receiver1 i@sender0
8.  Link : m1@receiver1 output
9.  Link : m0@sender0 cin@channel2
10. Link : cout@channel2 i@receiver1
11.
12. [sender0]
13. source : sender.CDD
14.
15. [receiver1]
16. source : receiver.CDD
17.
18. [channel]
19. components : link0@GGAD link1@GGAD
20. out :  cout
21. in :   cin
22. Link : cin in@link0
23. Link : out@link0 in@link1
24. Link : out@link1 cout
25.
26. [link0]
27. source : link.CDD
28.
29. [link1]
30. source : link.CDD
```

Figure 8. GADscript representation of a coupled model

instance of a predefined model specification. For example, the sender0@sender creates an instance of the atomic model sender that we have defined earlier. The GADscript specifications of atomic and coupled models are saved in files (referred to as CDD and MA files respectively) so that they can be reused in later model development. For

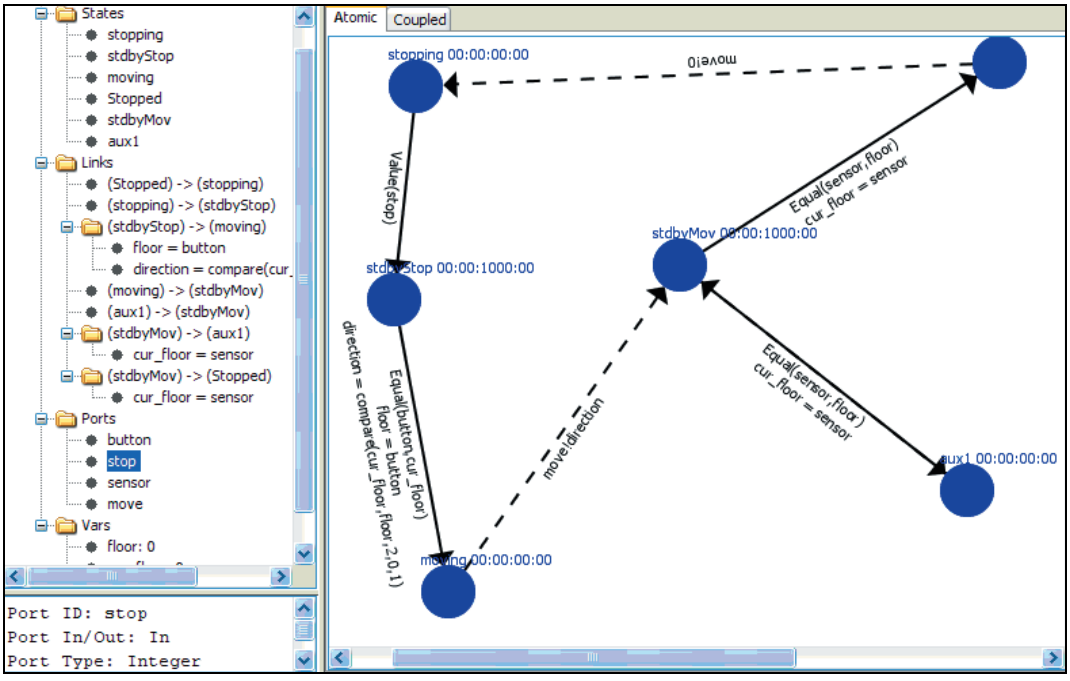

Figure 9. A screenshot of the CD++Modeler graphical user interface

instance, line 13 in Figure 8 refers the definition of the sender model to file sender.CDD. The DEVS Graph allows users to define hierarchical models in a modular way, as required by the DEVS formalism.

These facilities are implemented in CD++Modeler, a DEVS-based toolkit for graphical model construction and 2D visualization. As one of the core modules of the CD++Builder Eclipse plugin [10], CD++Modeler provides a GUI that allows users to construct DEVS atomic and coupled models graphically in a simple drag-and-drop fashion using the notations just introduced. Figure 9 shows a screenshot of the GUI. The navigation panel on the left shows a tree of units for the current model (i.e., states, ports, variables, and transitions of an atomic model or the components of a coupled model) for quick access. The model editing panel on the right provides a workspace where users can construct atomic or coupled models graphically. The graphical specifications of atomic and coupled models can be imported as templates (or classes) in construction of other coupled models. Likewise, many instances with different properties can be created from an existing model template, promoting model reuse and reducing model development time.

Users can also use customizable graphical metaphors, including a repository of image icons for representing different model components in DEVS Graphs. Figure 10 shows the DEVS Graph specification of a supply chain model using customized image icons. This presents an intuitive mapping between the model and the real system, enriching the expressive power of the tool. By parsing the output streams generated during the simulation, CD++Modeler can visualize DEVS models in two different ways: by plotting the input and output (I/O) trajectories of a model component (atomic or coupled) based on the external (X) and output (Y) messages, or by showing the interactions between the ingredient components of a DEVS coupled model.

Figure 11 shows the animation of I/O trajectories on the various ports of an atomic model. Users can adjust the scaling of the signals, change the time format, and focus on specific ports by selecting the checkboxes on the control panel. This oscillogram-like animation allows users to investigate model behavior at the I/O functional level and establishes a straightforward causal relationship between the inputs and outputs, greatly facilitating the model analysis task. Figure 12 shows the animation of interactions among the components of a coupled model. The animation is overlaid on top of the DEVS Graph specification of a coupled model, displaying message passing along the coupling links with 2D-text effects. For example, the animation in Figure 12 shows that a message carrying the value of variable Material is sent from the Transportation component to the Manufacturer component at simulated time 43:30:00:000. Users can play the whole animation or step through it one frame at a time by using the control buttons. This animation gives a high-level view of the interior behavior of a coupled model, helping users interpret and reconstruct what is happening in the simulation.

