# DEVStone: a Benchmarking Technique for Studying Performance of DEVS Modeling and Simulation Environments

Ezequiel Glinsky Gabriel Wainer
*Dept. of Systems and Computer Engineering*
*Carleton University*
*4456 Mackenzie Building*
*1125 Colonel By Drive*
*Ottawa, ON. K1S 5B6. Canada.*
*{eglinsky, gwainer}@sce.carleton.ca*

## Abstract

*DEVS is a sound, formal modeling and simulation (M&S) framework that supports hierarchical, modular model composition. DEVS-based M&S environments have been used successfully to understand, analyze, and develop a wide variety of systems. As the systems under study become larger and more complex, the performance of the simulator becomes critical. Nevertheless, evaluating the performance of such simulators is a complex process that requires the execution of large numbers of models with different characteristics. We present DEVStone, a synthetic benchmark devoted to automate the evaluation of DEVS-based simulation approaches, which generates models similar to those existing in the real world. DEVStone facilitates performance analysis for successive versions (e.g., upgrades or fixes) of the same simulation engine, and provides a common metric to compare different M&S environments.*

## 1. Introduction

DEVS (Discrete EVents systems Specification) [1] is a sound formal framework for discrete-event modeling based on generic dynamic systems concepts, which supports provably correct, efficient, event-based, distributed simulation. A real system modeled with DEVS is described as a composite of submodels that can be behavioral (atomic) or structural (coupled). The framework supports the construction of models in a hierarchical, modular fashion, allowing component reuse and reducing development and testing time. There is a common preconception that the hierarchical nature of DEVS models may degrade the efficiency of model execution. These arguments are based primarily on intuition, as no study has compared the efficiency of DEVS simulators, nor the different versions of a particular simulation engine. Instead of limiting our effort solely to testing individual models, we developed a synthetic benchmark to aid not only this but also future initiatives in the area. To do so, we introduced the DEVStone benchmark, a synthetic generator that automatically creates models according to our goals. DEVStone generates models with different size, complexity and behavior, resembling different kinds of applications. Hence, it is possible to analyze the efficiency of a simulation engine with relation to the characteristics of a category of models of interest. The tool can be used to assess the efficiency of several DEVS simulation engines, and it provides a common metric to compare the results using different tools.

DEVS is a mathematical formalism with well-defined concepts of hierarchical and modular model construction, coupling of components, and support for discrete event model approximation of continuous systems. The hierarchical nature of DEVS allows coupling existing models in a modular fashion in order to build larger systems. An atomic DEVS model is:

$$M = < X, S, Y, \delta_{int}, \delta_{ext}, l, ta >$$

The model is in state $s \in S$ for the time specified by ta($s$). When that time is consumed, the system outputs the value $\lambda(s)$ and changes immediately to the state $s' = \delta_{int}(s)$. If an external event $x \in X$ is received before the expiration time ta($s$), the new state $s'$ of the system is determined by $\delta_{ext}(s, e, x)$, where $e$ is the time elapsed since the last transition. A DEVS coupled model is described as:

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

where $\boldsymbol{X}$ is the set of input events; $\boldsymbol{Y}$ is the set of output events; $\boldsymbol{D}$ is an index for the components of the coupled model, and $\forall\, i \in D$, $\boldsymbol{M_i}$ is a basic DEVS (*i.e.*, an atomic or coupled model), $\boldsymbol{I_i}$ is the set of influencees of model $i$. The influencees of a model define to which model outputs must be sent. and $\forall\, j \in I_i$, $\boldsymbol{Z_{ij}}$ is the $i$ to $j$ translation function, which converts the outputs of a model into inputs for other models.

Our particular implementation of DEVStone was used to test the performance of CD++ [2], a tool that implements DEVS theory. CD++ was revised and extended several times, and it supports stand-alone [2], parallel [3] and Real-Time simulation [4]. CD++ and has also been successfully used to model a variety of applications: urban traffic [5], [6], [7]; complex physical systems [8], [9]; computer architectures [10]; and different artificial, and biological systems [9].

