# Recent Advances on DEVS Modeling and Simulation Methodologies

*Gabriel Wainer*
Dept. of Systems and Computer Engineering - V-Sim
Carleton University
1125 Colonel By Drive. Ottawa, ON. K1S 5B6. Canada.
gwainer@sce.carleton.ca

with *Rodrigo Castro* and *Ernesto Kofman* (Systems Dynamic Laboratory. UNR. Rosario, Argentina).

**ABSTRACT**: DEVS is an increasingly accepted framework for understanding and supporting modeling and simulation. DEVS is a sound formal framework based on generic dynamic systems, including well defined coupling of components, hierarchical, modular construction, support for discrete event approximation of continuous systems and support for repository reuse. DEVS theory provides a rigorous method for representing models, and presents an abstract way of thinking about the world with independence of the simulation mechanisms, underlying hardware and middleware. In this presentation we will introduce and summarize a variety of advances carried out by our team in this field. The article summarizes the contents of our discussion, and it is based on previous articles (whose full version can be found in the list of references).

## 1 Introduction

In recent years, we have witnessed tremendous advances in model building and simulation execution thanks to the improvements in software and hardware technology. For many existing systems, analytical solutions are not feasible, while direct experimentation can be dangerous or impractical. Discrete-Event simulation methodologies were created to model systems that exist in finite set of discrete states over continuous periods of time (i.e. queuing systems, computer networks, manufacturing facilities, etc.).

DEVS (Discrete Event systems Specifications) is a technique defined in [1] that allows the modular description of discrete-event systems that can be integrated using a hierarchical approach. DEVS has been successfully used in a wide variety of applications and environments, providing ease for reuse of simulation models. Another advantage of using DEVS is that different existing techniques (Bond Graphs, Cellular Automata, State Charts, Partial Differential Equations, Petri Nets, Queuing networks, Timed Automata, etc.) have been mapped to DEVS. This permits sharing information at the level of the model, and different submodels can be specified using different techniques, while keeping independence at the level of the simulation engine. Existing DEVS tools have showed their ability to execute such wide variety of models with high performance in standalone or distributed environments.

## 2 The DEVS Formalism

A real system modeled with DEVS is described as a composite of submodels, each of them being behavioral (atomic) or structural (coupled). A DEVS atomic model can be informally described as in Figure 1.

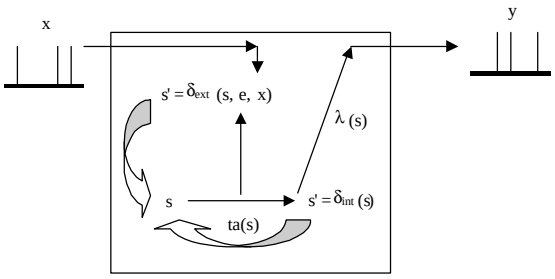



***Figure 1. Informal description of an atomic model.***

Each atomic model can be seen as having an interface consisting of *input* (x) and *output* (y) *ports* to communicate with other models. Every *state* (**s**) in the model is associated with a *time advance* (**ta**) function, which determines the duration of the state. Once the time assigned to the state is consumed, an internal transition is triggered. At that moment, the model execution results are spread through the model's output ports by activating an *output function* (l). Then, an *internal transition function* ($d_{int}$) is fired, producing a state change. Input external events are received in the input ports, and they activate $d_{ext}$.

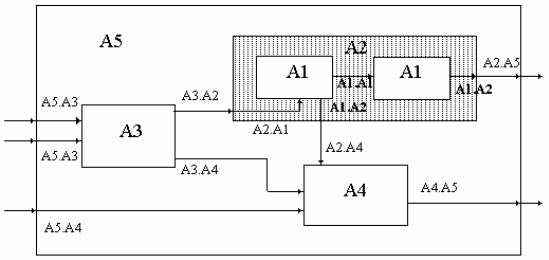



***Figure 2. Informal description of a coupled model.***

A DEVS coupled model is composed by several atomic or coupled submodels. Coupled models are defined as a set of components (atomic or coupled), which are interconnected through the model's interfaces. The model's coupling defines how to convert the outputs of a model into inputs for the others, and to inputs/outputs to the exterior of the model, as seen in Figure 2.

Many applications need components containing with variables and time. These can be modeled as Ordinary Differential Equation with initial conditions, which have traditionally been simulated by discretizing the time domain, and solving the ODE over each discrete time interval. Recently, Quantized DEVS [2] permitted to solve this problem using a different approach, depicted

in Figure 3. Using Q-DEVS, a curve is represented by the crossing of an equal spaced set of boundaries, separated by a *quantum* size. Only when a crossing occurs, an event is generated, reducing substantially the frequency of message updates.

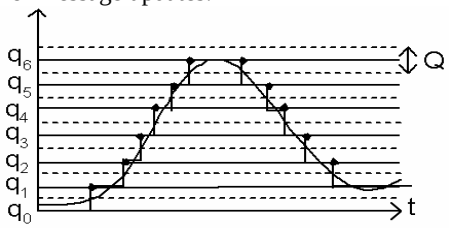
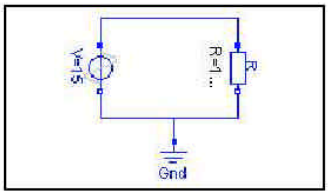



***Figure 3. Quantized DEVS (Q-DEVS)***

A different approach to model continues systems, is the one used by Modelica [3], an object-oriented language for modeling physical systems. Modelica was designed to support library development and model exchange. Models in Modelica are mathematically described by differential, algebraic and discrete equations. Modelica has many libraries of standard components (ODEs, block diagrams, electrical and mechanical).




```
Model circuit
Modelica.Electrical.Analog.Sources.SineVoltage
           V(V=15,freqHz=60);
Modelica.Electrical.Analog.Basic.Resistor
           R1(R=10);
Modelica.Electrical.Analog.Basic.Ground Gnd;
 equation
   connect(V.p, R1.p);
   connect(R1.n, V.n);
   connect(R1.n, Gnd.p);
 end circuit1;
```

***Figure 4. Electrical model in Modelica*.**

The example presented in Figure 4 shows a model of an electrical circuit (V generates a sine voltage). In [4] we showed that Modelica models can be translated into DEVS, which allows seamless integration of continuous and discrete-event components.

Another method to define continuous system considers the discretization of the space where the model is defined, using a grid describing the physical properties of the space. Cell-DEVS [5] is one of these methods, based on DEVS. A Cell-DEVS model is seen as a lattice of cells holding state variables and a computing apparatus, which is in charge of update the cell state according to a local rule. This is done using the present cell state and those of a finite set of nearby cells (called its neighborhood). Cell-DEVS uses a discrete-event approach: each cell is defined as a DEVS atomic model, and it can be integrated into a coupled model representing the cell space.

