Proceedings of the 2002 Winter Simulation Conference
E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

PERFORMANCE ANALYSIS OF REAL-TIME DEVS MODELS

Ezequiel Glinsky

Departamento de Computación
FCEN – Universidad de Buenos Aires
Planta Baja. Pabellón I.
1428 Buenos Aires, ARGENTINA

Gabriel Wainer

Dept. of Systems and Computer Engineering
Carleton University
4456 Mackenzie Building. 1125 Colonel By Drive
Ottawa, ON. K1S 5B6, CANADA

ABSTRACT

The CD++ toolkit was developed to implement the theoretical concepts specified by the DEVS formalism. CD++ has been recently enhanced to support real-time simulation, where events have to be processed in a timely manner. A synthetic benchmarking tool is used to test several models with different workloads, complexities, structures and sizes. Additionally, experiments were carried out under different scenarios to analyze the behavior in such conditions. Some problems and limitations were detected in particular cases. Lately, a flattened simulation technique has been introduced in the toolkit. The experiments presented in this work show that the flattened simulator is more efficient than the hierarchical one.

1 INTRODUCTION

The DEVS (Discrete EVents Systems specification) formalism provides a framework for the construction of hierarchical models in a modular fashion, allowing model reuse and reducing development and testing time (Zeigler et al. 2000). DEVS models can be executed using abstract simulation mechanisms independent of the model itself. Models are built using a set of basic models called **atomic**, which can be combined to form **coupled** ones. A DEVS atomic model is described as:

$$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, D >$$

Here, **X** is the input events set, **S** is the state set, and **Y** is the output events set. There are also several functions: δ_{int} manages internal transitions, δ_{ext} external transitions, $\boldsymbol{\lambda}$ the outputs, and **D** the elapsed time.

A DEVS coupled model is defined as:

$$CM = < I, X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\} >$$

Here, **X** is the set of input events, and **Y** is the set of output events. **D** is an index of components, and for each i $\in$ D, $\mathbf{M_i}$ is a basic DEVS model, where $M_i = < I_i, X_i, S_i, Y_i, \delta_{inti}, \delta_{exti}, ta_i >$. $\mathbf{I_i}$ is the set of influencees of model i. For each $j \in I_i$, $\mathbf{Z_{ij}}$ is the i to j translation function.

The **CD++** toolkit (Rodriguez and Wainer 1999) implements DEVS theory. A specification language allows the creation of coupled models, the initial configuration for the atomic models, and the creation of external events to be used during the simulation. Lately, the CD++ tool has been enhanced to support parallel and real-time simulation engines (Troccoli and Wainer 2001; Glinsky and Wainer 2002a).

1.1 Virtual-Time Simulation

DEVS provides the advantages of a discrete event approach in terms of execution performance. Discrete event models evolve in continuous time. Events are instantaneous and can occur asynchronously at unpredictable times. DEVS simulators can be seen as hierarchical schedulers of events that activate the corresponding submodels.

The CD++ toolkit originally presented a *virtual-time* approach. This strategy advances the time disregarding any real clock attached to the simulation mechanism and periods of inactivity are skipped.

The simulation in CD++ is carried out by *Processors* that drive the simulation by exchanging messages. Two types of *Processors* exist:

1. *Simulators:* drive the simulation of atomic models, and
2. *Coordinators:* drive the execution of coupled components and coordinate the activities of all their dependant children.

In order to execute a simulation using the virtual-time approach, CD++ keeps the current simulation time. When the simulation starts, this simulation time is initialized to zero. Then, the imminent event (i.e. the event with the earliest time of occurrence) is computed and the simulation time is advanced accordingly in order to process that event. Once it has been processed completely, the new imminent event is computed, the simulation time is advanced and the new event is processed. This cycle of advancing the simulation time and processing the imminent event is repeated. The model execution ends when the simulation time reaches the stop time indicated by the user, or else when there are no more pending events.