We used the synthetic generator to analyze the performance of different simulation techniques in CD++, which allowed us to show the feasibility of our approach. Moreover, the performance results permitted us to characterize execution time of a standard DEVS simulator. The benchmark can be used to determine which directions and decisions should be taken when updating the tool's simulation techniques. Furthermore, DEVStone can be used to aid the measurement and improvement of other existing simulation tools.

## 2. Devstone: A synthetic model generator

When analyzing previous studies on the performance of DEVS environments, we can see that they were only focused on performance results for a given tool, and were limited to a given type of models of interest. However, those studies do not provide a thorough analysis for the execution of models with different characteristics, neither do they give a common metric to compare results among different DEVS simulators.

We propose a method to compare not only different versions of a particular simulation engine but also different DEVS-based software. Varied model structures permit obtaining prototypes representative of those found in real world applications. In order to provide a complete and thorough evaluation, we created the DEVStone synthetic model generator, which produces a variety of models with diverse structure and behavior performing a mix of common operations. We focus in the aspects that impact performance: the size of the model and the workload carried out in the transition functions. DEVStone produces models using various parameters: *type* (different structure and interconnection schemes between the components), *depth* (number of levels in the modeling hierarchy), *width* (number of components in each intermediate coupled model), *internal transition time* (execution time spent by internal transition functions), and *external transition time* (execution time spent by external transition functions).

We build an artificial coupled model with *d* coupled components, all of which consist of *w-1* atomic models, with the exception of the lower level of the hierarchy, which is composed of a single atomic model. In addition, internal and external transition functions are programmed to execute a fixed amount of time, consuming CPU clocks by running Dhrystones [11], which consists of a mix of instructions using integer arithmetics. DEVStone uses three different types of models with variations in their internal and external structure:

- **LI**, with a low level of interconnections;
- **HI**, with a high level of input couplings; and
- **HO,** HI models with numerous outputs.

In **LI** models, every coupled component includes only one input and one output port. The input port is connected to each component, and only one component generates an output through the output port in the external component. Fig. 1 shows a sample LI model.

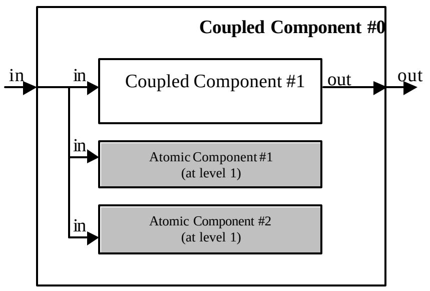

Coupled Component #0
in
in
Coupled Component #1
out
out
in
Atomic Component #1 (at level 1)
in
Atomic Component #2 (at level 1)

**Fig. 1. Example of a LI model: (a) top level.**

As we know the model structure and the time spent by each component in executing transition functions, we can compute the theoretical execution time for the model. First, we devise the number of atomic and coupled models in the structure, which is:

$$\#\,Atomic\ Models = (width - 1) * (depth - 1) + 1$$

Since models follow a predefined interconnection pattern, we can anticipate message routes triggered by external events and time spent in transition functions:

$$\#\,Internal\ Transitions = \#\,Atomic\ Models \qquad (1)$$
$$\#\,External\ Transitions = \#\,Atomic\ Models$$

**HI** models have the same number of atomic components, but more interconnections. Each component $k$ connects its output to the input port of component $k+1$ (with the exception of one atomic component on each level, which does not have any output port). Therefore,

$\#\ Atomic\ Models = (w-1) * (d-1) + 1$
$\#\ Internal\ Transitions = S_{(i=1\ ..\ w-1)}\ i * (d-1)+1$ (2)
$\#External\ Transitions = S_{(i=1\ ..w-1)}\ i * (d-1)+1$