Each cell uses N inputs to compute its next state. These inputs, which are received through the model's interface, activate a local computing function (t). A delay (**d**) can be associated with each cell. The state (**s**) changes can be transmitted to other models, but only after this delay. Once the cell behavior is defined, a coupled Cell-DEVS can be created by putting together a number of cells interconnected by a neighborhood relationship. A Cell-DEVS coupled model is informally presented in Figure 5.

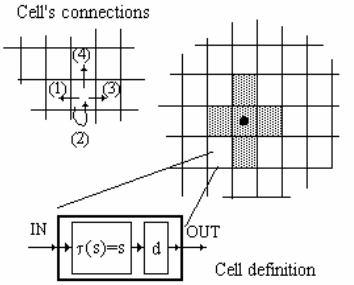



***Figure 5. Description of a Cell-DEVS coupled model.***

A coupled Cell-DEVS is composed of an array of atomic cells of a given size and dimensions. Each cell is connected to its neighborhood through DEVS I/O ports.

Several tools have implemented DEVS theory, including ADEVS, CD++, DEVS/HLA, DEVSJAVA, DEVSim++, GALATEA, PyDEVS and SimBeams (a non-comprehensive list can be found at [6]).

The generality of DEVS made it widely used to describe many classes of systems, given it permits modeling systems with a set of infinite possible states, and where the new state after an event arrival may depend on the elapsed time in the previous state. Models showing this behavior cannot be represented by any other discrete formalism. The following is a non-exhaustive list of such applications, which shows the relevance of the approach:

- In [7], an environment for the analysis of multi-agent robots was presented, using DEVS representation of a mobile robot is combined with stochastic learning.
- Models of large ecosystems, including watersheds [8] and fire spreading [9] that enabled the understanding and prediction of environmental phenomena.
- Prototyping and testing environment for embedded system design [10]. This allows verifying embedded systems' in the form of formal and simulatable applications prior to the deployment stage.
- Urban traffic analysis [11], providing support for evaluating signal control strategies, alleviating congestion in peak hours, and understanding conflicts.
- A decision support system for an intermodal container terminal [12]. In this case there is a need for studying spatial allocation of containers, routing goods, scheduling of operations and resource allocation.
- Analysis of the behavior of a distributed Intrusion Detection System to identify suspicious computer network traffic in real-time [13].
- Complex systems in aerospace manufacturing and military applications [14]. These applications showed how to integrate DEVS and the HLA [15], allowing distributed simulation, interoperability, and reuse.
- Supply chain applications [16], to help to determine strategies to provide the most profitable operating environment considering site location, replenishment policies, transportation policies, and inventory levels.

- Signal filters for SACHEM, a real-time diagnosis system [17], developed to supervise the blast furnaces of a large steel producer using knowledge acquired from experts. DEVS models are used to filter input continuous signals and to convert them into discrete events fed into an expert system.
- DHMIF (DEVS Hardware Model Interchange Format), a formal means to integrate the representation of hardware models developed with heterogeneous languages [18]. It can represent digital circuits modeled in different hardware description languages.

## 3 STDEVS

Stochastic models play a fundamental role in discrete event system theory. Any system involving uncertainties, unpredictable human actions, machine failures, system overloading, etc. requires a non–deterministic treatment. Examples of stochastic discrete event formalisms are Markov Chains, Queuing Networks and Stochastic Petri Nets, which permit simulating stochastic models in several applications. However, although some early work have studied the relationship between stochastic and pseudo-random processes and DEVS [19] and there is an extension for stochastic DEVS limited to finite state sets [20]; there is not a general theory nor a general formalism related to non determisitic DEVS models.

We are currently working on formally extending DEVS for modeling of stochastic systems (STDEVS). Taking into account that DEVS can work with sets of infinite possible states, we make use of Probability Space Theory and combine it with DEVS system theoretic definition to define the new formalism. Atomic STDEVS model are:

$$M_{ST} = (X, Y, S, G_{\text{int}}, G_{\text{ext}}, P_{\text{int}}, P_{\text{ext}}, l, ta)$$

*X, Y, S, l*, and *ta* have the same definition that those of classic DEVS. However, STDEVS has *probability spaces* that model the stochastic processes that calculate the next state. In order to construct the internal transition stochastic dynamic description, we start defining the *internal set-collecting function* $G_{\text{int}} : S \to 2^S$, a function assigning a collection of sets $G_{\text{int}}(s) \subseteq 2^S$ to every model state $s \in S$. $G_{int}(s)$ is the collection of all the *important* subsets of *S* for which, when we are in state *s*, we know the probability that the system goes into them. This is, there is a probability function $P_{\text{int}} : S \times 2^S \to [0,1]$ so that $P_{int}(s,G)$ gives the probability of going from state *s* to any subset of future states *G* in $G_{int}(s)$. Then, considering the event space $F_{\text{int}}(s) = M(G_{\text{int}}(s))$ (i.e., the smallest sigma algebra to which all the sets in $G_{int}$ belong), the triplet $(S, F_{\text{int}}(s), P_{\text{int}}(s,\cdot))$ obtained is a well defined probability space. Similarly, the probability space for the external transition stochastic description, results in the triplet $(S, F_{\text{ext}}(s,e,x), P_{\text{ext}}(s,e,x,\cdot))$. Here, $G_{\text{ext}} : S \times \Re_0^+ \times X \to 2^S$, $F_{\text{ext}}(s,e,x) = M(G_{\text{ext}}(s,e,x))$, $G_{\text{ext}}(s,e,x) \subseteq 2^S$, and $P_{\text{ext}} : S \times \Re_0^+ \times X \times 2^S \to [0,1]$.

We have proven that STDEVS is a generalization of DEVS [21], i.e., DEVS is a particular case of STDEVS. This fact allows combining DEVS and STDEVS models in coupled models of combined stochastic and deterministic systems. STDEVS provides an unified framework with continuous systems, by interacting with a novel family of numerical integration algorithms which allows the simulation of continuous systems in term of DEVS [22], exhibiting important advantages over discrete time approximations in the simulation of hybrid systems. The strategy pursued behind the STDEVS definition is to converge towards Control-Oriented Hybrid-Systems Modeling and Real-Time Simulation. The objective of guaranteeing the Quality of Service (QoS) of complex, resource-limited computing systems, has been targeted many times motivating diverse control strategies for admission control, load balancing and resource sharing problems. These control strategies are aimed at avoiding system congestion and saturation in the presence of different, unpredictable workload scenarios or abnormal system conditions.