This strategy is useless if we intend to make our models to interact with the environment (such as in Hardware-in-the-loop or Man-in-the-loop simulations). In these cases, the time in the simulation framework does not evolve at the same speed as within its surroundings.

1.2 Real-Time Simulation

In order to overcome these problems, we have modified the CD++ toolkit to provide *real-time simulation*. A real-time system is defined as a system whose correctness depends not only on the logical results of computation, but also on the moment when the results are produced. If a system delivers the correct answer after a certain deadline, it could be regarded as an unsuccessful response (Stankovic 1988). Consequently, a real-time simulator must handle events in a timely fashion where time constraints can be stated and validated. These new features allow interaction between the simulator and the surrounding environment. Therefore, inputs can be received by ports connected to real input devices such as sensors, timers, thermometers or even data collected from human interaction. Similarly, outputs can be sent through output ports connected to devices such as motors, transducers, gears, valves or any other component.

The idea of the real-time simulator implementation is to tie time advance to the *wall-clock* (physical time). The simulator was modified to execute under real-time constraints that can be imposed easily by the user. We are able to analyze the results and the time in which these results are produced across the simulation. When this technique is used, models that interact with the environment can be executed.

The *root coordinator* manages the advance of time along the simulation. When the *virtual-time* approach is used, the messages are immediately generated by the *root coordinator* to initiate a new simulation cycle. Alternatively, when the *real-time* simulation is performed, the coordinator waits until the physical time reaches the next event time to initiate the new cycle.

Periods of inactivity are not skipped by the real-time simulator. The simulation process remains quiescent while these periods are being experienced. Instead of forcing a time advance up to the next programmed event and thus anticipating the execution of a programmed task, the simulator expects the scheduled time to be reached and only then starts the new simulation cycle. Hence, messages interchanged between processors are sent, ideally, at their actual scheduled time.

Typically, a model has to react to an external event within a given time to produce an output in order to solve a given problem, e.g., in case of having a sensor indicating dangerous overheat, an energy plant needs to shut down a part of its system within given period.

A way to indicate a deadline time for an external event is provided in the real time extension of the toolkit. Consequently, it is now possible to check whether a *deadline* is successfully met by the simulator.

Recently, a non-hierarchical simulation technique has been implemented in CD++ (Glinsky and Wainer 2002a) to provide better performance in the execution of complex DEVS models.

2 TESTING THE REAL-TIME SIMULATOR

In order to analyze the performance of the real-time algorithms used, we developed a testing workbench. We have measured:

- *Number of missed deadlines:* represents the number of deadlines that have been missed along the entire execution of a model. A deadline is missed if its response time is greater than its associated deadline.
- *Worst-case response time:* represents the maximum time between the arrival of an event and the output that the model produces in response, in the entire simulation process.

A wide set of models was tested in order to define accurately the performance of the real-time simulator under different scenarios. In order to make the analysis easier and more accurate, a synthetic experimental frame has been developed and used. The synthetic model generator is able to produce a variety of models, intending to mimic the structure of real applications (Glinsky and Wainer 2002b).

Different parameters are taken into account to analyze a given test case. These parameters are:

1. *Model size:* it can be subsequently divided in *number of components per level* (*i.e.* width) and *number of levels in the model hierarchy* (*i.e.* depth).
2. *Number of interconnections between components:* this parameter describes the complexity and characteristics of the existing interconnections in the model. The information is obtained by the model type (*i.e. Type-1*, *Type-2* or *Type-3*) when the synthetic generator is being used
3. *Workload in transition functions:* the number of milliseconds that have to be spent in the internal and external transition functions
4. *Number of external events:* the number of external events that are received along the entire simulation.
5. *Inter-event period:* the period between an event and the following one. It describes the frequency of event arrival
6. *Associated deadline:* the deadline that has been associated to each incoming event, e.g., a deadline

of 50 milliseconds means that the output for an event has to be issued within 50 milliseconds after its arrival.

The first three parameters are intrinsically related to the model itself. They correspond to the specific characteristics of each model.