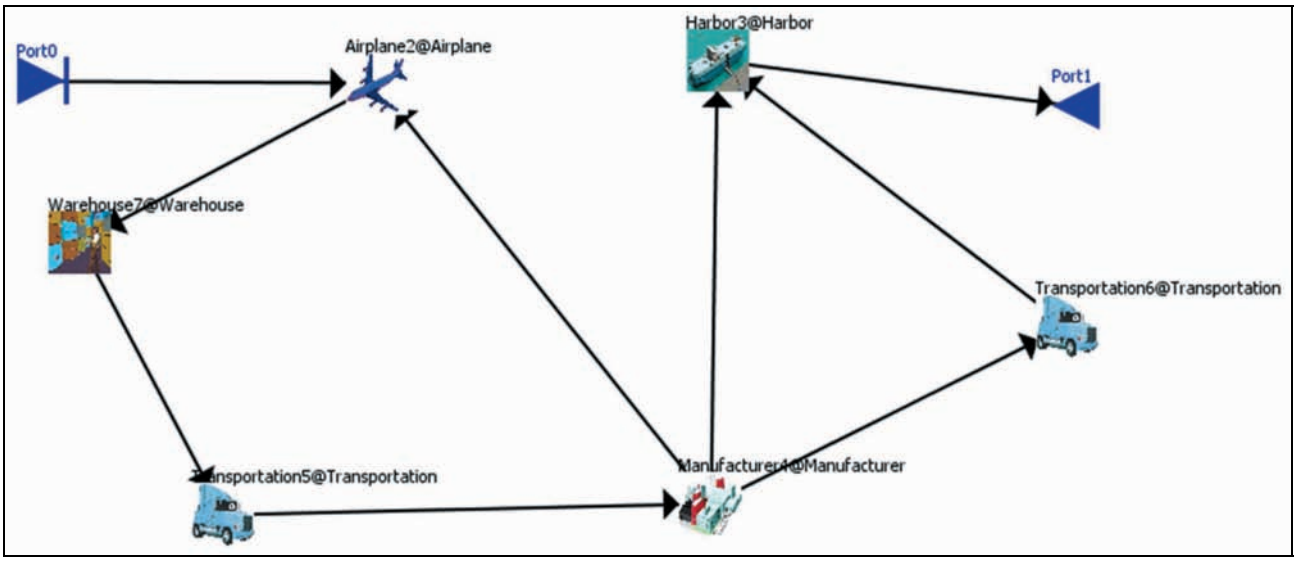

Figure 10. A supply chain model defined in CD++Modeler using customized image icons

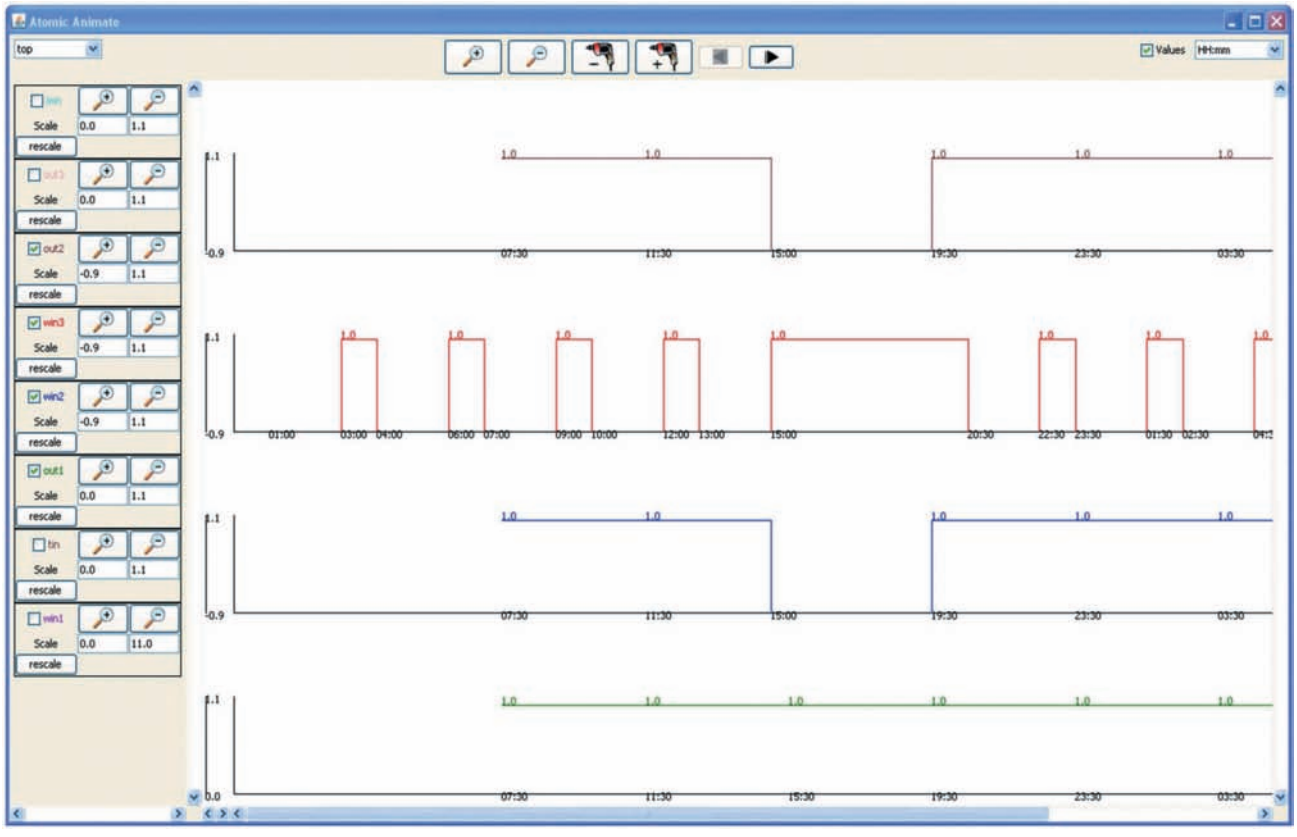

Figure 11. Animation of the input and output trajectories of a DEVS model in CD++Modeler

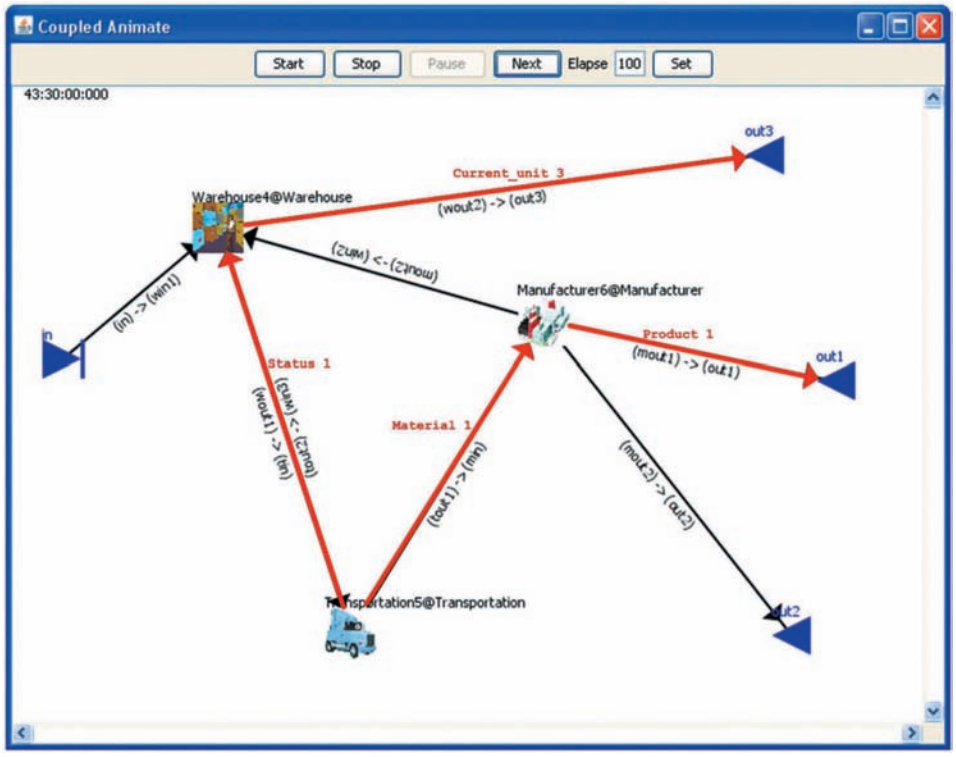

Figure 12. Animation of the interactions between components of a DEVS coupled model in CD++Modeler

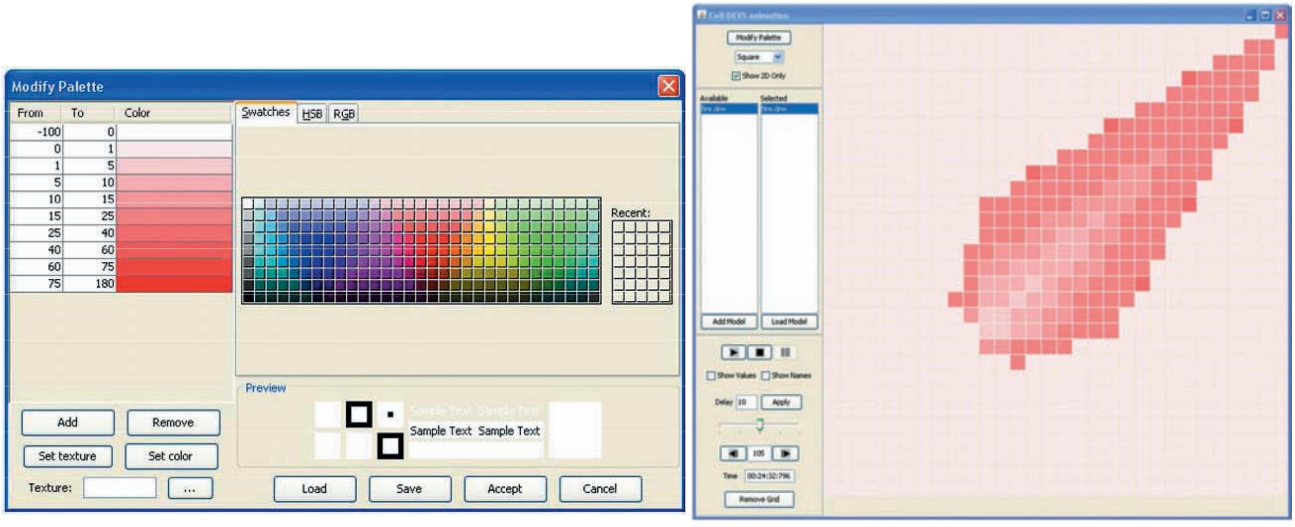

Figure 13. Animation of 2D Cell-DEVS models using user-specified palette

In Cell-DEVS models, users are usually interested in the evolution of the cell space as a whole. To this end, CD++Modeler provides a GUI to animate cell spaces. Figure 13(a) shows a GUI that allows users to choose different color palettes. Figure 13(b) illustrates the animation of a 2D Cell-DEVS model that simulates the propagation of

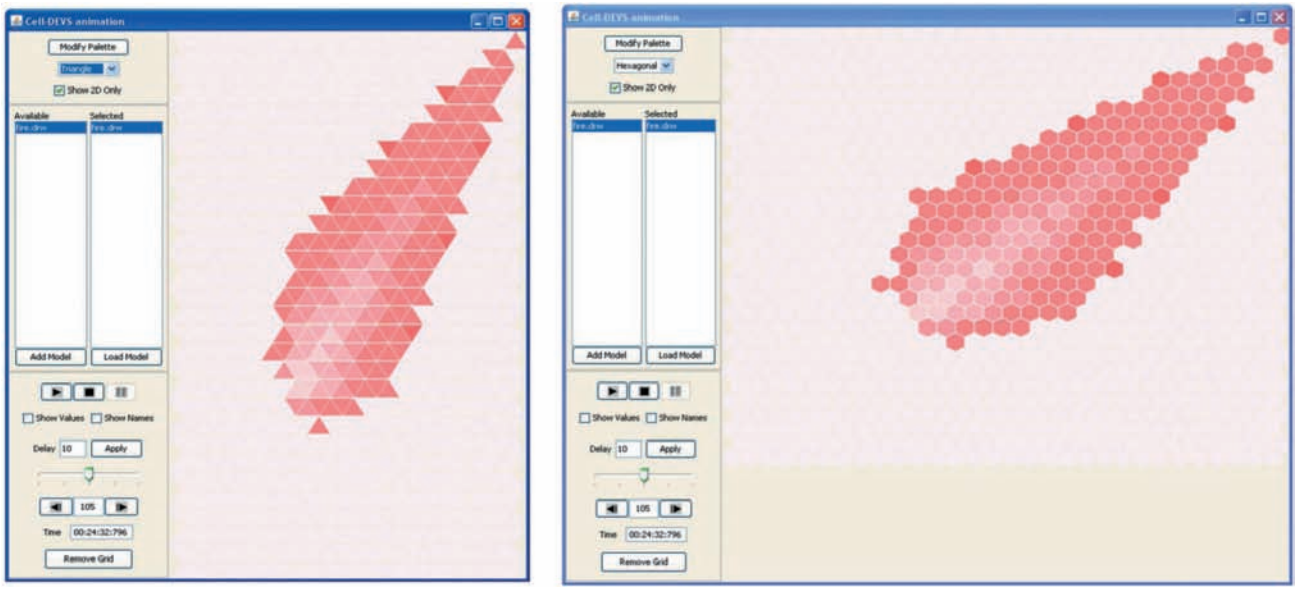

Figure 14. Animation of 2D Cell-DEVS models using different geometries

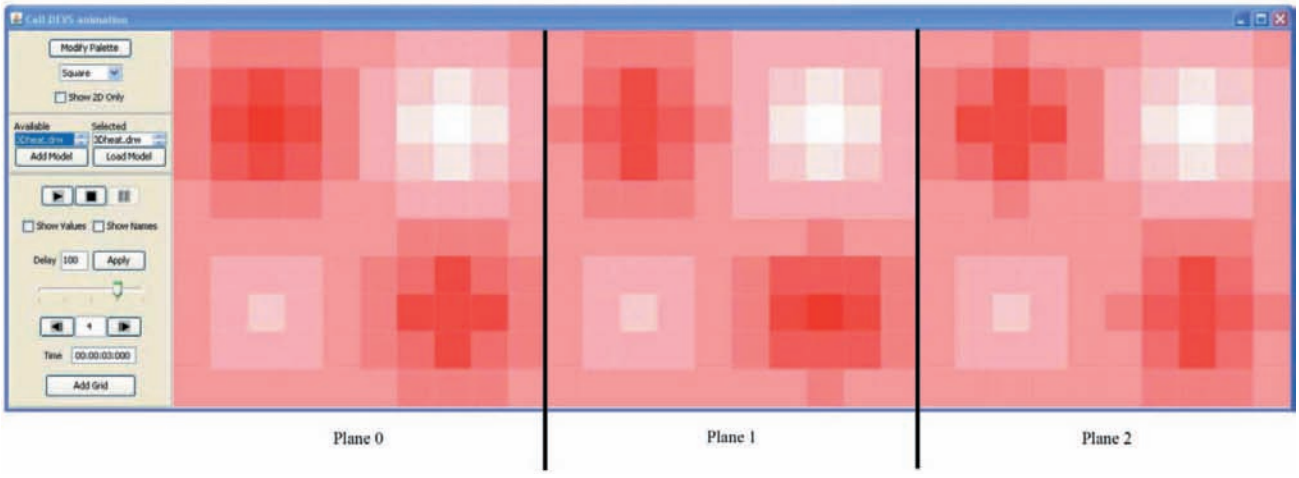

Figure 15. Animation of a 3D heat diffusion Cell-DEVS model as separate planes

forest wildfires. Users can load multiple Cell-DEVS models, change the frame duration, run or pause the animation, and single-step (forward or backward) through the animation sequence. These functionalities are also available through a Cell-DEVS applet visualization engine that can be found at www.sce.carleton.ca/faculty/wainer/wbgraf.

Although Cell-DEVS models usually employ a square geometry (which is relatively simple for model specification and visualization), this might fail to accurately model certain phenomena that possess highly uniform or isotropic properties (i.e., homogeneous behavior in every direction). Using different geometries such as triangular or hexagonal-shaped cells may enable more appropriate and natural model definitions [3]. CD++Modeler provides a lattice translator that is able to perform automated mapping between different geometries in visualizing Cell-DEVS models. Figure 14 shows the animation of the fire propagation model using triangular and hexagonal-shaped cells. As illustrated in Figure 15, CD++Modeler visualizes high-dimensional cell spaces as a series of individual planes due to the inherent limitation of 2D visualization, sacrificing the intuitiveness and

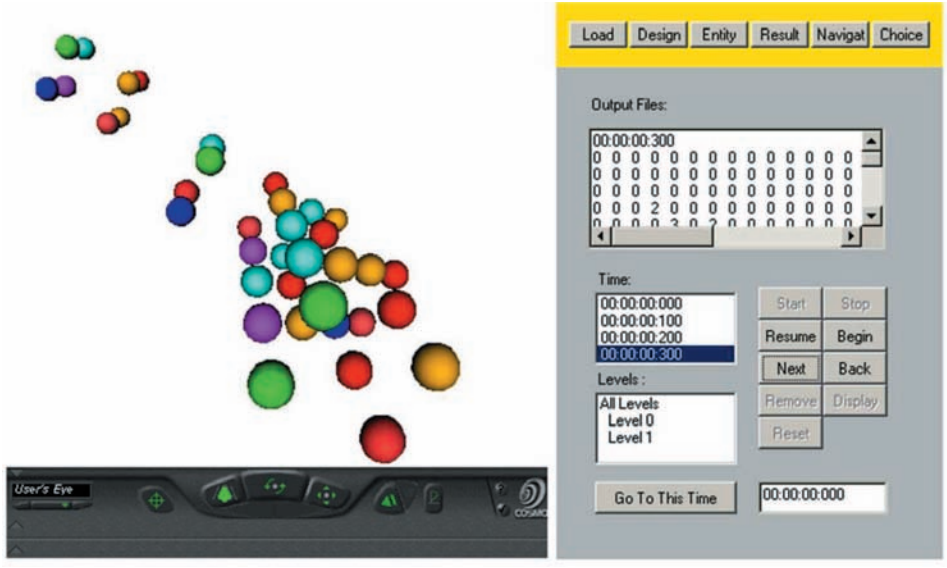

Figure 16. Graphical user interface of the `NavigatePanel`

expressive power to a certain degree. This problem can be addressed with more advanced 3D visualization techniques, as we will discuss in the following sections.

4. Web-based 3D Animation Using VRML

The Virtual Reality Modeling Language (VRML) is a web-based graphics language for describing interactive 3D objects and virtual worlds created using a scene-graph structure. It defines a universal interchange VRML file format, and allows users to interact with a scene through viewpoints, movement, and rotations. A scene graph consists of an ordered collection of hierarchically grouped nodes that represent objects and their properties, generate events in response to environmental changes and user interaction, and participate in event routing mechanisms through which the effect changes are propagated to other nodes. Although VRML has now been superseded by Extensible 3D (or X3D, an open ISO standard for real-time 3D computer graphics), the technique still enjoys widespread use in education and research communities.

Conceptually, every VRML world contains at least a viewpoint from which the world is currently being viewed. Navigation is the action to change the position and/or orientation of the current viewpoint, allowing the user to move through the virtual world or examine an object from different perspectives. A *navigation node* describes the characteristics of the desired navigation behavior, and a *viewpoint node* specifies a predefined location and orientation in the virtual world to which the user may be moved via scripts or browser-specific user interfaces.