**HO** type models have a more complex interconnection scheme (two input and two output ports in each level. The second input port in the coupled component is connected to its first atomic component. That atomic model connects its output to the second output of its parent). The increased number of interconnections results in more transition functions, and consequently more overhead. For this type,

$\#\ Atomic\ Models = (w-1) * (d-1) + 1$
$\#$ Internal Transitions = $\Sigma_{(i=1\ ..\ w-1)}\ i * (d-1)+1$ (3)
$\#$ External Transitions = $\Sigma_{(i=1\ ..\ w-1)}\ i * (d-1)+1$

DEVStone can be used in any simulator with capabilities for defining and executing Dhrystone code. We can use single-layered models to compare tools with non-hierarchical structure. Likewise, if the chosen modeling technique does not support the execution of internal transitions, we can compare them with DEVS models without internal transitions scheduled.

## 3. Performance of Virtual-time Simulators

CD++ has different simulation engines. A standalone simulator is used in single-processor simulations, while the parallel version (based on [12]) was built on top of Warped [13], which provides different optimistic synchronization algorithms and a non-synchronized protocol (called *NoTime*). We will present the results obtained when DEVStone was applied to characterize the overhead of these simulators.

| | Type | Depth | Width | $d_{int}$ | $d_{ext}$ |
|---|---|---|---|---|---|
| E | HI | 3 | 6 | 50 ms | 50 ms |
| F | HI | 6 | 3 | 50 ms | 50 ms |
| G | HI | 5 | 5 | 50 ms | 50 ms |
| H | HI | 6 | 6 | 50 ms | 50 ms |
| I | HO | 3 | 6 | 100 ms | 0 ms |
| J | HO | 6 | 3 | 0 ms | 100 ms |
| K | HO | 5 | 5 | 50 ms | 50 ms |
| L | HO | 6 | 6 | 50 ms | 50 ms |

**Fig. 2. Simulation parameters**

The first tests are devoted to compare the overhead of three CD++ simulators: (i) original, (ii) parallel with unsynchronized kernel, and (iii) parallel with optimistic kernel. We compared the execution results with the theoretical execution time for each type.These models were executed using 10 external events at a constant rate, each of them triggering a known number of external and internal transition functions. We use the Dhrystone code to compute the time spent on each of these functions, which is used to calculate the time required to process a single event.