On the other hand, the last three parameters are involved with the simulation scenario under which the model is being executed. They are not related to the model, but to the constraints imposed by the user (i.e. the associated deadline) and the environment (i.e. the number of external events and inter-event period).

In the testing process, a wide set of parameters was used to analyze several cases of interest.

Test results show both the *percentage of success* and the *worst-case response time* for each case. The *percentage of success* is obtained as follows,

$$\textit{Percentage of Success} = \frac{\underline{N - \textit{number of Missed Deadlines}}}{N} * 100 \qquad (1)$$

On the other hand, the *worst-case response time* is obtained as follows,

$$\textit{worst-case response time} = \max(r_1, r_2, \ldots, r_N) \qquad (2)$$

where r_i is the response time for the *i-eth* event, and N is the number of events for the given simulation.

The experiments have been grouped in different categories that are explained in the following section.

3 TEST RESULTS

We developed a set of tests analyzing different types of models. Each test used 100 incoming events and a fixed inter-event period of 30 milliseconds. Deadlines are imposed at 60 milliseconds after the event arrival, which would let all events to be processed on time if the simulator would not add overhead to the execution. The following is an excerpt from the event file used in these experiments.

Event time	***Deadline***	***Input port***	***Assoc. output port***	***Value***
`00:05:000`	`00:05:060`	`in`	`out`	`1`
`00:05:030`	`00:05:090`	`in`	`out`	`1`
`00:05:060`	`00:05:120`	`in`	`out`	`1`
`00:05:090`	`00:05:150`	`in`	`out`	`1`
`00:05:120`	`00:05:180`	`in`	`out`	`1`
`00:05:150`	`00:05:210`	`in`	`out`	`1`
`00:05:180`	`00:05:240`	`in`	`out`	`1`
`00:05:210`	`00:05:270`	`in`	`out`	`1`
(…)	(…)	(…)	(…)	(…)
`00:34:910`	`00:34:970`	`in`	`out`	`1`
`00:34:940`	`00:35:000`	`in`	`out`	`1`
`00:34:970`	`00:35:030`	`in`	`out`	`1`

Table 1: External Event file with Associated Deadlines

For instance, the first event arrives through the **in** port at time *00:05:000* and its output must be issued though the **out** port before *00:05:060*. The second event arrives 30 milliseconds later, at *00:05:030*, whereas its associated deadline is *00:05:090*, and so on.

Once the execution is over, we take into account the number of deadlines that have been met and the number of missed deadlines in order to compute the *percentage of success* for each simulation. The *worst-case response times* enable a more comprehensive study of the real time performance. If the worst-case response time is smaller or equal than the associated deadline for a given simulation, then the number of missed deadlines is zero. This means that all events have been completely processed before their associated deadlines, and therefore we achieve a success of one hundred percent.

The testing included Type-1 and Type-3 models. The first is a very simple model type with a small number of interconnections between components. The latter is a much more complex model, with a greater number of interconnections between components (further explanations about the topology and contents of the testing workbench can be found in (Glinsky and Wainer 2002b)).

We started studying models with fixed depth and variable number of components per level. The following figures illustrate the results obtained.

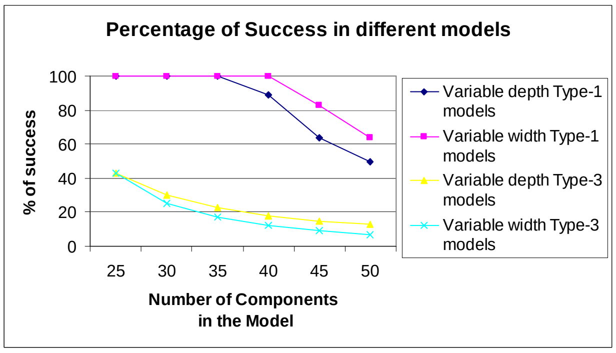

Figure 1: Percentage of Success in Type-1 and Type-3 Models with Variable Width and Depth