The most sophisticated techniques are based on Control Theory, trying to maximize Objective Functions defined for the system's performance, which in turn define the design of their supporting control strategies. These functions are typically targeted to boost the quality metrics that will shape the QoS as seen from a user's standpoint (i.e., throughput, response time, delays) while keeping cost-related metrics low (hardware utilization, queue lengths, storage space, power consumption). In the case of the Utility Computing paradigm, Objective Functions are explicitly expressed in terms of contractual obligations and revenue objectives associated with the service offered.

The strategy pursued behind the STDEVS definition is to converge towards Control-Oriented Hybrid-Systems Modeling and Real-Time Simulation. QoS requirements usually need to be mapped into class-partitioned or service-differentiated loads that might respond to stochastic variation patterns along time, and might require different system resource sharing. System's hardware/software utilization depends on several low level particularities of the architectures, serving both user's demands and local tasks. Additionally, when operation limits are reached (i.e., task timeouts, maximum connections reached, system running low on batteries, etc.), a non-linear end to end system behavior is obtained, due to abrupt state changes.

Within this scenario, the use of modeling and simulation disciplines to design, test, validate and verify the various algorithms implementing the varied control strategies is crucial. Only with the support of powerful simulators and strong descriptions, the design of QoS-Controlled computing systems can be taken to the level of precision, standardization and productivity industry requires. Furthermore, Real-Time simulation enable to implement the control systems designed in the modeling and simulation stages right away into the target system under study and evaluate performance enhancements.

The proposed methodology is based on the use of non-linear discrete-time models to describe the evolution of the computing systems under study. These models and

their simulation environments shall be described with STDEVS which enables the representation of standard control theory techniques. Finally, models must expose open and robust interfaces, so inter-domain areas in computing systems modeling can be faced under a common and hierarchical framework. STDEVS is envisioned as a common theoretical framework and practical simulation environment to bring together the control design discipline with the discrete stochastic representation of networks and computing system's shared resources phenomena.

## 4 The CD++ toolkit

CD++ [23] is a modeling and simulation tool based on implementing DEVS theory. The tool provides a specification language that allows describing model coupling; additionally, atomic models can be developed using C++. CD++ was built as a hierarchy of classes in C++, each corresponding to a simulation entity using the basic concepts defined in [1]. The *Atomic* class implements the behavior of an atomic component, whereas the *Coupled* class implements the mechanisms of a coupled model.

CD++ makes use of the independence between modeling and simulation provided by DEVS, and different simulation engines have been defined for the platform: a stand-alone version, a Real-Time simulator [24], and a Parallel simulator [25]. At present, a CD++ wrapper has been built, enabling CD++ simulations to run as HLA federates [26], and the simulation engine is being extended to support distributed simulation of atomic models using the HLA, and a similar approach was created based on Web-Service implementation [27]. Another current effort is focused in providing support for development of real-time simulation in embedded platforms.

Model definition in C++ allows the user great flexibility to define behavior. Nevertheless, a non-experienced user can have difficulties in defining models using this approach. The provision of graphical notations is a powerful tool to define models. Graph-based notations have the advantage of allowing the user to think about the problem in a more abstract way. Therefore, we have used an extended graphical notation to define atomic models behavior. Each graph defines the state changes according to internal and external transition functions, and each is translated into an analytical definition.

The state machine specification presented following shows two views: the left side of the GUI contains a sorted tree diagram, and the right contains a visual representation of the model. External transitions are displayed as dashed lines, with internal transitions as solid lines. The input and output ports are visible in the tree diagram.

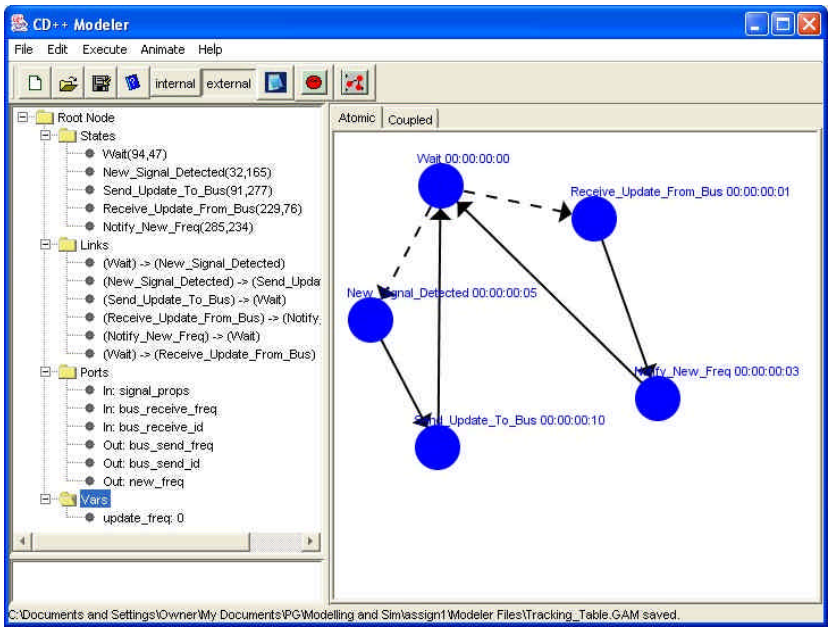



***Figure 6. Specification of a state-based atomic model.***

Once an atomic model is defined, it can be combined with others into a multicomponent model using a specification language specially defined with this purpose. It describes the internal and external coupling scheme. If the name of the model is not included, the default will be the coupled model currently being defined.

CD++ also includes an interpreter for Cell-DEVS models. The language is based on the formal specifications of Cell-DEVS. The model specification includes the definition of the size and dimension of the cell space, the shape of the neighborhood and borders, as presented. The cell's local computing function is defined using a set of rules.

CD++ was built as a class hierarchy in C++, where each class corresponds to a simulation entity. There are two basic abstract classes: *Model* and *Processor*. The former is used to represent the behavior of the atomic and coupled models, while the latter implements the simulation mechanisms. *Simulators* manage the atomic models. *Coordinators* manage coupled models. The

*Root Coordinator* manages global aspects (starting/stopping the simulation, communication with the environment). This reflects the clear distinction between the model and its simulator. CD++ was redesigned to provide parallel execution of DEVS and Cell-DEVS [28]. The parallel version of CD++ was built on top of Warped [29], a simulation kernel that provides an implementation of Time Warp. A flat simulation mechanism reduces the message passing overhead by simplifying the underlying simulator structure, while keeping the model definition and preserving the separation between model and simulator [30]. The Flat Coordinator eliminates the coordinators in the hierarchy by making direct messaging communications between the Flat Coordinator and the simulators [31], as shown following.