We have developed a web-based 3D visualization toolkit called as CD++/VRML using VRML constructions [28]. The visualization starts with an empty VRML root file that is embedded in an HTML web page, representing an empty scene to which the simulation data can be added as child nodes. These nodes can then be manipulated by an external Java Applet according to the simulated model behavior in CD++. The applet acts as the interface between the CD++ environment and the VRML virtual world and provides a set of functions for the user to control the scene. As shown in Figure 16, the NavigatePanel is the main component of the applet. It keeps track of the data currently displayed in the scene, a history of recently navigated nodes, and the name of every displayed node. This information changes whenever the VRML scene is updated during the navigation.

This graphical interface was extended for graphical modeling and visualization of urban traffic. Advanced Traffic Language Specifications (ATLAS) is a language that enables users to specify the topology and detailed constructions of a city section in high-level descriptions and to carry out microscopic traffic simulation using automatically generated executable models [6]. A city section is composed of a set of different constructions representing all kinds of standard elements that can be found in a city landscape. The built-in constructions defined in ATLAS include street segments, parking lanes, crossings (or intersections), traffic lights, traffic signs, railways, and road worksites [29]. There are several inherent advantages associated with this technique. First, users can concentrate on the traffic problem to be solved, rather than being bogged down in the messy details of low-level programming. By decoupling the ATLAS language from the underlying simulation environment, users do not need to have a deep understanding of the M&S theories to con-

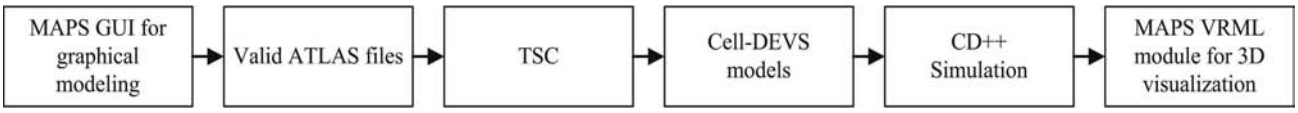

Figure 17. Procedure and techniques for urban traffic modeling and simulation

duct the experiments, significantly lowering the learning curve. Secondly, it provides an open environment where different M&S methodologies can be easily integrated into the toolkit without modifying the models that have already been defined in ATLAS. Thirdly, the city section under study is specified using a formal language and the correctness of the resultant models is validated based on proven formalisms, reducing potential errors in the modeling process. Fourthly, the simulation is performed using a discrete-event approach that can achieve a higher level of precision and simultaneously reduce execution time. Finally, Cell-DEVS provides native support for defining different types of timing constructions, reducing the effort required to build complex behavioral models.

In order to facilitate model construction and support rapid changes in system configuration and rendering, we adapted the CD++/VRML toolkit to include a modeling facility for traffic networks and moving vehicles. The environment, called MAPS [14, 29], allows users to draw city sections directly through a GUI, and automatically translates the imagery into valid ATLAS files, which are then compiled by the Traffic Simulation Compiler (TSC) into Cell-DEVS models for simulation in CD++. It eliminates the need for learning the ATLAS language and reduces the model development time. A VRML-based output module is included in MAPS to reconstruct the specified city section in VRML virtual worlds and to animate traffic flows in realistic 3D graphics according to the simulation results. Figure 17 illustrates the procedure of the M&S process and the main techniques involved at each stage. Key features of MAPS include [14]:

- an intuitive GUI that allows users to quickly draw the traffic network of a city section
- automatically generate the road intersections without user intervention
- automatically create segments for each road without user intervention
- traffic decorations (e.g., traffic lights, signs, etc) can be easily added, changed, or deleted
- parameters of ATLAS constructions can be dynamically modified to adjust simulation configurations
- parse and validate the drawing of a city section into ATLAS files.

As the core class of the software package, ATLASParser is responsible for translating the imagery of a city section into valid ATLAS code. It first removes and stores the crossings of the traffic network and the city-level decorations (e.g. railways), and then loops through the roads to identify the intersections. If an intersection already exists in the saved crossing list, then the user-specified crossing is used. Otherwise, new intersections are automatically created as needed and the roads are cut into appropriate segments, each having its parameters set based on the road configuration (e.g. whether parking is allowed or not, the position at which the road segment crosses a railway line, etc.). The process continues until all the roads and lanes in the imagery are parsed.

Figure 18 shows a screenshot of the MAPS GUI for defining a city section that includes railways (black rectangle with white line), crossings (yellow circles), road worksites (yellow squares), stop signs (red squares), and parking spaces (blue rectangles), in a bidirectional road network. The ATLAS parameters of each construction (e.g. speed limits, curvature of the road) can be modified on the right-hand side property panel, and the corresponding ATLAS code is automatically generated by MAPS. When compared with plain text ATLAS specifications, this graphical representation of the traffic model is more appealing, intuitive, and simpler for users to understand and configure the experiments.

The output module uses the automatically generated ATLAS file to reconstruct a static scene of the city section in VRML virtual world. It determines the location and direction of each vehicle at a particular point in time based on the CD++ simulator. A 3D car shape is displayed on the screen at the appropriate cell of a road segment for the time duration as indicated by the simulation. In this way, the animation shows a microscopic view of the traffic flow on the road network. Users can navigate through the virtual city section using the VRML navigation panel introduced in Figure 16. The 3D visualization is achieved by showing the ATLAS constructions and vehicles with VRML nodes. Different shapes are used to represent the vehicles and various constructions such as the cells of a road segment, crossings, and traffic signs. A one-to-one correspondence is established between the Cell-DEVS model components, the ATLAS constructions, and the VRML visual objects. In order to visualize the ATLAS file properly, the output module needs to calculate the length and rotation angle for each road segment and then place it at the appropriate position in the VRML scene. Once the length and angle are determined, the out-

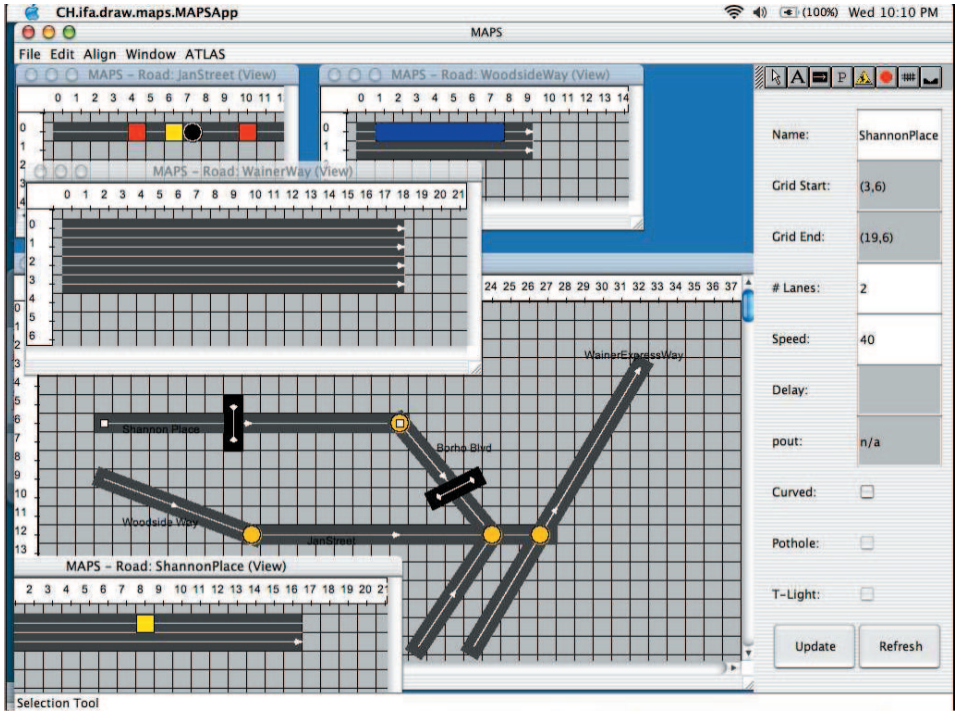

Figure 18. A screenshot of the MAPS GUI [14]

put module maps the segment to a visual object that is stretched from the given start point, and scaled and rotated according to the calculated values. Figure 19 exemplifies a static view of the road network on Carleton University campus. The output module animates traffic flows in a 3D VRML virtual world by parsing the log file generated in CD++. The entering of a car in a particular cell is detected by an output message (Y) with value 1 sent from the cell, while the departure of a car is indicated by a Y message with value 0. In the former case, the output module creates a VRML car shape and aligns its orientation with respect to the hosting cell. In the latter case, the car shape is removed from the cell and the log file is searched to find the corresponding entering event scheduled at the destination cell. The car shape will be put at the new location after the delay time as given in the Y message.

5. Advanced Techniques for Visualization of DEVS and Cell-DEVS models

Although the CD++/VRML toolkit provides a visual M&S environment that empowers users to create sophisticated 3D graphics for data exploration in scientific and engineering applications, it also has some limitations. First, it relies on the obsolete VRML standard that is no longer widely supported due to, among other things, its lack of advanced features, large file size, and relatively low frame rates. Secondly, it supports 3D animation of Cell-DEVS models only, considerably restricting its applicability in general DEVS-based systems. To overcome these drawbacks and to meet the diverse needs of different users, we have developed mechanisms to integrate the CD++ environment with a variety of commercial and open-source visualization and rendering techniques, including Autodesk Maya [11], OpenGL [12], and Blender [13]. In this section, we will elaborate on these advanced techniques and demonstrate their capabilities with a wide range of applications.

5.1 CD++/Maya – A High-Performance 3D Visualization Engine for CD++

Autodesk Maya [11] is one of the leading commercial software packages for 3D modeling, animation and visual effects. Maya software interface is fully customizable and allows users to extend their functionality by providing access to the Maya Embedded Language (MEL). With MEL, users can tailor the GUI to fulfill their specific needs and to develop in-house tools. The MEL scripting language has been used in our research to create a high-performance 3D visualization engine, referred to as CD++/Maya, for DEVS and Cell-DEVS models [30], allowing for interoperability between a DEVS-based M&S tool and an advanced generic visualization environment like Maya.

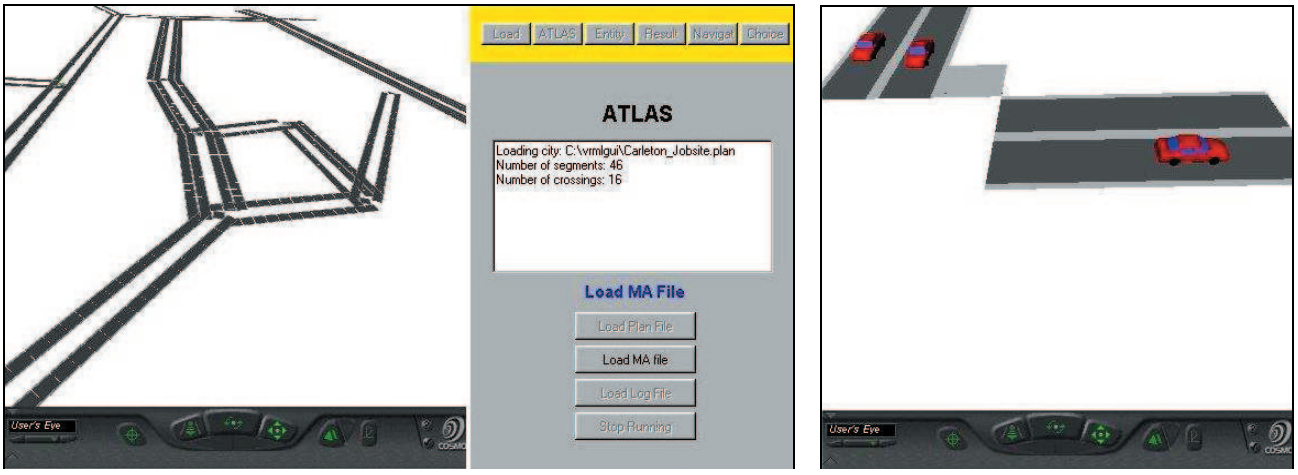

Figure 19. A static view of the road network on Carleton University campus [14]

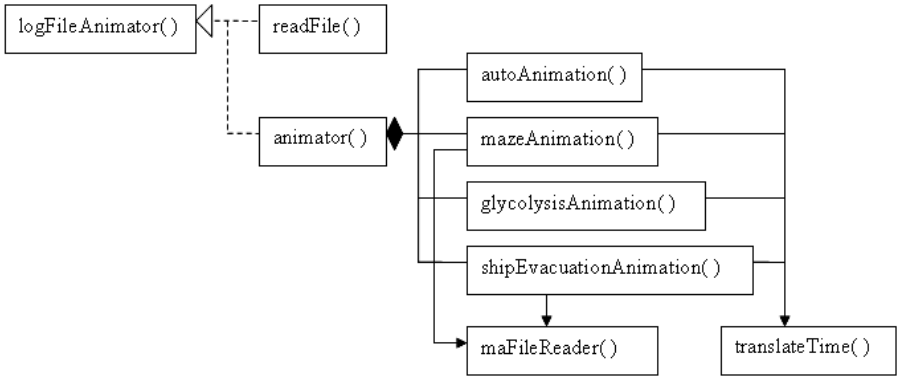

Figure 20. Major modules defined in the CD++/Maya visualization engine [30]

Users create a static scene in Maya, providing the necessary background for 3D animation of the simulation results. As usual, the specifications of DEVS and Cell-DEVS models under study are submitted to the CD++ environment, which generates a stream of events during the simulation. CD++/Maya loads and initializes the predefined scene file based on the model definition, and overlays 3D animation on top of the static scene according to the event data. The development of CD++/Maya follows a modular approach. Figure 20 shows the major modules defined in the visualization engine [30].