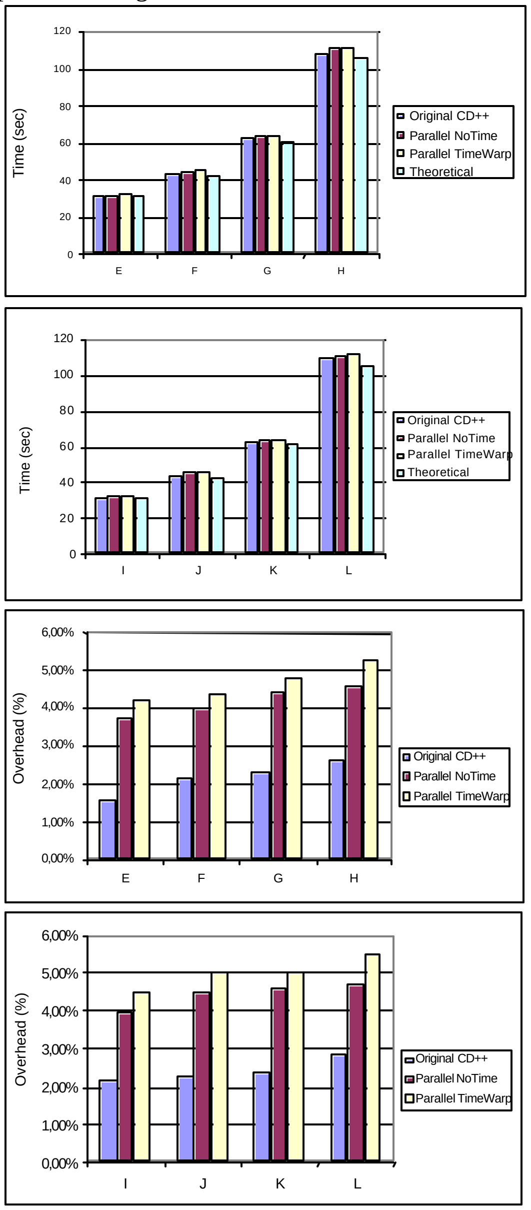


**Fig. 3. Execution times with (a) HI, (b) HO; Overhead (c)HI (d) HO.**

We report the worst execution case from the group (variation for the best execution cases was less than 1%, mainly due to overhead of the OS). The experiments were executed in a single processor, allowing us to measure the pure overhead incurred by the parallel simulator. As expected, the original version provided the best execution time, whereas the parallel unsynchronized kernel (NoTime) version always outperformed the parallel optimistic (TimeWarp) version. The amount of overhead for the original version is only between 1% and 3% in every case. Likewise, the overhead of the parallel versions is below 5.5% for the most complex problems (a promising result, as the amount of speedup time achievable by these simulators is considerable). When we analyze HI models E (3x6) and F (6x3), we can see the greater impact over efficiency when executing models with larger number of levels. Here, both models execute the same number of transition functions (F creates a larger number of intermediate coordinators). Despite these results, we obtained worse overhead for models with a very large number of components (approximately 10,000), due to the number of messages interchanged. CD++ uses the abstract simulator presented in [11], which create a one-to-one correspondence between models and execution engines (called *processors*): *simulators* execute atomic models, and *coordinators* execute coupled models.

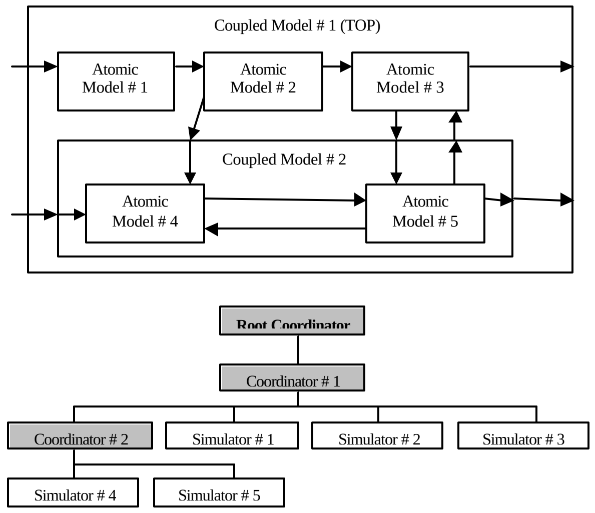


**Fig. 4. Sample model and processor hierarchy**

The number of these intermediate coordinators can be arbitrarily large depending on the model under study. Fig. 5 shows the number of simulators, coordinators, and the number of messages involved in the processing of a single external event for large models.

| | R | S | T | U |
|---|---|---|---|---|
| # Components/level | 100 | 100 | 150 | 150 |
| # of levels | 100 | 100 | 75 | 75 |
| Type | LI | HO | LI | HO |
| # of atomic models | 9.8K | 9.8K | 11K | 11K |
| # of simulators | 9.8K | 9.8K | 11K | 11K |
| # of coupled models | 99 | 99 | 74 | 74 |
| # of coordinators | 99 | 99 | 74 | 74 |
| # of root-coordinators | 1 | 1 | 1 | 1 |
| # of messages | 79K | 3.5M | 89K | 3M |
| Execution overhead | 9.96% | 10.8% | 9.4% | 10.3% |

**Fig. 5. Hierarchical simulation results**

R and S have identical structure, resulting in a remarkable difference in the number of messages involved. The same for T and U: the overhead is approximately 10%. Model U has 12.50% more components than S, although the overhead incurred by CD++ on executing the former is lower. The same happens with models T and R. These results provided a hint to optimize the simulation technique: reducing intermediate coordinators would improve time spent routing messages. A new simulation technique flattens the simulator [14], reducing the number of messages exchanged. The idea is to create only one root coordinator and one flat coordinator. The flat coordinator executes the $\delta_{int}$, $\delta_{ext}$ and $\lambda$(s) functions for each atomic component, mapping the ports for all atomic and coupled components in the hierarchy.