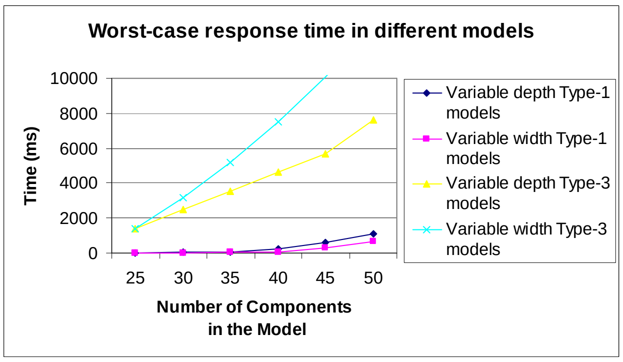

Figure 2: Worst-case Response Time in Type-1 and Type-3 Models with Variable Width and Depth

Figure 1 shows the *percentage of success* for Type-1 and Type-3 models when depth is variable and the width is fixed, and also when the width is variable and the depth is fixed.

Figure 2 illustrates the *worst-case response time* for each case. Similarly, deeper models have worse response times due to their larger size and increased complexity. The same concept applies to wider models, whose size is also increased.

In addition, both charts show that Type-3 models are more difficult to manage due to their complex structure.

As we have explained before, the simulation is carried out by exchanging messages between processors. The previous charts illustrate that if the number of active components is increased, deadlines are more likely to be missed and therefore the *percentage of success* is reduced accordingly. This situation arises because the number messages needed to perform the simulation grows in relation to the size and complexity of the model.

Particularly, the figures show that it is harder to simulate Type-3 models when the number of components increases due to their complex structure, in comparison with the equivalent (and more simple) Type-1 models. Consequently, the *worst-case response times* are remarkably increased for Type-3 models.

Under these conditions, when a Type-1 model has up to 50 active components in its structure, more than fifty-percent of its deadlines are met. In contrast, a Type-3 model with more than 35 active components can achieve less than twenty-percent of success in these experiments. Type-3 models have a larger number of interconnections and, consequently, the incurred overhead makes it harder to complete the entire simulation cycle on time. On the other hand, the deadlines for Type-1 models can be met more easily by the simulator.

A second set of tests considered that events could arrive at a different pace, depending on the surrounding environment. In this experiments, the inter-event periods varied from 20 to 80 milliseconds. Type-3 models were employed in these cases.

In addition, the associated deadlines varied from 0 to 1000 milliseconds, showing how the strictness of deadlines affected the *percentage of success* and the *worst-case response times*.

The following figures show the obtained results after the execution of a given Type-3 model with different frequencies of event arrival and different associated deadlines imposed by the user.

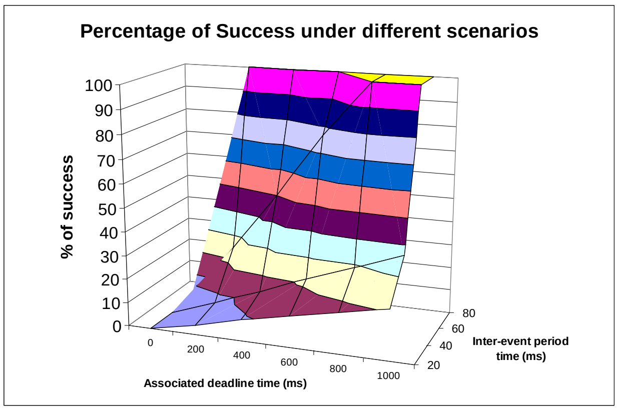

Figure 3: Percentage of Success in Executions in Type-3 Models with Varying Frequency of Events and Strictness of Associated Deadlines