The logFileAnimator module serves as an interface between CD++ and Maya. Figure 21 shows the GUI where users can specify the log file and choose among available scene files. Two modes are available for analyzing the simulation data: *direct analysis* and *animation*. The former allows advanced users to take a close look at the content of log files for debugging and model verification purposes, while the latter presents an intuitive 3D animation for the selected model.

The animator module reads the simulation results, creates Maya objects based on the event data, and runs the animation in the 3D scene. Similar to CD++/VRML, a one-to-one correspondence is established between the model components and the Maya visual objects. Users can create customized animations using the MEL scripting language. In the following sections, we show the capability of CD++/Maya using different modeling examples, focusing on the visualization aspects of the M&S process. Interested readers can find the detailed model definitions in [30–32].

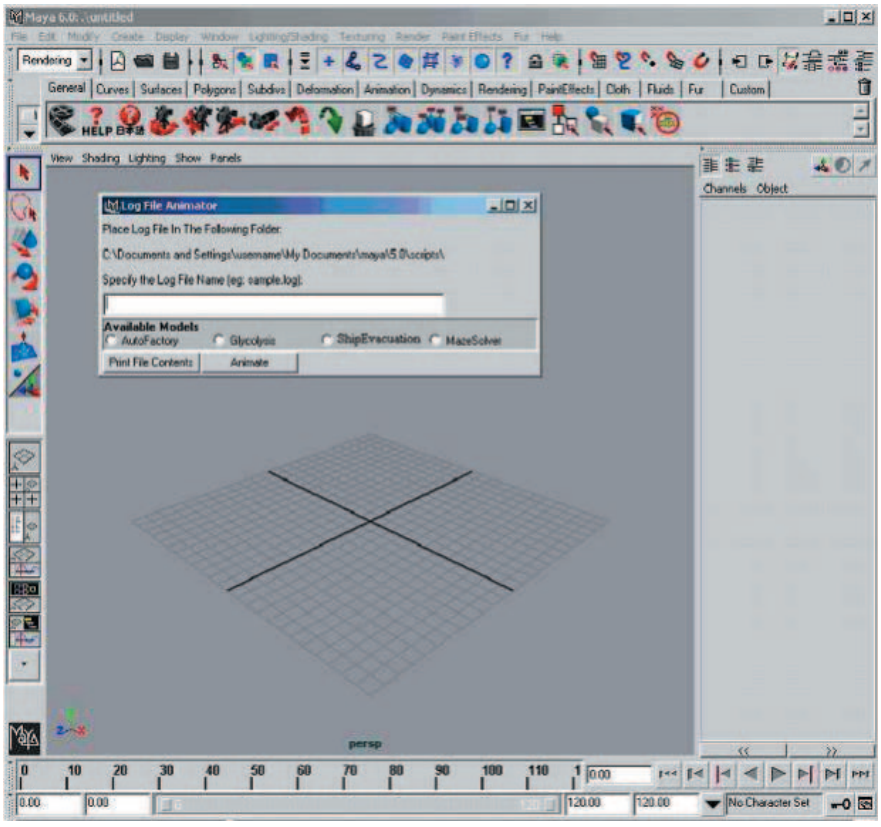

Figure 21. The dialog window for the **`logFileAnimator`** module [30]

5.1.1 Maze Solving

Our first example focuses on a Cell-DEVS model that implements a maze-solving algorithm [33]. The algorithm is coded as a set of rules that effectively block off every dead-end path (i.e., a path that is accessible from only one direction) in the maze so that in the end only those cells that form the final solution remain unblocked when the system becomes stable. A 2D visualization can be defined in CD++Modeler. However, users can gain more insight and understanding of the model behavior by exploring the multiple viewpoints provided in CD++/Maya, as seen in Figure 22.

5.1.2 A Car Manufacturing Model

We have developed a DEVS model that simulates a car factory trying to maintain a suitable production level by coordinating the operation of its various assembly lines. The automobile parts are produced by four assembly lines respectively (i.e. chassis, body, transmission case and engine). A car is then assembled using one component from each of these assembly lines. The structure of the model is depicted in Figure 23(a). A log file like the one shown in Figure 23(b) is generated when the model is executed by the CD++ simulator. The input and output trajectories of the DEVS model can be animated in 2D using CD++Modeler as discussed in Section 3. Although this allows a detailed analysis of the simulation data, it is still rather abstract and elusive for general users to have an intuitive and global comprehension of the simulated phenomenon. To increase the degree of realism of the visualization, we animate the car manufacturing model in CD++/Maya. Figure 24 illustrates two snapshots of the animation at different virtual times.

In the log file, the availability of an automobile part is represented as a Y message sent from the corresponding assembly line to the final assembling warehouse. Such an activity is shown in the animation as a 3D icon that stands for the specific automobile part moving between the entities in the virtual world. For example, the animation in Figure 24(a) shows that three auto parts are made available at virtual time 02:000 and Figure 24(b) shows a finished car moving out of the assembling warehouse at virtual time 04:000.

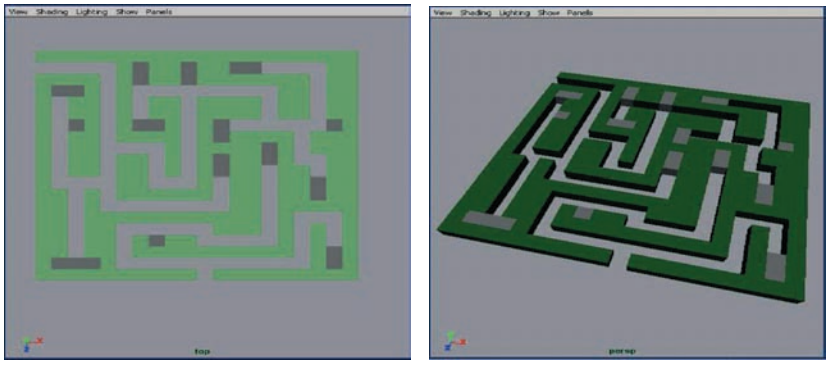

Figure 22. Maze-solving results and CD++/Maya animation from different perspectives [30]

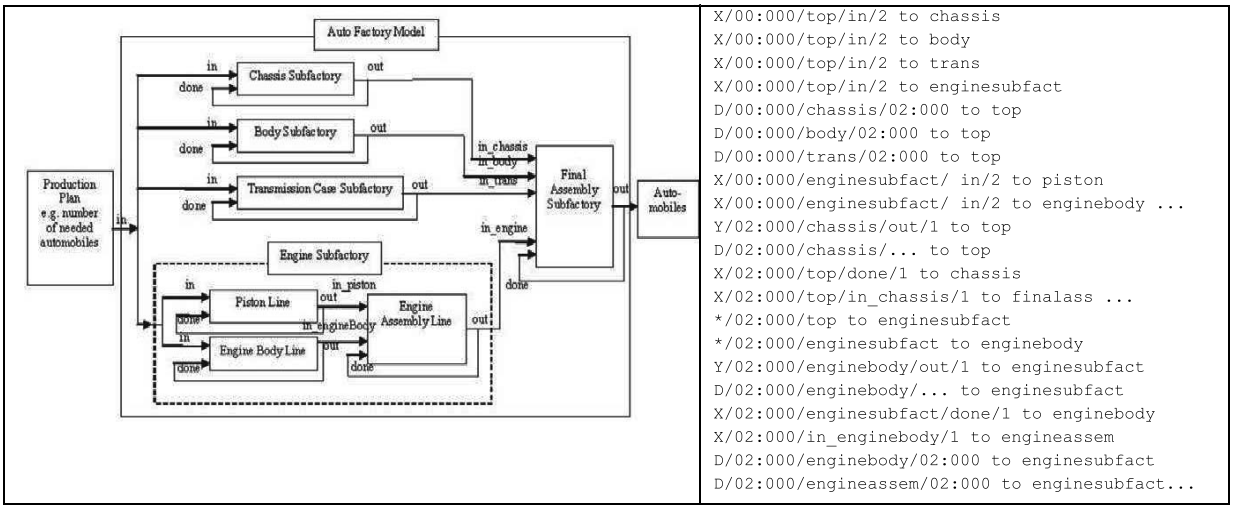

Figure 23. A car manufacturing model and the generated log file [30]

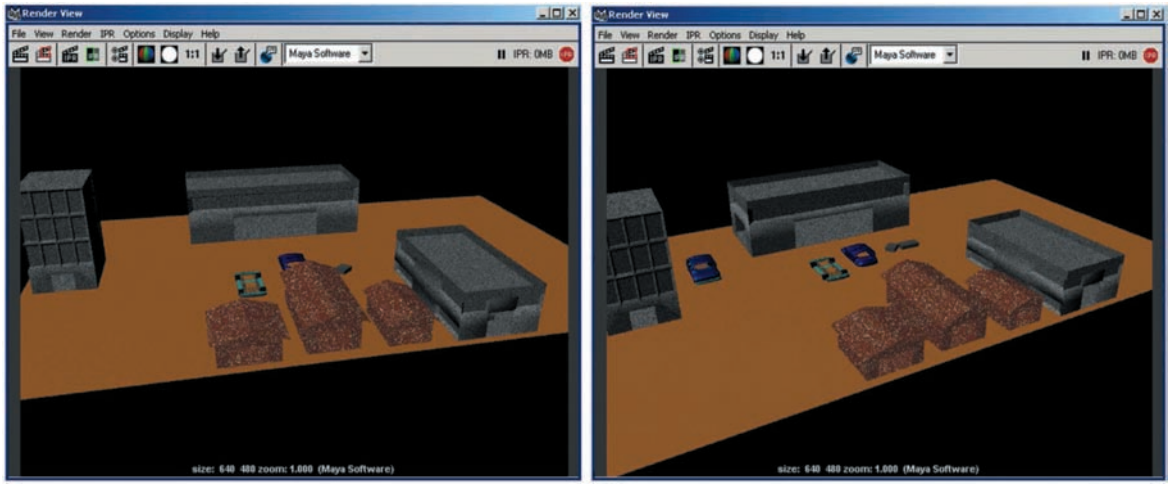

Figure 24. Animation of the car manufacturing model in CD++/Maya [30]

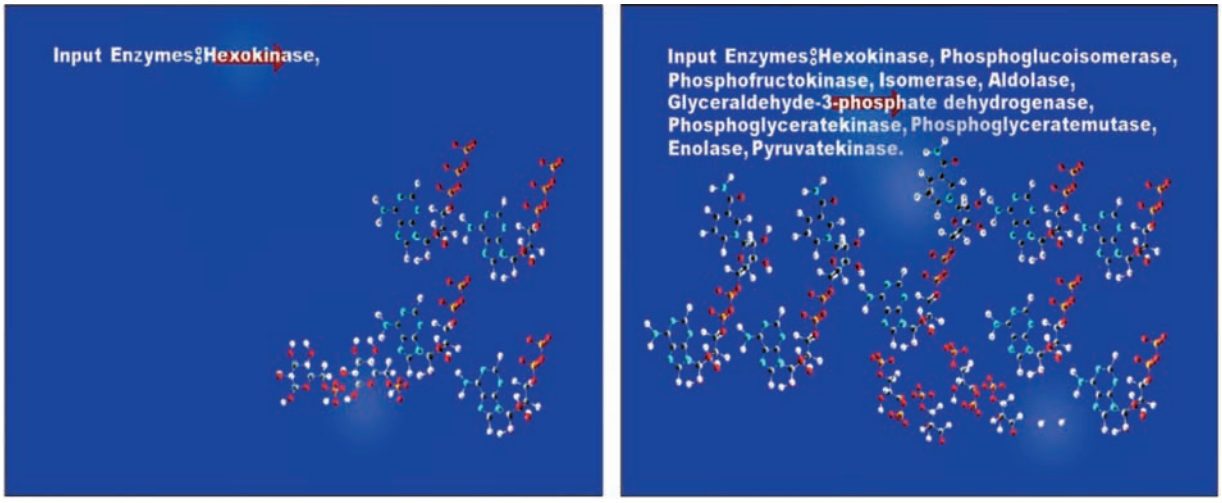

Figure 25. Animation of the glycolysis process in CD++/Maya [30]

5.1.3 Glycolysis and Krebs Cycle Pathways

Molecular visualization plays a central role in many chemistry and biology researches due to its effectiveness in revealing information on complex molecular structure and dynamics. In [34], we developed a DEVS model of different pathways in the cell (namely, glycolysis and Krebs cycle), representing the sequence of reactions occurring in the process of breaking down one glucose molecule into two pyruvate molecules. Ten steps are involved in the process, which can be divided into two phases. In the first phase a glucose molecule is converted into two glyceraldehyde-3-phosphate molecules (GDP), which are then converted into two pyruvate molecules in the second phase. Each step was defined as a DEVS atomic model, and a DEVS coupled model was constructed by connecting all the atomic components to model the process. The simulation results were then visualized with CD++/Maya. Figure 25 illustrates the animation of the two steps occurring in the simulation. Figure 25(a) shows the end of step 1 when two alpha-gluco-phospates (G-6-P) and two adenosine diphosphates (ADP) are formed. Figure 25(b) shows step 6 that begins at the presence of three molecules of nicotinamide adenine dinucleotide (NAD+).