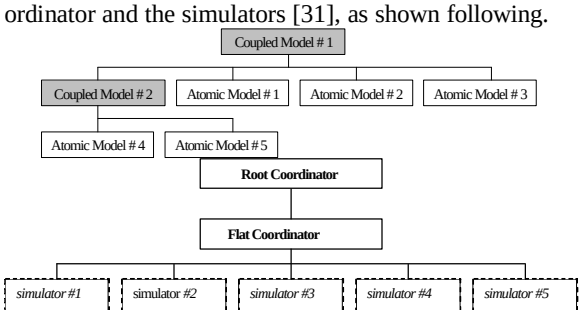



***Figure 7. Flat Coordinator (a) Example of a model hierarchy, (b) Associated processor hierarchy***

We took advantage of the separation of concerns by focusing on the *processors*' class hierarchy only (all classes inheriting from *model* remain unchanged from those defined in earlier versions of the tool, allowing direct reuse of existing models). Two new classes are introduced [25]: *Flat Coordinator* (*FC*) and *Node Coordinator* (*NC*). Additionally, we modified the *Simulator* and *Root Coordinator* classes. The algorithms we defined are based on those in [32]. The Root Coordinator only handles I/O operations, and starts/stops the simulation. The NC is in charge of synchronization and time management for the LP. The FC is responsible for receiving, translating, and sending messages between its descendants, using a flat data structure with coupling information for every component.

The Embedded version of the simulator, called *eCD++* [33] provides a Flat Coordinator, an interpreter for the state notation presented in Figure 6, and a real-time engine. eCD++ allows the models to be simulated in real-time by tying the simulation time to the real-time clock, and permitting interaction between the simulator and the surrounding environment. The inputs can be received by ports connected to real input devices such as sensors, timers, thermometers, or data collected from human interaction. The outputs can be sent through output ports connected to devices such as motors, transducers, gears, valves, or any other component. For the *real-time* simulation, the coordinator waits until the physical time reaches the next event time to initiate a new cycle.

Timeliness along a simulation is a substantial property in the real time approach. Thus, it is important to check timing constraints along the simulation. Particularly, the time at which an event has been completely processed is a meaningful measure of success. In a typical real-time situation, the model has to react to an external event and generate the output within a given time in order to solve a problem. The eCD++ real-time extension allows the modeler to indicate the deadlines for external events. The simulator can check whether the physical time meets the associated deadline analyzing successful and unsuccessful deadlines for further study of the process.

## 5 M/CD++

*MCD++* is an extension to CD++ that allows simulation of a particular class of dynamic systems, those related to the electrical domain. The electrical circuits can be modeled in Modelica, and then simulated using a discrete event simulator. This presents a completely different approach on dynamic systems simulation techniques, compared to the existing implementations. *MCD++*, is based on Q-DEVS theory, and provides the extensions needed to accomplish simulation based on discrete events [4].

The user must provide a source code file as input to the Modelica compiler. The electrical library we defined for Modelica/CD++ starts by converting Modelica models into Bond Graphs, which represent continuous systems as a set of elements that can interact with each other by exchanging energy and information, and this exchange determines the dynamics of the system. The compiler constructs the corresponding model of the circuit in a Bond Graph representation. In this Bond Graph, we check for algebraic loops and singularities. Then, we generate an optimized Bond Graph corresponding to the electrical circuit, which is used to generate a coupled DEVS model specification in CD++.

Figure ***8*** presents the simulation results of different Modelica/CD++ test cases for the circuit introduced in Figure 4 using sinusoidal and pulse voltage source. The results obtained are consistent with the real behavior of the electrical circuits. The figure shows the voltage and the current on resistor R. The resistor is a passive element in the oscillating electrical circuit so the current and the voltage on the resistor are in phase; the amplitude of electrical current is I=V/R1.

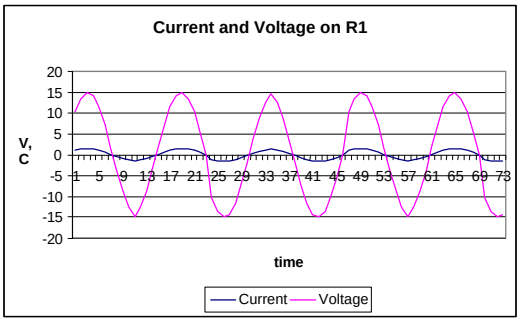



***Figure 8. Electrical model with sine signal.***

## 6 WEB SERVICE-ENABLED CD++

As mentioned in Section 4, we have defined an extension to CD++ using Web-services [27]. In order to do so, the toolkit was *wrapped* by a web service (exposing

its functionality to remote users/services), and the simulation web service was extended to execute distributed models. We used the main web service standards such as XML, SOAP, Web Service Description Language (WSDL) for storing and parsing the configuration files used by the service, describing and exposing the service functionality, and messaging among the simulation services themselves as well as with the users, respectively. The model is decomposed into different partitions, each of which is assigned to a machine for execution with SOAP being used for messaging among the machines.

The web service was designed to provide a robust environment for running different simulation sessions concurrently and independently. The service was split into two independent parts: the *web service components* are used to handle the web service activities, and the *simulation components* are used to interact with CD++.

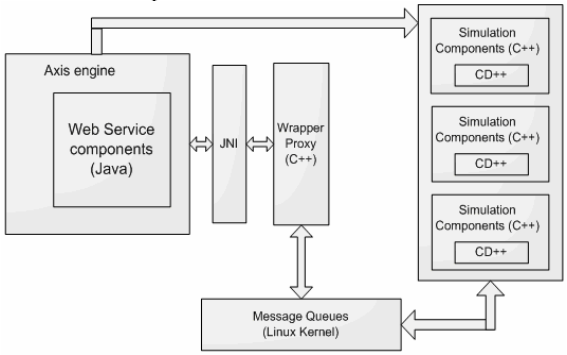



***Figure 9. Simulation service***

The *web service components* are deployed in an Axis server, which in turn runs within an Apache Tomcat server. Axis loads all the deployed services, which include the *JavaWrapper*, the server-side stubs, and the client-side stubs. At this point the simulation service is ready to receive client requests.

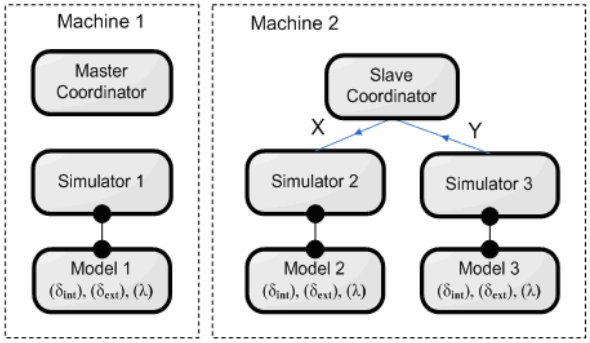



***Figure 10. Master* and *Slave* coordinators**

Model partitioning information is provided through a *grid configuration file* (an XML file containing the addresses of the machines executing the model and the parts of the model running on each machine). We use one coordinator in each machine for message routing among the local *processors* [28]. The idea depends on using two kinds of coordinators:

- *Master:* responsible for synchronizing the model execution, interacting with upper level coordinators and message routing among local and remote components.