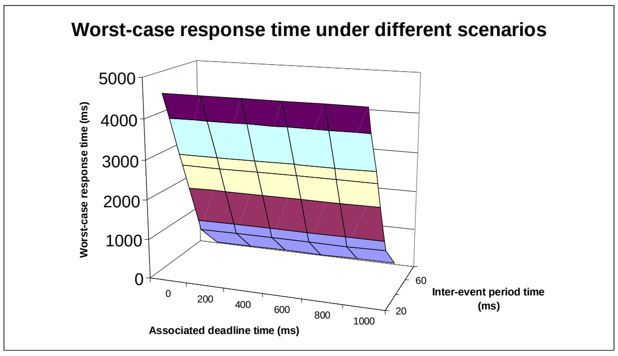

Figure 4: Worst-case Response Time in Type-3 Models with Varying Frequency of Events and Strictness of Associated Deadlines

These charts show the obtained results after the execution of several simulations performed under a combination of different frequencies of event arrival and different strictness on the imposed deadlines.

Figure 3 illustrates not only the favorable impact of greater inter-event period times (i.e. lower frequencies), but also greater associated deadlines (i.e. less strict constraints).

Extremely tight timing constraints do not allow the simulator to meet those deadlines on time. In this particular case, even with deadlines that are not very strict (1000 milliseconds) if the inter-event period is 20 milliseconds, only 20% of the deadlines are met. As deadlines become more relaxed, the percentage of success is correspondingly increased because constraints are more likely to be met, regardless of the frequency of events.

Furthermore, we can point out that with fair constraints and frequencies, simulation results are correct, e.g., when a simulation is executed using 600 milliseconds in deadlines and 60 milliseconds between event arrivals, a success of 77% is achieved in these Type-3 models. If the same frequency of events is received by a simulation whose deadlines are 800 milliseconds, then the achieved success is 100%.

In contrast, Figure 4 shows that worst-case response times remain constant regardless of variations on the associated deadline times. Actually, these experiments show that the response time for an event is unrelated to the timing constraint it might have. However, the inter-event periods have an important effect on the worst-case response times for a given simulation. When the frequency of event arrival is extremely high, the *worst-case response times* are remarkably increased. This situation occurs because excessively small inter-event times do not allow the simulator to process all the messages involved with the event e_k before the arrival of the next event, e_{k+1}. The degradation of performance can be noticed by observing the *worst-case response* time for a given simulation.

The last set of tests to be discussed describe the effect of executing different workloads in the transition functions. Atomic models can execute time-consuming code in their internal and external transition functions. Our synthetic model generator produces Dhrystone code to resemble real workload that would be executed by the atomic components. Dhrystone code is a synthetic benchmark intended to be representative for system (integer) programming (Weicker 1984).

Simulations were run using different workloads in the internal transition function only, in the external transition function only and in both transition functions.

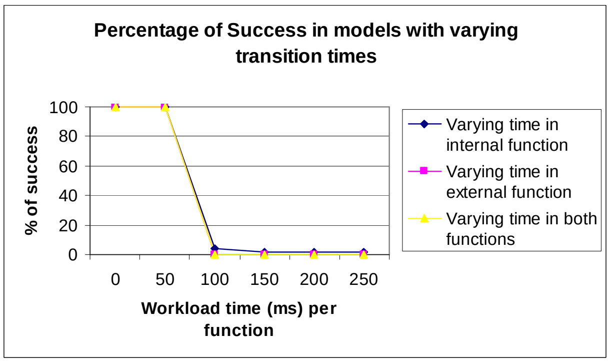

Figure 5: Percentage of Success in Type-3 Models with Varying time in their Transition Functions

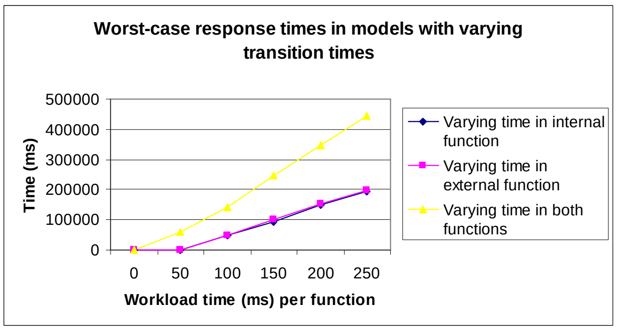

Figure 6: Worst-case Response Time in Type-3 Models with Varying Time in their Transition Functions

Figure 5 shows proper *percentages of success* when the workload time per function is 0 or 50 milliseconds, in spite of the place where the time-consuming code is being executed. In contrast, a noticeable reduction in the *percentage of success* is observed in all cases when the time in the transition functions is increased to 100 milliseconds.