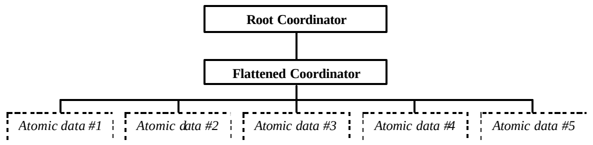


**Fig. 6. Flat simulator approach for Fig.4**

We applied DEVStone to this new version of CD++, and compared the results with the hierarchical version. Figure 7 shows the execution results for the parameters presented in Figure 5. As simulators and coordinators disappeared, and one flat coordinator is created regardless of the number of components, the resulting overhead is close to 5%.

| | R | S | T | U |
|---|---|---|---|---|
| # Components/level | 100 | 100 | 150 | 150 |
| # of levels | 100 | 100 | 75 | 75 |
| Type | LI | HO | LI | HO |
| # of atomic models | 9.8K | 9.8K | 11K | 11K |
| # of simulators | 0 | 0 | 0 | 0 |
| # of coupled models | 99 | 99 | 74 | 74 |
| # of coordinators | 0 | 0 | 0 | 0 |
| # of root-coordinators | 1 | 1 | 1 | 1 |
| # of flat coordinators | 1 | 1 | 1 | 1 |
| # of messages per single external event | 4 | 9.8K | 77 | 11K |
| Execution overhead | 4.5% | 5.6% | 4.4% | 5.2% |

**Fig. 7. Flat simulation results**

Figure 8 compares both simulators, showing the difference between actual and theoretical execution times for different models with variable depth (width 6), and with variable width (depth 7). Each components has a workload of 50 ms for the transition functions, and models receive 100 events at a fixed frequency.

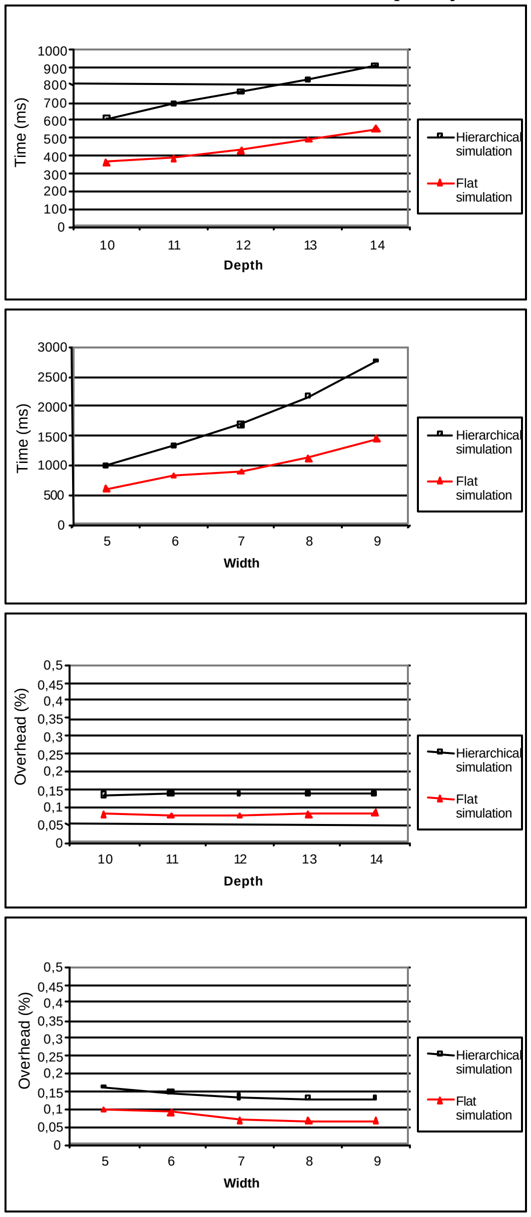


**Fig. 8. Difference theoretical/ execution times: (a) LI, (b) HO; Overhead comparison of hierarchical and flat simulators: (c) LI, (d) HO**