- *Slave:* responsible for message routing among the local model components dispensing and remotely if the master coordinator is residing on a different machine.

## 7 DEVSVIEW

Originally, CD++ only provided results on text files, making it difficult to study execution results of the model. Visualization tools are crucial in helping to understand better the behavior of the system of interest, thus, different visualization facilities were incorporated [34]. DEVSView is able to run on OpenGL-based environments.

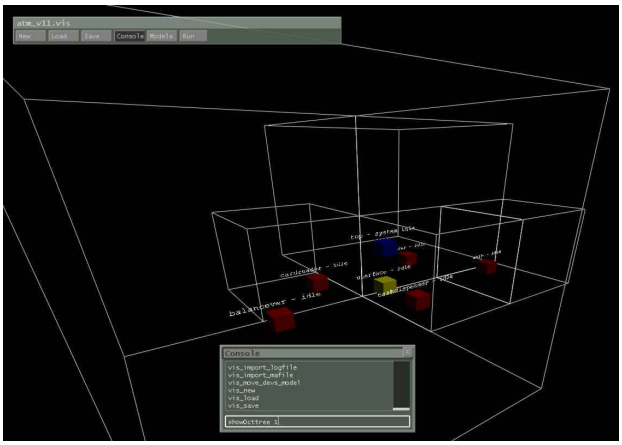


***Figure 11. DEVSView outputs.***

DEVSView provides basic services that enable simple visualizations. The visual models are stored in an octary space partitioning tree. This data structure recursively divides the scene extents into eight regions, which enables efficient algorithms for rendering scenes, object selection, and other frequently used scene operations (Figure 11).

A different version of the software was constructed using Maya [35], a powerful application for 3D modeling and animation. Maya interface is fully customizable and it allows users to extend their functionality within Maya by providing access to the Maya Embedded Language (MEL). Using MEL, programmers can tailor the user interface to their needs and to add in-house tools. Maya's modeling and animation tools were used to create three-dimensional environments for Cell-DEVS and DEVS models [36]. To do that, the user must use Maya facilities to create visual scene files, while an application written in MEL permits to create a user interface that allows CD++ log files to interact with Maya, and to visualize the corresponding model in a 3D visual environment. This instantiates a MEL script specific to a particular model, and animates the 3D world in accordance with the CD++ log file, as seen in Figure 12.

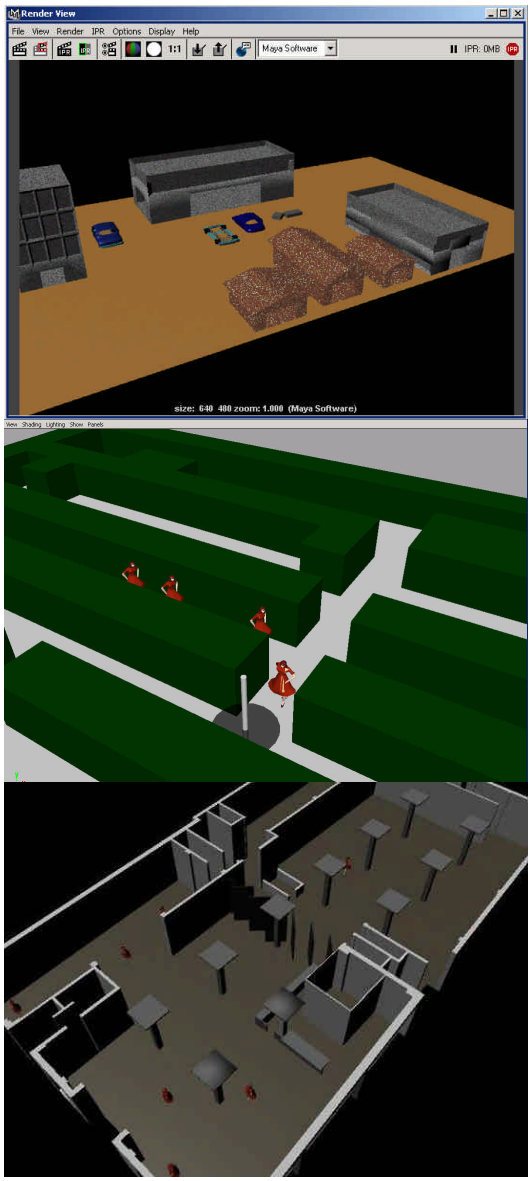



***Figure 12. Visualization in CD++/Maya.***

## 8 Application examples

We developed a large number of applications in a variety of fields, available for public use. Different areas of application include generic artificial systems, biology, defense, emergency planning, construction, environmental sciences, physics, chemistry, urban traffic, and others. Detailed analysis on each of these fields will be found in [37]. In this section we introduce a few examples in different fields to show some recent results.

### *8.1 Models in Physics and Chemistry*

We have developed a number of models with application in Physics and Chemistry, including particle collisions, finite element approximation of heat, flow injection analysis, binary solidification, crystal growth, plastic deformation, spring behavior, and others. In this section we show two different examples [38]. The first one, a model of Diffusion Limited Aggregation (DLA) occurs when diffusing particles stick to and progressively enlarge an initial seed represented. The seed typically grows in an irregular shape resembling frost on a window. Diffusion is a random motion with respect to the direction. There are two kinds of particles in a grid: fixed (seeds) and mobile.

We built a model of DLA, as a 2D Cell-DEVS. Initially, a certain percentage of the cells are occupied by mobile particles, and there are at least one or more seeds. The system evolves with the following rules.

- A particle can move in four directions (N/S/E/W)
- A particle becomes fixed an adjacent cell is fixed.
- An empty cell will be occupied if there is at least one mobile particle trying to move in, and there is no seed adjacent to the mobile particle.
- A mobile particle that cannot move will select a new direction at random.
- A mobile particle disappears if it strays too far from the center.

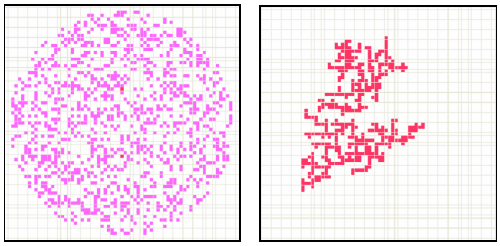


***Figure 13. Two seeds and 30% concentration.***

Several scenarios were executed with different seeds and concentrations. Figure 13 presents a case with concentration of 30% (grid: 71x71). The model presents fractal growth properties based on the initial configuration.