5.1.4 Evacuation under Emergencies

In recent years, numerous models for emergency response solutions have been proposed, with the goal of modeling human behavior in the event of an emergency evacuation. We created a 3D Cell-DEVS model to simulate the behavior and movement of every person in a crowd during an evacuation. The model employs orientation information defined by a parameter, called as *distance potential*, which is used to guide individuals towards the emergency exits. Normally, a person will move in the direction decreasing the distance potential (i.e. moving towards the exits). However, some people may move in the wrong direction under stress and panic, which is controlled by a given *panic level*. People move at different speeds, and collision detection mechanisms are included.

We executed experiments for different configuration settings (number of people involved in the evacuation, number of exits, and various combinations of the parameters). Figure 26(a) shows a segment of the 2D animation generated with CD++Modeler, in which a person is denoted as a red dot and the two layers of the model have to be displayed separately. In Figure 26(b), we illustrate the Maya scene file created specific for the evacuation model. The static scene constitutes a realistic visual model of the building under study and provides the background on which 3D Maya animation can occur. Figure 26(c) shows the 3D animation at different virtual times, and observed from different viewpoints. Each human figure is represented by a 3D avatar. Similar to CD++/VRML, the visualization engine retrieves CD++ simulation data, and relocates the avatars based on the coordinates and values of the cells. The resulting frame-based motion of human figures allows users not only to easily trace each individual person throughout the building, but also to gain deeper insight into the evacuation process as a whole. Likewise, the 3D rendering gives more details about the building evacuation than the symbolic 2D animation in CD++Modeler. This enables users to quickly identify potential bottlenecks and choke points in the building, to plan and test different evacuation strategies, to evaluate a variety of alternative structures and configurations of the building, and most importantly, to achieve all these goals quickly and at low cost. Finally, these models can be used for training pur-

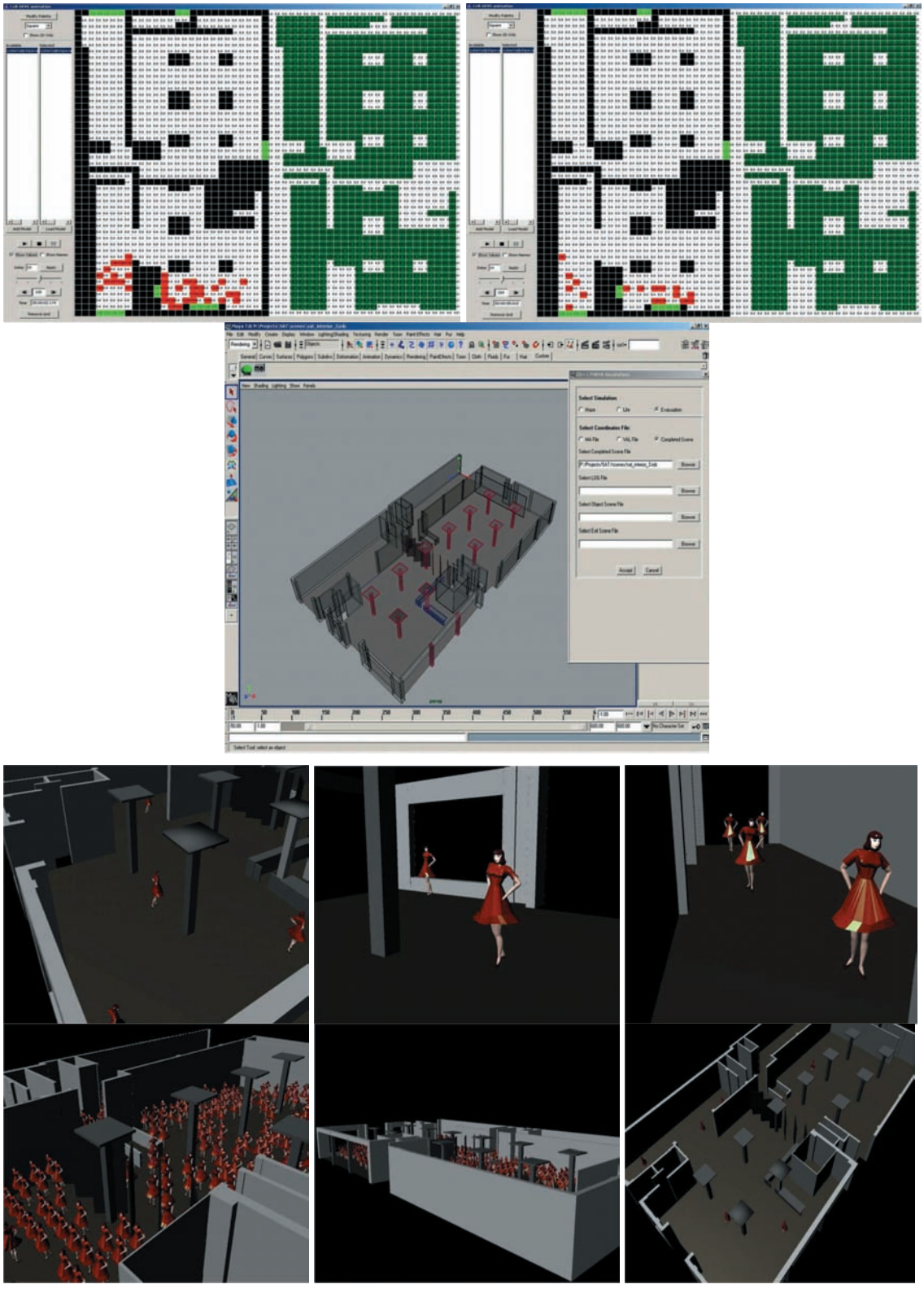

Figure 26. Animation of the emergency evacuation model using CD++Modeler and CD++/Maya

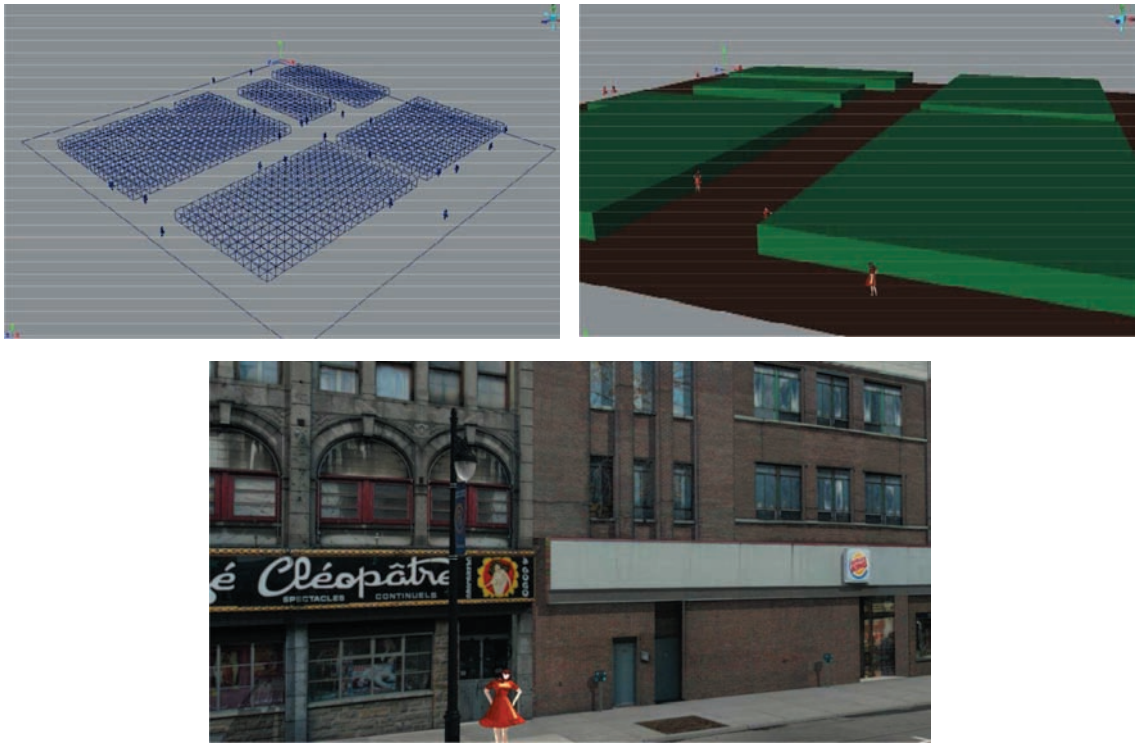

Figure 27. Animation in Maya/CD++ from different viewpoints and layers

poses as they can be easily incorporated in immersive environments with specialized 3D visualization capabilities.

CD++/Maya toolkit provides an integrated environment that allows for seamless interoperability between DEVS-based simulation toolkits and high-performance visualization facilities. It enables users to analyze massive simulation data in an intuitive and efficient fashion with increased realism, responsiveness (or interactivity), and immersion, representing a step forward in advancing model-based design and development of state-of-the-art virtual reality systems. Figure 27 shows an extension to the model introduced in Figure 26 that illustrates the evacuation of an area in downtown Montréal (where the previously simulated building is located). These simulated results were automatically generated from 3D geographical data of the area, combined with the previously defined evacuation model, showing that a user can easily create 3D versions of the model to gain deeper insight. The final figure shows the integration of these results with the Eucalyptus platform (a distributed environment for participatory design using 3D visualization for architecture), showing the potential of this application [32, 35].

5.2 DEVSView – An OpenGL-Based Tool for Visualization of DEVS and Cell-DEVS Models

CD++/Maya is a powerful visualization engine for creating advanced 3D animation of DEVS and Cell-DEVS models. However, it is only an ideal environment for large projects where high performance is the major concern. The cost associated with this toolkit can be prohibitive for small-scale applications (in terms of large software installation size, hardware requirements for adequate rendering, licensing costs, etc.). In order to provide an alternative visualization method for cost-sensitive users, we have developed an open-source toolkit, referred to as DEVSView, for visualizing CD++ simulations based on standard OpenGL techniques [36]. OpenGL [12] is a standard specification for developing cross-platform and language-neutral applications that utilize vendor-supplied hardware acceleration in 2D and 3D computer graphics. It has been widely used in a broad range of industries, including Computer-Aided Design (CAD), virtual reality, flight simulation, and video games. GLUT (OpenGL Utility Toolkit) is a cross-platform library for writing portable OpenGL programs [37].

DEVSView provides a generic method for mapping simulation results into a visual representation. Visualization in DEVSView consists of visual models that are directly translated from the atomic and coupled components

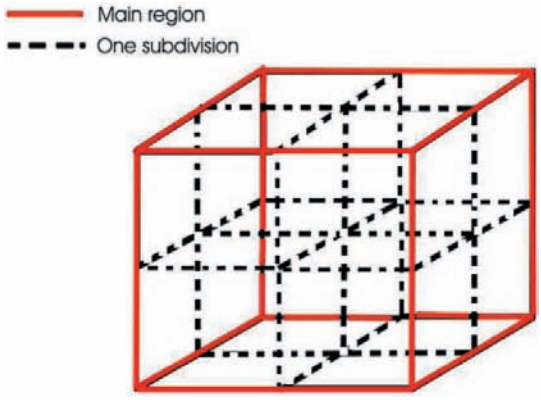

Figure 28. Partition of 3D space in oct-tree [36]

of a DEVS or Cell-DEVS model, establishing a one-to-one relationship between visual models and their corresponding simulated components. The animation is executed in an event-driven fashion, and each visual model uses a state transition and an event animation system that can be manipulated by a set of events, corresponding to the external and output messages generated in CD++. The key features of DEVSView are summarized as follows:

- it is a GLUT-based GUI that provides all necessary controls to define and playback 3D animations
- it includes customizable visualization of DEVS and Cell-DEVS models by associating each visual model with a state transition system that comprises user-defined visual states and transition rules
- it defines an event animation system for each visual model to create customized visual effects upon the arrival of certain events
- it provides efficient view culling through an improved oct-tree algorithm [38]. An oct-tree is a data structure that represents a 3D space by recursively subdividing it into eight subspaces. The visual objects in a 3D scene are assigned to the smallest (fittest) regions that can contain them completely. Each region is implemented as a node in the tree. The initial space and the division of its immediate subspaces are illustrated in Figure 28. The proposed culling algorithm can accurately match a graphical object to its fittest region, significantly improving the rendering performance.