In general, as the workload in the transition functions grows, the *percentage of success* is reduced in all the experiments.

Additionally, Figure 6 shows the *worst-case response time* for each case. The *worst-case response times* are evidently increased, because of the time that has to be spent on executing the atomic transition functions. When the workload is executed in both transition functions, the *worst-case response time* is doubled.

In general, the study shows appropriate results in the execution of *real-time* DEVS models. However, missed deadlines and extremely long response times may arise because of a remarkable degradation of performance. Recently, a non-hierarchical simulator has been developed in the CD++ toolkit to overcome such loss of performance (Glinsky and Wainer 2002a).

4 FLATTENED SIMULATION TECHNIQUE

We have explained before that the simulation process is message-driven; it is based on the message exchange among processors. The message passing consumes an important amount of time, mainly if the model structure is too complex or extremely large (Glinsky and Wainer 2002a; Kim et al. 2000).

The degradation of performance can be minimized if the processor hierarchy is flattened. Thus, the number of exchanged messages can be reduced accordingly and better performance results can be obtained with the flattened simulation approach.

4.1 Test Results Using the Flattened Simulator

A comparison of both hierarchical and flattened simulators is presented here. The executed models have been created using our synthetic model generator. Different sizes and shapes have been employed.

The first set of experiments show the execution of Type-1 models with variable depth (i.e. number of levels in the model hierarchy).

Not only the hierarchical and flattened results, but also the *theoretical results* are shown in the next figure. The theoretical results are simply the sum of all the time spent in executing the workload that is found in the internal and external transition functions. Neither the overhead incurred by the simulators nor any other factors that may affect simulation performance are included in the theoretical results.

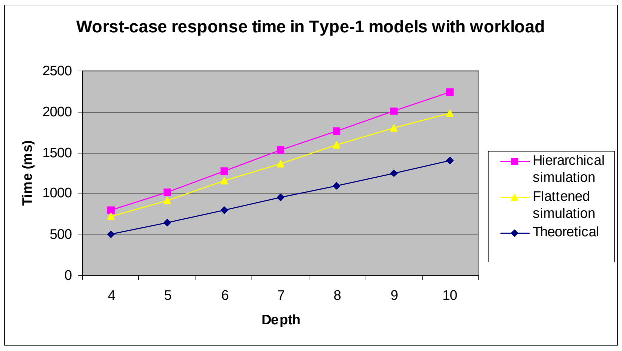

Figure 7: Comparison of Worst-case Response Time Using Hierarchical and Flattened Techniques in Type-1 Models with Variable Depth

Figure 7 shows that the use of the flattened simulation technique provides better response times in the execution of these DEVS models. In deeper models, the difference between the hierarchical and flattened approaches becomes more noticeable.

The previous experiments have analyzed models whose depth was variable. The next chart shows the results for models whose depth is fixed and width (i.e. number of components per level) is variable.

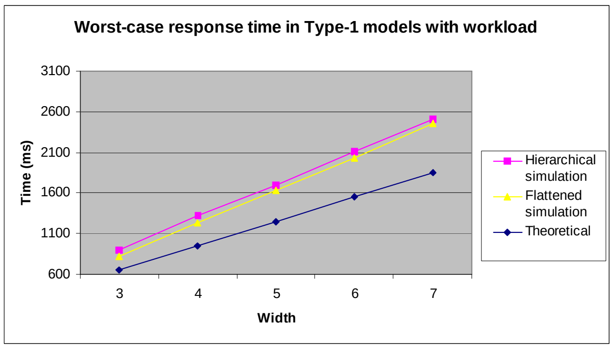

Figure 8: Comparison of Worst-case Response Time Using Hierarchical and Flattened Techniques in Type-1 Models with Variable Width

Figure 8 shows that the flattened simulator outperforms the hierarchical one, providing better response times in *real-time* simulations.