We also defined a model of driven diffusion, which describes the random motion of two types of particles in a system under the influence of an external field. The field may drive one species of particles along the field direction while the other species against that direction [37]. This kind of model can simulate the behavior for certain materials such as superionic conductors and solid electrolytes.

Initially, the space is occupied by the two randomly distributed particles A and B, and each particle has a randomly chosen direction to face (N/E/S/W). In the case of an external electrical field appearance (assuming the field points to the NE), the preferable moving direction of particle A is N or E while the preferable moving direction of particle B is S or W. The probability of A and B hops along that preferable direction is *a*, and the probability against the direction is (1-a).

Different tests were carried out, using different densities, space size, and initial states. Particles are initially distributed at random according to the given density value. The following figure, for instance, shows a case in which the density of the whole space is 40%. We can see that the distribution of the two particles exhibits striped, banded structure. Within each strip, there are two sub-strips each having approximately the same amount of particles. This indicates the non-

homogeneities of the distribution of two particles and thus results in reduced current in the system.

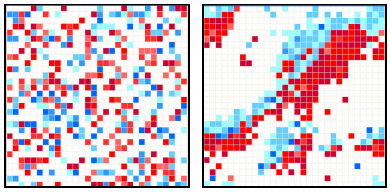


*Figure 14. Density of 40%*

### *8.2 Models in Biology*

We have created different models in a whole organelle scale. In [39], we created a precise model of the reactions in the mitochondrion, which creates energy for cellular activity by the process of aerobic respiration. We used CD++ to model and simulate biological pathways, thus providing a systematic method for creating models consisting of sets of lower-level interactions. The following figure shows a snapshot of some of the reactions in the Krebs Cycle (formation of Acetyl CoA).

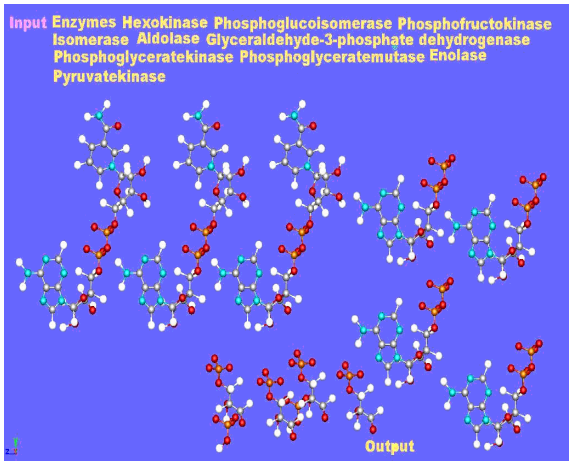



*Figure 15. Krebs cycle 3D visualization.*

We also created models of liver cells [40]. Our model is based on the one presented in [41], which defines the various reactions of the liver maintains. Our design demonstrates the process of substance transformations occurring with in the liver's lobule. The lobule is modeled like a hexagonal cylinder with 3 stages (zones), and several nodes are placed inside the lobule, where each node is connected to at least one other node. Each node is responsible for receiving a substance, and transforming it, and each node works interdependently of each other. We built a DEVS model based on these assumptions, which represents the chemical composition of blood entering the liver lobule. A substance would enter the portal vein (PV), and it is then fed to all the nodes that are in zone I. After the nodes of zone I are finished transforming the substance, their output is fed to the nodes of zone II and then zone III. After this, the output is supplied to the central vein (CV). We studied different reactions in the lobule, including gluconeogenesis, glycogen synthesis and degradation, phosphorylation and glycolosis. Each of these reactions was analyzed, based on the lobule model presented in Figure 16.

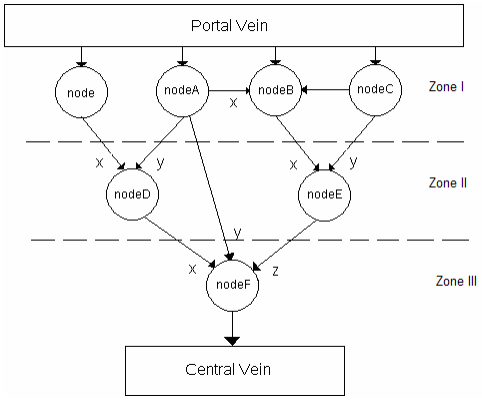



***Figure 16. Zones and Nodes*** [41]

### *8.3 Environmental Systems*

We defined a variety of models in environmental sciences, including pollution, watershed formation, ant foraging, vegetation growth, pesticide percolation, etc. [42]. In this section we show a Cell-DEVS fire model based on a well known model for fire propagation in forests due to Rothermel [43]. Three parameter groups determine the fire spread ratio: vegetation type, fuel properties, and environmental parameters. When Rothermel's rules are applied to a given fuel model and environmental parameters, it can determine the spread ratio (i.e. the distance and direction the fire moves in a minute). The first step is to use the fuel model, the speed and direction of wind, topography and dimensions of the cellular space to obtain the spread ratio in every direction. Instead of using a time-based approach, the model uses the delay function to compute fire spread. Figure 17 shows the implementation of this model using a hexagonal mesh.

```
dim : (20,20)  delay : inertial
neighbors : (-1,-1) (-1,0) (-1,1) (0,-1)  (0,0)
(0,1) (1,-1)  (1,0)  (1,1)

[FireBehavior]
rule: {[5]+(15.24/13.680)} {(15.24/13.680)
   * 60000} {[0]=0 and [5]!=? and [5]>0}
rule: {[6]+(15.24/5.10)} {(15.24 /5.106 )
   * 60000} {[0]=0 and [6]!=? and [6]>0}
rule: {[4]+( 15.24/2.950)} {(15.24/2.950)
   * 60000} {[0]=0 and [4]!=? and [4]>0}
...
```

***Figure 17. Rothermel's fire forest model.***

The rules defining the local computing function are devoted to detect the presence of fire in the eight neighboring cells. For instance, the first rule checks if the current cell is not burning ([0]= 0) and if the SW neighbor has started to burn ([5]>0). If this condition holds, the new value of the cell will be `[5]+(15.24/13.680)`, which is the time the fire will start in the cell. We use a delay of `(15.24/13.680) * 60000` ms after which the present cell state will spread

to the neighbors. The remaining rules represent a similar behavior for the neighbors.