At the beginning of the visualization, DEVSView parses the CD++ results to create a visual model for each DEVS and Cell-DEVS model component. The visual models are then customized to follow a visual state transition system and/or to produce animations for certain events. The animation is carried out by sending the events to the source/destination visual models that are involved in the message exchanges. An event contains information about the source and destination visual models, the input and output ports through which the event is sent, the simulated virtual time of the event, and the value of the event. Based on this information, the state transition rules determine how an arriving event will affect the appearance of the visual model, and the event animation rules decide what kind of animation will be produced for the event itself. Formally, a visual model is defined by the following basic elements:

- a unique name (or label)
- a list of input and output ports to exchange events with other visual models
- a location in the 3D coordinate system, an orientation, and a non-zero size
- a list of visual states, one of them is marked as current and determines the visual appearance of the model
- a visual state transition system defined as a state machine that consists of visual states, and the transitions between them, as determined by a list of state transition rules
- an event animation system that generates visual effects for certain events based on a list of user-specified event animation rules.

Figure 29 shows an example visual model called pinver (Pin Verifier) for the animation of an Automatic Teller Machine (ATM). The visual model is currently in its idle state and hence displayed as a red 3D box, which is specified in the state transition rules. Users can edit the properties of a visual model (e.g., visual state, shape, color, label).

For Cell-DEVS models, all the cells in a cell space share the same list of visual states, state transition rules, and event animation rules, even though they may have different current states, positions, orientations, and sizes. Sharing common information among the cells reduces file size and memory consumption, facilitates visual model definition, and improves animation performance.

The visual state transition system and event animation system play a central role in the visualization. They process the events received by a visual model to generate the desired animation based on user-defined rules. The granularity or detail of the animation is controlled by a mechanism (called the *value rule*) that acts as a filter for the incoming events. A value rule allows an event to be passed to the downstream state transition and event animation rules only if the event value satisfies certain criteria. By controlling the value rules, users can focus only on the animation of those events of particular interest, while removing the other events of less importance from the scene.

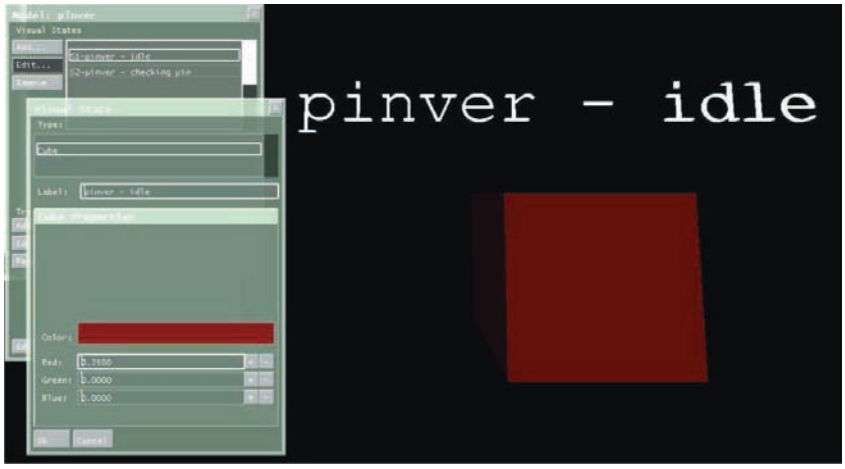

Figure 29. An example visual model and the state editing panel [36]

Three types of value rules are currently supported in DEVSView [36]:

1. all values: a value rule of this type does not impose any filtering
2. equal value: a value rule of this type allows an event to pass only if its value is equal to a user-specified constant
3. range of values: a value rule of this type passes an event if its value is within a predetermined range.

Upon arrival of an event, the visual state transition system evaluates the rules associated with the current state to determine if any should be invoked. Each transition rule has four properties, including a unique ID (per visual model), a port name and type (i.e. input or output), a value rule, and the next state. The state transition system triggers a rule if the incoming event can pass the value rule and it matches the port and direction of the transition rule. As a result, the visual model is changed to the next state as specified in the transition rule, and the visual appearance of the model is altered accordingly. In addition, the state transition is recorded in the visual model, providing a history data buffer for fast animation playback if needed.

The event animation system allows visual models to animate the processing of certain events. The animation is triggered when an arriving event satisfies certain criteria as defined by the event animation rules. This provides a convenient mechanism for users to express the semantic of the event that gives the reasons why visual state transitions occur. For example, using only a visual cue for the secure login component of our ATM (which is continuously oscillating between the data-acceptance and the data-validation states) does not give users enough information to figure out the model's behavior. There may be

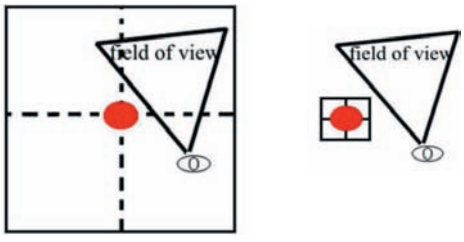

Figure 30. Illustration of the view culling algorithm in DEVSView [36]

many possible reasons for such oscillations (i.e., invalid card, invalid password, insufficient privilege). A simple event animation displaying 'invalid password' at the secure login model would clearly indicate the reason for the behavior. Each event animation rule has five properties, including a unique ID (per visual model), a port name and direction (i.e. input or output), a value rule, the source state, and the animation length. An event animation rule is fired if the current state of the visual model is the same as the source state, and the arriving event passes the value rule and matches the port and direction of the event animation rule. Animations are created based on the information of the event and/or other internal variables in the visual model.

Complex 3D scenes containing many different visual objects may require a significant time to render. Determining which objects are needed to be refreshed in a frame is important for efficiency and performance. View culling is the process of calculating which objects are currently in view and therefore require rendering. To do so, we use an oct-tree data structure, which is traversed to check whether a node is in the current view or not. If a node

```
Let ON = the current oct-tree node
Let ParentON = the parent oct-tree node of ON
Let VF = the view frustum (i.e. the field of view)
If ParentON intersects VF {
       Calculate ON visibility
       If ON is not visible {
                Stop traversing ON
       }
}
Else if ParentON is completely visible {
       ON is therefore completely visible
}

If ON is completely visible or intersects VF {
       Draw each scene node contained in ON
}
If ON has children oct-tree nodes {
Draw each child oct-tree node by recursively invoking this algorithm
with ParentON = current ON, and new ON = child oct-tree node.
}
```

Figure 31. Pseudo code of the view culling algorithm [36]

is out of view, the entire sub-tree rooted at that node can be pruned. On the other hand, if a node is completely in view, then the whole sub-tree extending from that node is visible (and no further visibility check will be performed for its descendents). We defined an efficient culling algorithm that is able to assign visual objects to several oct-tree regions to better define the outline of the objects. Consider a small object located at the centre of the root region, as shown in Figure 30(a). Since the object is associated with the root region, it can be culled only when the root node is pruned, despite the fact that it may rarely be in view. If the small object can be added to the eight fittest regions that contain it completely, as illustrated in Figure 30(b), then the scene will be culled much more efficiently. Figure 31 gives the pseudo code of the culling algorithm.

DEVSView was developed in three main parts: a parser that extracts the atomic and coupled model components (as well as the events in CD++ simulations), a GUI that provides animation control mechanisms (and allows users to specify the graphical representation of the visual models), and a scene database that can efficiently organize visual models in the 3D space. Figure 32 shows the package diagram for the DEVSView toolkit.

DEVSViewerDisplay is responsible for converting user inputs into commands that can be processed by ViewerControl. DEVSViewerDisplay also controls the rendering of the GUI and all 3D objects. It utilizes the services

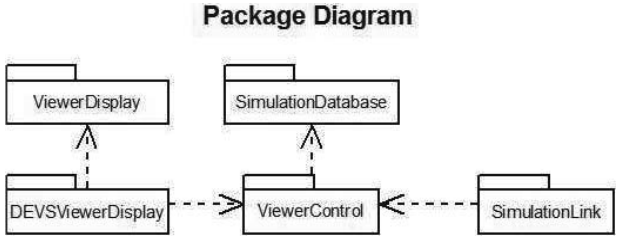

Figure 32. Package diagram for the DEVSView toolkit [36]

provided by ViewerDisplay for event-driven functionalities. SimulationLink serves as the linkage between DEVSView and the simulation environment. It parses the simulation results and notifies ViewerControl about new events and visual models. ViewerControl processes the requests from DEVSViewerDisplay and SimulationLink. It translates the requests into a proper sequence of interactions with the SimulationDatabase, which stores the events, visual models, and other related information for the visualization. These packages communicate with each other by passing data of various types, including the standard C++ types, several basic structures for expressing position and time information, and property sets containing the attributes of different entities. Further details of the design and implementation of these packages can be found

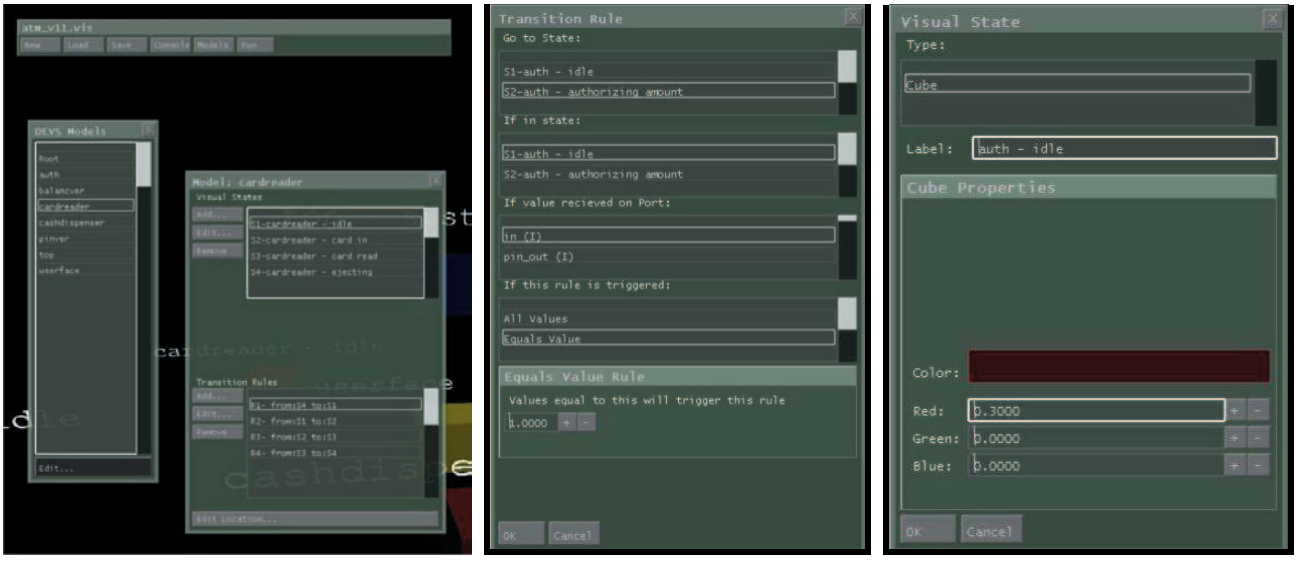

Figure 33. Major GUI components: (a) Model Edit; (b) Transition Rules; (c) Visual State [36]

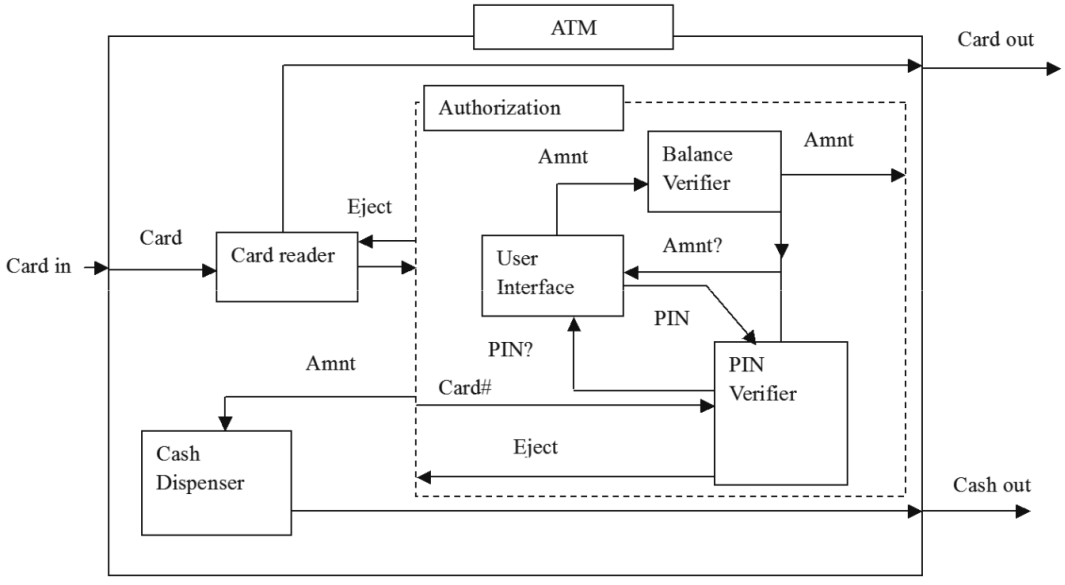

Figure 34. Structure of a DEVS model for an ATM banking machine [7]

in [36]. Figure 33 shows the major GUI components of the DEVSView toolkit. The capability of DEVSView is demonstrated with two example models. Figure 34 gives the structure of a DEVS model that simulates an ATM, and a snapshot of the animation in DEVSView is shown in Figure 35.