Regardless of the type of model (LI or HO) and the structure (deeper or wider), we can see a clear improvement in execution times when using the flat approach. W hen executing a small LI model (10x6) the difference between theoretical and execution time is reduced in 38.3%. For larger LI models the reduction remains stable or even rises marginally (*e.g.*, for the a 14x6 model, it is 39.6%). Similar results can be seen in the execution of these HO models, which are more complex and have approximately the same size. In general, the reduction obtained for these HO models is in the range of 39.4%-47.7%. The overhead is always less than 0.16% for the hierarchical simulator, and 0.1% for the flat simulator.

Even though models in Figure 8 (a) are larger than those in Figure 8 (b), the latter ones have more complex structures that compensate the differences in size, resulting in similar percentages of overhead. The flat simulator reduces the overhead in 40%-56% depending on the depth, width, and complexity of the model, although it is important to note that the overhead is stable for both techniques.

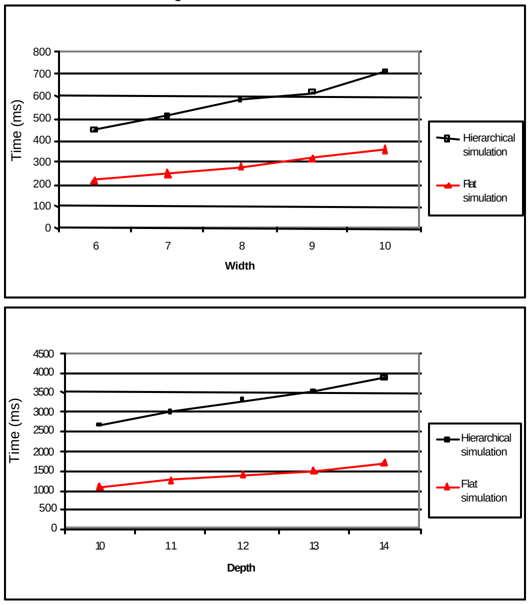


**Fig. 9. Hierarchical/flat simulators: (a)LI (b)HO**

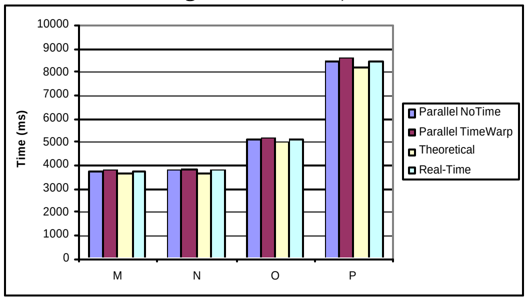
In order to analyze the performance degradation purely due to overhead in the simulation engine, we executed several examples with empty transition functions (total execution time is solely depending on the message exchange). Figures 9 (a) and (b) show the execution of several LI models with variable width and fixed depth of 8, and HO models with variable depth and fixed width of 8 (*i.e.*, models that have between 36 and 92 components). Figure 9 (a) shows that regardless of the model's width, the flat simulator reduces the execution time in 52.4%-54.7%. Figure 9 (b) shows that for HO models the improvement in performance becomes more noticeable when the depth of the model increases.