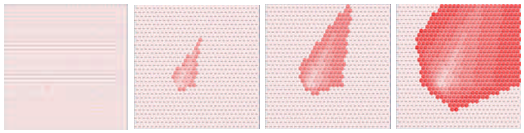


***Figure 18. Fire propagation results (2 h. period)***

As we can see, the burning time of a cell depends on the spread ratio in the direction of the burning cell. Changes in the propagation are related to the changes produced by the adjacency properties. This is a clear departure from the classical approach to cellular models where all active cells are updated at the same time.

### *8.4 Traffic Simulation*

In this section we present an introduction to the ATLAS M&S platform. In ATLAS, a modeler can easily describe a city section, including traffic signs, traffic lights, etc. [44]. ATLAS is formally defined as a set of constructions, mapped into DEVS and Cell-DEVS models [45].

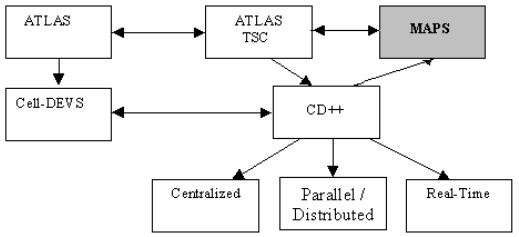
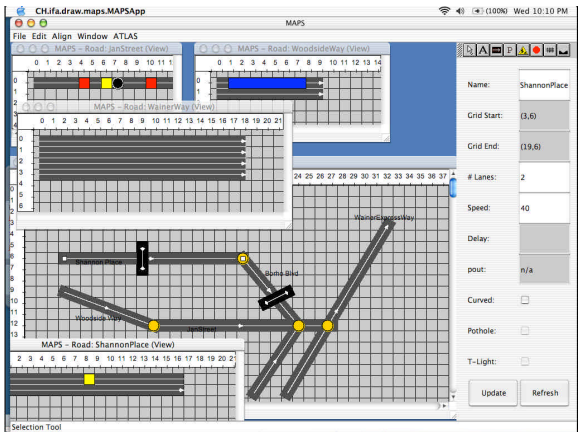



***Figure 19. Structure of the ATLAS software platform.***

The behavior for each of the constructions presented in this language was validated in terms of correctness. Then, a compiler was built following the specifications [46]. The compiler, called ATLAS TSC (Traffic Simulator Compiler), generates code by using a set of templates that can be redefined by the user.

A front-end program (MAPS) allows the user to draw a small city section complete with roads, intersections, and decorations, and then parse the drawing to create a valid ATLAS file. Likewise, the output can generate realistic 3D graphics.



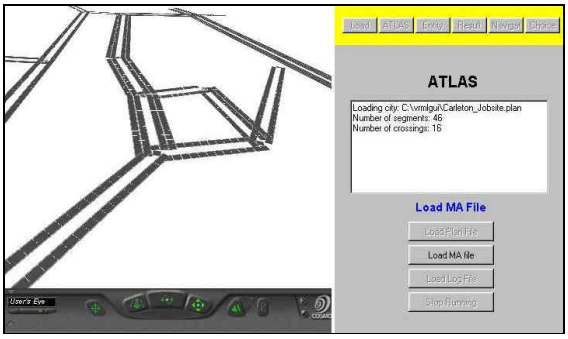



***Figure 20. ATLAS GUI.***

## 9 Conclusion

The use of DEVS can improve the security and cost in the development of the simulations. The main gains are in the testing and maintenance phases, the more expensive for these systems. The use of a formal approach made easy the development of the applications. DEVS was successfully applied in such a variety of applications due to the ease for model definition, improved composition and reuse, and as a result of hierarchical coupling. DEVS includes explicit specification of the model timing, and uses a discrete event approach for simulation. This provides precision and speedups in the execution time, as models advance triggered by instantaneous asynchronous events in contraposition with time stepped approaches. This allows enhanced model definition and high performance.

We are currently working on the completion of the theoretical framework of STDEVS, studying properties such as closure under coupling and legitimacy. We are also working on a formal proof of the fact that classic DEVS models whose transition functions depend on randomly generated parameters constitute particular cases of STDEVS. This property allows to develope STDEVS models replacing the use of probability spaces by simple random functions.

We are also working on a standardized version of DEVS models within a DEVS Study Group [6], whose goal is to enable DEVS environments to interact, and to include non-DEVS models in larger simulations. In this way, we will be able to shorten the gap existing between academic versions of DEVS, and industrial/government needs. Having standardized means of defining models will enable defining standard libraries that can be integrated in user-friendly modeling and simulation environments.

### References

[1] Zeigler B., Praehofer H. and Kim T.G., "Theory of Modeling and Simulation, second edition" *Academic Press, 2000.*

[2] B. P. Zeigler, "DEVS Theory of Quantization" *DARPA Contract N6133997K-0007: ECE Dept., the University of Arizona, Tucson,* AZ. 1998.

[3] Modelica, "Language Specification, version 2.1" *Http://www.modelica.org, 2004.*

[4] M. D'Abreu and G. Wainer, "M/CD++: modeling continuous systems using Modelica and DEVS" *Proceedings of MASCOTS 2005.* Atlanta, GA. 2005.