Figure 35(a) shows the animation when a customer inserts a debit card into the ATM machine. The event animation is shown as a 3D-text effect 'Card Inserted' just beside the CardReader visual model. Accordingly, the CardReader model transitions to the 'card in' state and the top model changes to the 'customer in system' state, while all other models are idle. The oct-tree regions allocated to the visual models are outlined in Figure 35(b). Figure 35(c) shows the animation of a Cell-DEVS model that represents the movement of three bouncing balls in

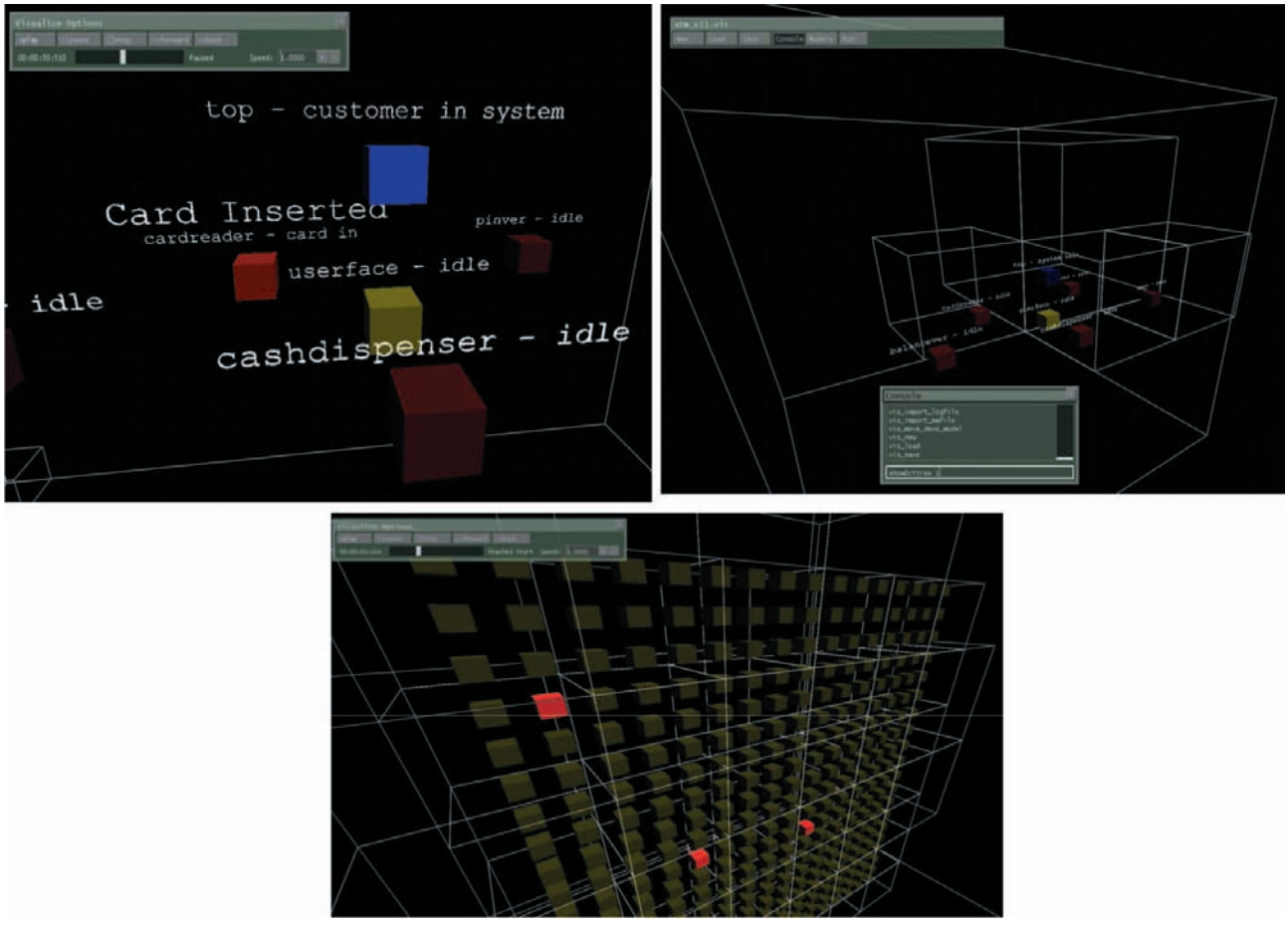

Figure 35. DEVSView animation of the ATM and Bouncing Ball models [36]

a closed area. We can see the objects bouncing back and forth in a closed container, which shows the use of non-toroidal cellular models where the cells on the border have different behaviors from the rest of the cells.

5.3 CD++/Blender

Blender [13] is an OpenGL-based freely available 3D modeling and animation software package being actively developed and widely used in a broad array of industrial applications. It has a mature and robust feature set that is similar in scope and depth to other high-end 3D applications such as 3ds, Max and Maya. A fairly comprehensive feature comparison of existing 3D applications can be found in [39]. Using the Blender software suite, we developed another open-source visualization toolkit, known as CD++/Blender, in an attempt to combine the advantages of CD++/Maya and DEVSView in an integrated environment. CD++/Blender has a relatively small installation footprint and can be used to create advanced animation of complex DEVS and Cell-DEVS models on various computing platforms. It uses Python scripting language to parse CD++ simulation results, and generates customizable visualization based on the same design as DEVSView. The design and implementation of the toolkit will not be reiterated in this section, as it is based on the previous ones. Instead, we demonstrate the capability of CD++/Blender with several examples.

Figure 36 shows a screenshot of the same model presented in Figure 27 using CD++/Blender. Users can specify model definition and CD++ log files using the buttons on the left panel. The animation is shown in the main animation window based on a predefined 3D scene file as well as the events executed (the current status of the animation can also be displayed in a floating text console). Users can customize the animation and navigate the 3D scene through the control panel underneath. One of Blender's great strengths is that the GUI is entirely drawn in OpenGL and the content of every window can be

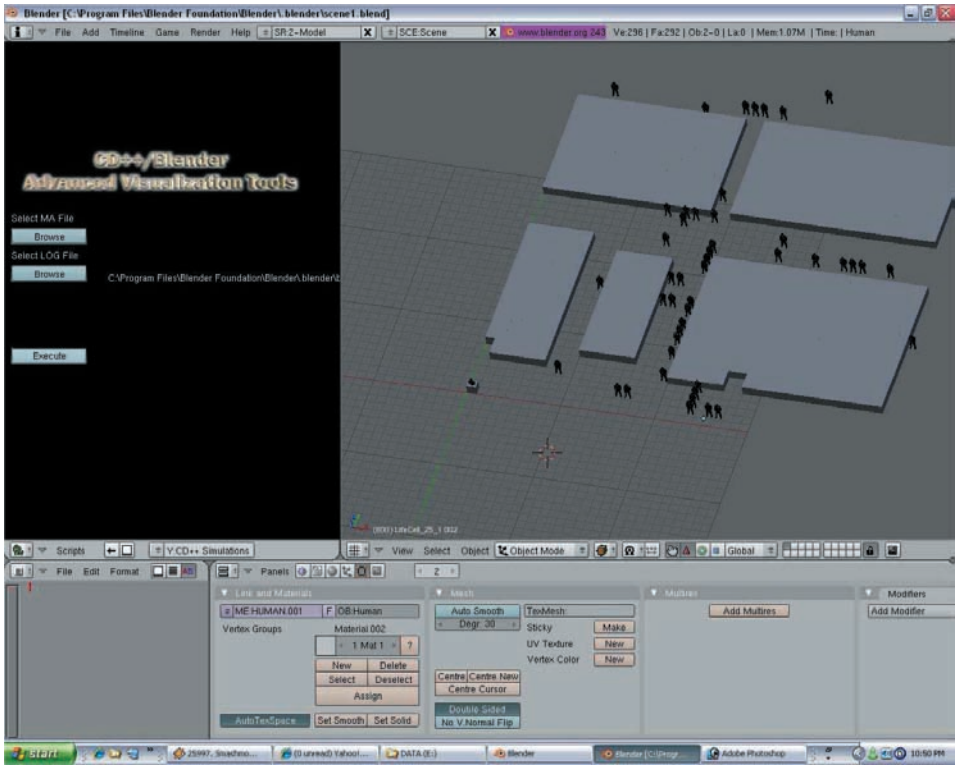

Figure 36. The CD++/Blender graphical user interface

panned, zoomed, and moved around just like other visual objects so that the screen can be organized to the user's taste for each specialized task.

Figure 36 shows how to create a 3D scene file in CD++/Blender. For each scene, users can create the props, dress and paint them with different materials and textures using the predefined window layouts in Blender. It is also possible to define multiple scenes within a single Blender file, allowing them to share and reuse common visual objects to reduce file size. With just a few button clicks, users can add various visual objects into the scene, ranging from simple shapes to sophisticated 3D avatars, as illustrated in Figure 38. Users can run the animation forward or backward and examine the simulation results from different viewpoints by navigating in the 3D virtual world. For large models, 3D scenes may become exceptionally confusing due to the increased complexity. This problem is solved in CD++/Blender by virtue of Blender's native support of layers. Each layer in the scene groups together the related visual objects of interest so that only the selected layers (or groups of visual objects) are rendered at any one time. This technique provides a better overview of the animation and allows user to examine the simulation data with varying granularities. As a high-end visualization engine for the CD++ environment, the open-source CD++/Blender toolkit achieves a level of 3D animation performance comparable to CD++/Maya, yet at a low cost, and opens the possibility for analyzing increasingly complex simulation data in an efficient and intuitive manner.