Generally, the analysis suggests that the use of a non-hierarchical simulator allows the execution of larger models with better performance results.

5 CONCLUSIONS

The real-time extension of the CD++ toolkit was tested using a variety of models. We executed small, medium and large models to show the behavior and limitations of the tool under several circumstances. Different model complexities have been used. Moreover, different timing constraints and environments have been studied. The impacts of both the frequency of event arrival and the strictness of the associated deadlines have been analyzed.

The analysis shows adequate performance on most cases, with response times that are quite reasonable for the executed models. Nevertheless, missed deadlines and poor response times may occur when extremely large (or complex) model structures, excessive high frequency on event arrivals or immoderate strictness on the imposed deadlines are considered. Particularly, the accumulation of unprocessed messages is an essential factor that affects performance when the frequency of event arrival is high.

The flattened technique reduces the number of messages exchanged in a simulation. The analysis shows that the non-hierarchical simulator is more efficient, allowing the execution of larger and more complex DEVS models.

REFERENCES

Glinsky, E. and Wainer, G. 2002a. Definition of Real-Time simulation in the CD++ toolkit. In *Proceedings of SCS Summer Computer Simulation Conference*, San Diego, CA.

Glinsky, E. and Wainer, G. 2002b. Performance analysis of DEVS environments. In *Proceedings of AI Simulation and Planning*. Lisbon, Portugal.

Kim, K., Kang W., Sagong, B. and Seo, H. 2000. Efficient Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-Hierarchical One. In *Proceedings of the 33rd Annual Simulation Symposium*. Washington DC.

Rodriguez, D. and Wainer, G. 1999. New Extensions to the CD++ tool. In *Proceedings of SCS Summer Computer Simulation Conference*, Chicago, IL.

Stankovic, J. 1988. Misconceptions about real time computing: A serious problem for next generation systems. *IEEE Computer*, Vol. 21, No. 10, pp. 10-19.

Troccoli, A. and Wainer, G. 2001. Performance Analysis of Cellular Models with Parallel Cell-DEVS. In *Proceedings of SCS Summer Computer Simulation Conference*, Orlando, FL.

Weicker, R. P. 1984. Dhrystone: A synthetic systems programming benchmark. In Communications of the ACM, volume 27, pp. 1013-1030.

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

AUTHOR BIOGRAPHIES

EZEQUIEL GLINSKY is a M. Sc. student in the Computer Sciences Department of the Universidad de Buenos Aires, Argentina. He is a Research and Teaching Assistant in the same department, and a member of the ParDEVS lab. He developed part of this work being a visiting research scholar at Carleton University. Glinsky has worked in the IT industry in Argentina for the past 7 years. Currently, he is an independent IT consultant. His e-mail address is <eglinsky@dc.uba.ar>.

GABRIEL WAINER received the M.Sc. (1993) and Ph.D. degrees (1998, with highest honors) of the Universidad de Buenos Aires, Argentina, and Université d'Aix-Marseille III, France. He is Assistant Professor at the Systems and Computer Engineering, Carleton University (Ottawa, Canada). He was Assistant Professor at the Computer Sciences Dept. of the Universidad de Buenos Aires, Argentina. He has been the PI of several research projects, and participated in different international research programs. Prof. Wainer is a member of the Board of Directors of the Society for Computer Simulation International (SCS). He is also a Co-associate Director of the Ottawa Center of the McLeod Institute of Simulation Sciences. His email and web addresses <gwainer@sce.carleton.ca> and <www.sce.carleton.ca/faculty/wainer/>.