[5] G. Wainer and N. Giambiasi, "N-Dimensional Cell-DEVS" *In Discrete Events Systems: Theory and Applications, Kluwer.Vol, 12, no, 1.pp,* 135-157. 2002.
[6] G. Wainer, "DEVS Standardization Study Group" . http:///www.sce.carleton.ca/faculty/wainer/standard. 2007.
[7] A. El-Osery, J. Burge, M. Jamshidi, M. Fathi and M. R. Akbarzadeh, "V-LAB: A Virtual Laboratory for Autonomous Agents - SLA based Controllers," *IEEE Transactions on Systems, Man and Cybernetics,* vol. 32, pp. 791-803, 2002.
[8] B. Zeigler, Y. Moon, D. Kim and G. Ball, "The DEVS Environment for High-Performance Modeling and Simulation" *IEEE Computational Science and Eng.4 (3),* pp. 61 -71. 1997.
[9] M. Vasconcelos, A. Goncalves and F. Barros, "Dynamic Maps" *In Proceedings of AI, Simulation and Planning in High Autonomy Systems.*Tucson, Arizona. 2000.
[10] S. Schulz, J. W. Rozenblit, M. Mrva and K. Buchenriede, "Model-based codesign" *Computer,* vol. 31, pp. 60-67, 1998.
[11] S. Chi, J. Lee and Y. Kim, "Using the SES/MB framework to analyze traffic flow". *Transactions of the SCS.Vol, 14, no.4, pp,* 211-221. 1997.
[12] L. M. Gambardella, A. E. Rizzoli and M. Zaffalon, "Simulation and Planning of an Intermodal Container Terminal" *Simulation,* vol. 71, pp. 107-116, 1998.
[13] T. H. Cho and H. J. Kim, "DEVS Simulation of distributed intrusion detection systems". *Trans. Soc. Comput. Simul. Int.,* vol. 18, pp. 133-146, 2001.
[14] H. S. Sarjoughian Zeigler, "DEVS and HLA: Complimentary Paradigms for M&S?" *Transactions of the SCS, (17), 4, pp,* 187-197. 2000.
[15] IEEE standard for modeling and simulation (M&S, ) high level architecture (HLA) - Framework and Rules. pp. i -22.
[16] D. Kim, H. Cao and S. Buckley, "Modeling and simulation of supply chain management based on DEVS and CORBA framework". *Winter Simulation Conference,* Phoenix, AZ. 1999.
[17] C. Frydman, M. Le Goc, N. Giambiasi and L. Torres, "Knowledge-Based diagnosis in SACHEM using DEVS models" *Trans. of SCS,* vol. 18, pp. 148-159. 2001.
[18] J. K. Kim, Y. G. Kim and T. G. Kim, "DHMIF: DEVS-based hardware model interchange format" in *European Simulation Symposium,* Marseille, France. 2001.
[19] S. Aggarwal, "Ergodic machines--probabilistic and approximate homomorphic simplifications." 1975.
[20] C. Joslyn, "*The process theoretical approach to qualitative DEVS*" in *7th Conference on AI, Simulation, and Planning in High Autonomy Systems (AIS '96),* 1996, pp. 235-242.
[21] E. Kofman and R. Castro, "*STDEVS. A novel formalism for modeling and simulation of stochastic discrete event system*" in *AADECA 2006,* Buenos Aires, Argentina. 2006.
[22] E. Kofman, "Discrete Event Simulation of Hybrid Systems" *SIAM J. Sci. Comput.,* vol. 25, pp. 1771-1797, 2004.
[23] Wainer, "CD++: a toolkit to develop DEVS models" *Software Practice and Experience,* vol. 32, pp. 1261, 2002.
[24] E. Glinsky and G. Wainer, "Modeling and simulation of systems with hardware-in-the-loop" *in Proceedings of the Winter Simulation Conference.*Washington, DC. 2004.
[25] E. Glinsky and G. A. Wainer, "New parallel simulation techniques of DEVS and cell-DEVS in CD++." in *Annual Simulation Symposium,* Huntsville, AL. 2006, pp. 244-251.
[26] C. Zhang, "Integrating existing DEVS simulations with the HLA". M. A. Sc. Thesis. Carleton University. 2004.
[27] R. Madhoun, B. Feng and G. Wainer, "Web-service-based distributed CD++" in *Proc. of Artificial Intelligence, Simulation and Planning,* Buenos Aires, Argentina. 2007.
[28] A. Troccoli and G. Wainer, "Implementing Parallel Cell-DEVS" *Proceedings of 36th IEEE/SCS Annual Simulation Symposium.*Orlando, USA. 2003.
[29] D. Martin, T. McBrayer and P. Wilsey, "WARPED: Time Warp Simulation Kernel for Analysis and Application Development" *Proceedings of the 29th Hawaii International Conference on System Sciences, 1996.*
[30] K. H. Kim, Y. R. Seong, T. G. Kim and K. H. Park, "Distributed Simulation of Hierarchical DEVS Models: Hierarchical Scheduling Locally and Time Warp Globally" *Trans.of the SCS.Vol, 13 (3), pp,* 135-154. 1996.
[31] Kim, K., Kang W., Sagong,B.and Seo, H., "Efficient Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-Hierarchical One" *In Proc. of 33rd Annual Simulation Symposium.*Washington DC, *2000.*
[32] A. Chow and B. Zeigler, "Parallel DEVS: A parallel, hierarchical, modular modeling formalism" *Proc.of the Winter Simulation Conference.*Orlando, FL. 1994.
[33] H. Yu and G. Wainer, "E-CD++: developing embedded DEVS applications". Internal Report. Carleton University. Dept. of Systems and Computer Eng., 2007.
[34] W. Venhola and G. Wainer. "DEVSView: A tool for visualizing CD++ simulation models". In *Proceedings of SpringSim 2006 (DEVS Symposium).* Huntsville, AL. 2006.
[35] ALIAS Corp. "Maya 6 Features in Detail" *http://www.alias.com/eng/products-services/maya/file/maya6_features_in_detail.Pdf. 2004.*
[36] A. Khan, G. Wainer, W. Venhola and M. Jemtrud, "On the use of CD++/Maya for visualization of discrete-event models" *Proc.of IMACS World Congress.* Paris, France. 2005.
[37] G. Wainer, *Discrete-Event Modeling and Simulation: A Practitioner's Approach.* Taylor and Francis.To Appear. 2008.
[38] W. Ding, X. Wu, L. Checiu, C. Lin and G. Wainer, "Definition of cell-DEVS models for complex diffusion systems" *Proc. SCSC 2005..* Philadelphia, PA. 2005.
[39] R. Djafarzadeh, T. Mussivand and G. Wainer, "Modeling energy pathways in cells" *In Proceedings of the 2005 DEVS Integrative M&S Symposium,.San Diego, CA.U.S.A.2005.*
[40] G. Wainer, B. Al-aubidy, A. Dias, R. Bain, S. Jafer, M. Dumontier and J. Cheetham, "Advanced DEVS models with applications to biomedicine" *Proc. of Artificial Intelligence, Simulation and Planning.* Buenos Aires, Argentina, 2007.
[41] C. A. Hunt, G. Ropella, M. Roberts and L. Yan, "Biomimetic in silico devices" *Computational Methods in Systems Biology; Lecture Notes in Bioinformatics 3082,*2005,pp.35-43.
[42] Wainer, "Applying Cell-DEVS Methodology for Modeling the Environment" *Simulation,* vol. 82, pp. 635, 2006.
[43] R. Rothermel, "A mathematical model for predicting fire spread in wildland fuels" *Research Paper INT-115.Ogden, UT: U.S.Department of Agriculture, Forest Service,* Intermountain Forest and Range Experiment Station. 40 pp. 1972.
[44] G. Wainer. "ATLAS: A language to specify traffic models using Cell-DEVS". *Simul. Model. Pract. Theory,* vol. 14, pp. 313-337, APR. 2006.
[45] A. Davidson and G. Wainer, "Specifying control signals in traffic models". *In Proceedings of AI, Simulation and Planning in High Autonomous Systems,* Tucson, AZ. USA. 2000.
[46] M. Lo Tartaro, C. Torres and G. Wainer, "Defining Models of Urban Traffic using the TSC Tool" *Proc.of the Winter Simulation Conference.*Washington, DC. 2001.