6. Conclusion

We have presented a detailed discussion of multiple graphical visualization facilities currently available in the CD++ environment. The objective of our research is to provide general users with a variety of easy-to-use toolkits to facilitate the M&S process and thereby promote the adoption of cutting-edge M&S technologies by a wider community of practitioners and researchers. A state-based modeling paradigm was introduced to allow for the definition of rather complex model behavior using DEVS Graphs. Based on the proposed graphical notations, a 2D modeling and visualization engine called CD++Modeler has been developed in our research to enable application specialists who do not have much experience in computer programming to easily construct models and to analyze simulation data using an intuitive graphical user interface. In this paper, we also summarized and reviewed a variety of mechanisms that have been employed to interface CD++ with sophisticated commercial and open-source 3D visualization and rendering

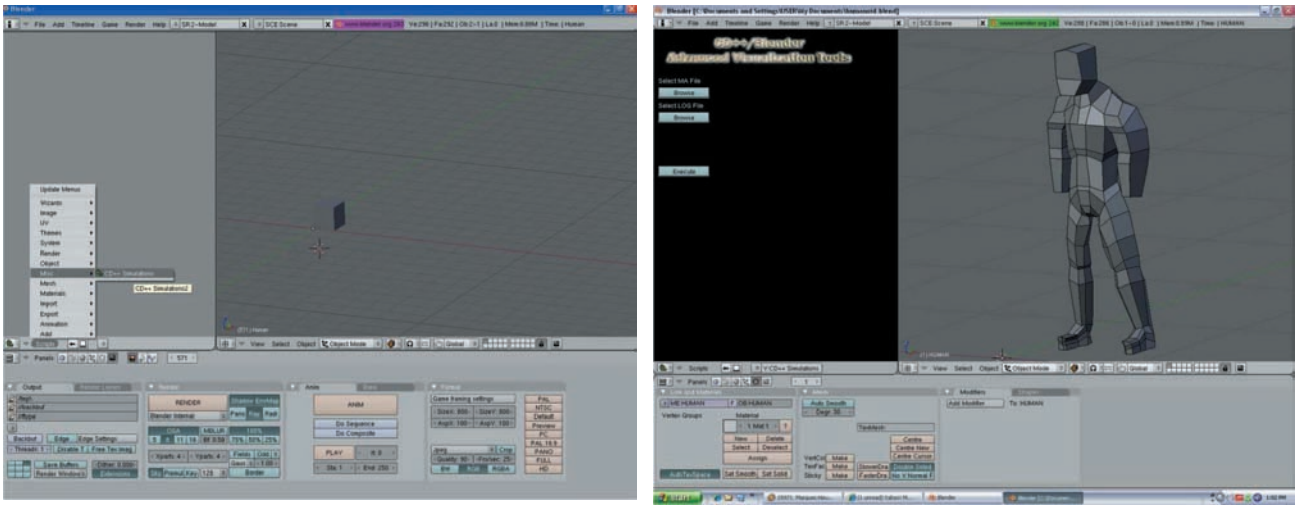

Figure 37. Creating 3D visual objects in CD++/Blender

techniques to fulfill the needs of different user communities, including CD++/VRML, MAPS, CD++/Maya, DEVSView, and CD++/Blender. The major characteristics and design considerations of the CD++ family of visualization toolkits were outlined and their relative merits and limitations were compared with various realistic applications. As demonstrated in the examples, these toolkits allow users to navigate and interact with the animated 3D model from different viewpoints for efficient investigation of simulation data. The fully automated M&S process significantly reduces the time and cost for model development and system testing as well as the learning curve for general users. We are currently working on the interface between these tools and existing GIS systems to automatically import graphical geographic data into the CD++ environment, further reducing the modeling effort. Research has also been carried out to integrate CD++ with advanced immersive virtual reality techniques in order to improve user experience in situations such as emergency response planning, interactive training, and serious gaming. As seen in our multiple experiences, the use of a modular architecture provides a framework that can be easily extended to incorporate other modeling and visualization techniques in future development. We have shown that these techniques can significantly facilitate the comprehension of continuously evolving models, making them suitable for efficient online decision making.

7. Acknowledgments

This work has been partially funded by the Natural Sciences and Engineering Research Council of Canada, the Canadian Foundation for Innovation, the Ontario Innovation Fund and CANARIE Inc. The architectural models were provided by Professor Michael Jemtrud and Jim Hayes. Numerous students participated in this project in particular Shannon Borho, Juan Ignacio Cidre, Ayesha Khan, Emil Poliakov and Wilson Venhola.

8. References

[1] Zeigler, B.P., H. Praehofer, and T.G. Kim. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems.* Academic Press, San Diego, CA.

[2] Wainer, G., and N. Giambiasi. 2002. N-dimensional Cell-DEVS Models. *Discrete Event Dynamic Systems* 12(2), 135–157.

[3] Wainer, G. 2002. CD++: A Toolkit to Develop DEVS Models. *Software: Practice and Experience* 32(13), 1261–1306.

[4] Wainer, G. 2004. Modeling and Simulation of Complex Systems with Cell-DEVS. In *Proceedings of the 36th Winter Simulation Conference*, Washington, DC, 49–60.

[5] Wainer, G. 2006. Applying Cell-DEVS Methodology for Modeling the Environment. *SIMULATION: Transactions of the Society for Modeling and Simulation International* 82(10), 635–660.

[6] Wainer, G. 2006. ATLAS: A Language to Specify Traffic Models using Cell-DEVS. *Simulation Modelling Practice and Theory* 14(3), 313–337.

[7] Wainer, G. 2009. *Discrete-Event Modeling and Simulation: a Practitioner's approach*. Taylor and Francis. [In Press].

[8] Farooq, U., G. Wainer, and B. Balya. 2006. DEVS Modeling of Mobile Wireless Ad Hoc Networks. *Simulation Modelling Practice and Theory* 15(3), 285–314.

[9] Fishwick, P.A. 2004. Toward an Integrative Multimodeling Interface: A Human-Computer Interface Approach to interrelating Model Structures. *SIMULATION: Transactions of the Society for Modeling and Simulation International* 80(9), 421–432.

[10] Chidisiuc, C., and G. Wainer. 2007. CD++Builder: An Eclipse-based IDE for DEVS Modeling. In *Proceedings of the 2007 DEVS Integrative M&S Symposium (DEVS'07)*, Norfolk, VA.

[11] Autodesk Maya Press. 2007. *Learning Autodesk Maya 2008: The Modeling & Animation Handbook*. John Wiley & Sons Inc., Hoboken, NJ.

[12] Segal, M., and K. Akeley. 2006. *The OpenGL® Graphics System: A Specification, Version 2.1*, http://www.opengl.org/registry/doc/glspec21.20061201.pdf

[13] Roosendaal, T., and S. Selleri. 2004. *The Official Blender 2.3 Guide: Free 3D Creation Suite for Modeling, Animation, and Rendering.* No Starch Press, San Francisco, CA.
[14] Borho, S., J. Pittner, and G. Wainer. 2004. Defining and Visualizing Models of Urban Traffic. In *Proceedings of the SCS 1st Mediterranean Multiconference on Modeling and Simulation*, Genoa, Italy.
[15] Wolfram, S. 2002. *A New Kind of Science*. Wolfram Media Inc., Champaign.
[16] Gardner, M. 1970. Mathematical Games: The Fantastic Combinations of John Conway's New Solitaire Game 'Life'. *Scientific American* 223(4), 120–123.
[17] Modelica Association. *Modelica® A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification, Version 3.0*, http://www.modelica.org/documents (accessed September 2007).
[18] Dymola AB, Lund, Sweden. *Dymola Release Notes, Version 6.1*, http://www.dynasim.se/Dymola61releasenotes.htm (accessed October 26, 2007).
[19] Baldwin, P., S. Kohli, E. A. Lee, X. J. Liu, and Y. Zhao. 2005. VisualSense: Visual Modeling for Wireless and Sensor Network Systems. *Technical Memorandum UCB/ERL M05/25*, University of California, Berkeley, CA.
[20] Bhattacharyya, S.S., E. Cheong, J. Davis II, M. Goel, C. Hylands, B. Kienhuis et al. 2007. Volume 1: Introduction to Ptolemy II. In Brooks, C., E.A. Lee, X. Liu, S., Neuendorffer, Y. Zhao, and H Zheng (eds), *Ptolemy II – Heterogeneous Concurrent Modeling and Design in Java*, Version 6.0. Technical Report UCB/EECS-2007-7, University of California, Berkeley, CA, USA, 2007.
[21] Fishwick, P.A., J. Lee, M. Park, and H. Shim. 2003. RUBE: A Customized 2D and 3D Modeling Framework for Simulation. In *Proceedings of the 35th Winter Simulation Conference*, New Orleans, LA, USA, 755–762.
[22] SourceForge. *The Sodipodi Project, Version 0.34*, http://sourceforge.net/projects/sodipodi/(accessed October 2007).
[23] Chou, H.H., W. Huang, and J.A. Reggia. 2002. The Trend Cellular Automata Programming Environment. *SIMULATION: Transactions of the Society for Modeling and Simulation International* 78(2), 59–75.
[24] Cai, Y., I. Snel, B. Cheng, B. Suman Bharathi, C. Klein, J. Klein-Seetharaman. 2005. BioSim – A Biomedical Character-Based Problem Solving Environment. *Future Generation Computer Systems* 21(7), 1145–1156.
[25] Filippi, J.B., M. Delhom, and F. Bernardi. 2002. The JDEVS Modeling and Simulation Environment. In Rizzoli, A.E. and, A.J. Jakeman (eds), *Integrated Assessment and Decision Support, Proceedings of the 1st Biennial Meeting of the International Environmental Modelling and Software Society*, Vol. 3, Manno, Switzerland, 283–288
[26] Praehofer, H., and Pree, D. 1993. Visual Modeling of DEVS-based Multiformalism Systems Based on Higraphs. In *WSC'93: Proceedings of the 25th conference on Winter simulation*, Los Angeles, CA, USA, 595–603.
[27] Christen, G., A. Dobniewski, and G. Wainer. 2004. Modeling State-Based DEVS Models in CD++. In *Proceedings of MGA, Advanced Simulation Technologies Conference 2004 (ASTC'04)*, Arlington, VA, USA.
[28] Wainer, G., and W.H. Chen. 2003. A Framework for Remote Execution and Visualization of Cell-DEVS Models. *SIMULATION: Transactions of the Society for Modeling and Simulation International* 79(11), 626–647.
[29] Wainer, G. 2007. Developing a Software Toolkit for Urban Traffic Modeling. *Software: Practice and Experience* 37(13), 1377–1404.
[30] Khan, A., and G. Wainer. 2005. A Visualization Engine Based on Maya for DEVS Models. In *Proceedings of SISO Fall Interoperability Workshop*, San Diego, CA, USA.
[31] Khan, A., G. Wainer, W. Venhola, and M. Jemtrud. 2005. On the Use of CD++/Maya for Visualization of Discrete-event Models. In *Proceedings of the 17th IMACS World Congress, Scientific Computation, Applied Mathematics and Simulation*, Paris, France.
[32] Poliakov, E., G. Wainer, J. Hayes, and M. Jemtrud. 2007. A Busy Day at the SAT Building. In *Proceedings of AIS 2007, Artificial Intelligence, Simulation and Planning*, Buenos Aires, Argentina.
[33] Lam, K., and G. Wainer. 2003. Modeling of maze-solving problems using Cell-DEVS. In *Proceedings of the 2003 SCS Summer Computer Simulation Conference*. Montreal. QC. Canada.
[34] Djafarzadeh, R., G. Wainer, and T. Mussivand. 2005. DEVS Modeling and Simulation of the Cellular Metabolism by Mitochondria. In *Proceedings of the 2005 DEVS Integrative M&S Symposium*, Spring Simulation Conference, San Diego, CA, USA, 55–62.
[35] Liu, S., Y. Liang, B. Xu, L. Zhang, B. Spencer, and M. Brooks. 2007. On Demand Network and Application Provisioning Through Web Services. In Proceedings of 2007 IEEE International Conference on Web Services (ICWS).
[36] Venhola, W., and G. Wainer. 2006. DEVSView: A Tool for Visualizing CD++ Simulation Models. In *Proceedings of the 2006 DEVS Integrative M&S Symposium*, Spring Simulation Conference, Huntsville, AL, USA.
[37] Kilgard, M.J. 2000. *The OpenGL Utility Toolkit (GLUT) Programming Interface, API Version 3*, http://www.opengl.org/documentation/specs/glut/
[38] Jackins, C.L., and S.L. Tanimoto. 1980. Oct-trees and Their Use in Representing Three-Dimensional Objects. *Computer Graphics & Image Processing (CGIP)* 14(3), 249–270.
[39] TDT 3D. 2007. 3D Software Comparison Tables. http://www.tdt3d.be/articles_viewer.php?art_id=99 (accessed November 2007).

Gabriel Wainer *(Senior Member, SCS) received MSc (1993) and PhD degrees (1998, with highest honors) from 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. At Carleton, he is a member of the Real-Time and Distributed systems lab, a chair of the Research Centre in Technology Innovation and the head of the Advanced Real-Time Simulation lab within Carleton University Centre for advanced Simulation and Visualization (V-Sim). He has held positions at the Computer Science Department of the University of Buenos Aires, and visiting positions in numerous places (University of Arizona, Ecole Polytechnique de Marseille, CNRS, University of Nice and INRIA Sophia-Antipolis). He is author of three books and over 170 research articles. He has collaborated in the organizing of over 70 conferences in the area. He was Principal Investigator of various research projects (funded by National Science and Engineering Research Council of Canada, Precarn, Usenix, the Canadian Foundation of Innovation, CANARIE, and private companies including HP, IBM, Intel and the MDA Corporation). His current research interests are related with modelling methodologies and tools, parallel/distributed simulation and real-time systems.*

Qi Liu *received his BEng from the Huazhong University of Science and Technology, China in 1993, and the MASc degree from Carleton University in 2006. He was a recipient of a Senate Medal for Outstanding Academic Achievement for his research work. He is currently a PhD candidate in the Department of Systems and Computer Engineering at Carleton University, Ottawa, ON, Canada. He is the Chair and founding member of the Ottawa Student Chapter of SCS. His research interest is centered on high-performance computing in simulation and advanced parallel/distributed simulation algorithms.*