The impact of the intermediate coordinators that are eliminated from the hierarchy results in fewer messages being exchanged, and more efficient simulation. The obtained results are similar to those presented before; the flat approach provides a reduction of 52%-58% in the total execution time depending on the size and type of the model

## 5. Performance analysis of RT simulators

CD++ was extended to allow RT execution [4]. In RT systems, a correct answer after a deadline is regarded as unsuccessful. RT CD++ allows the execution of events triggered by the RT clock, enabling interaction between the models and their environments. In order to analyze the performance of the new simulator, we used DEVStone to study the performance of different models. We started comparing the virtual-time and RT techniques. Our approach consisted in executing virtual-time and RT simulations using DEVStone, and comparing the time required to process a single event in virtual time against their worst-case response.

Figure 10 shows a comparison for sample LI models M (5x10), N (10x5), O (8x8), and P (10x10), with internal and external functions executing 50 ms of Dhrystone code. In all cases, models received 100 events at a constant rate, and we measured the worst-case response time. Since the new functionality that deals with deadlines adds a small amount of additional overhead in the RT version, it was expected to observe some degradation in performance. Nevertheless, the experiments showed that the added overhead is actually imperceptible. In general, we see that the RT results are in the same level of those obtained with the parallel unsynchronized version (incurring overheads in the range of 2%-3.5%) and outperforming optimistic TimeWarp (with overheads in the range of 3%-4.6%).



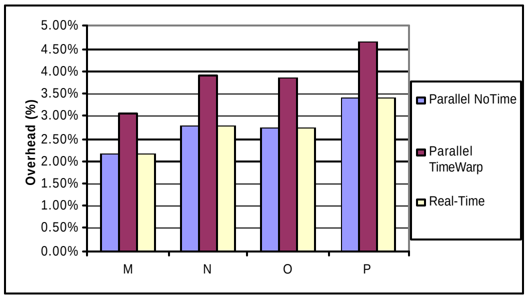


**Fig. 10. Comparing RT and virtual-time simulators: (a) Execution times (b) overhead**

We used DEVStone to measure the ability of CD++ to execute models with different structure under very demanding situations. Simulations received a fixed number of external events generated at a constant rate. Each model was expected to deliver responses for each event before a given deadline, and the percentage of success and worst-case response time:

$$\%\ success = \frac{(\#\ events - \#\ missed\ deadlines) * 100}{number\ of\ events}$$

$$Worst\text{-}case\ response\ time = max\ (r_1, r_2, \ldots, r_N)$$

where $r_i$ is the response time for the *i-th* event, and $N$ is the number of events.

The models used in the following experiments had different sizes ranging from 10 to 35 components in total (width and depth are in the range of 4 to 11). There is no Dhrystone code generated in the internal or external functions (all the time spent is overhead ). The results are grouped in four categories: (1) LI models with variable depth, (2) LI models with variable width, (3) HO models with variable depth, and (4) HO models with variable width. In the first set of experiments, each simulation receives 100 events with inter-event periods of 100 ms, and associated deadlines were set at 100 ms.

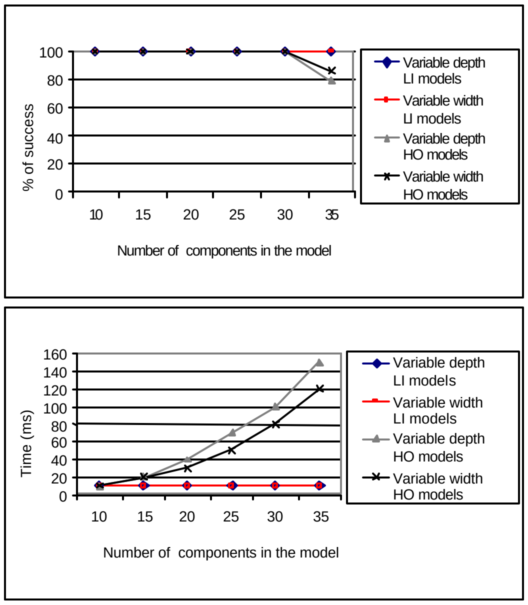


**Fig. 11. Comparing variable depth/width (a) Percentage of success, (b) WCET**

Figure 11 (a) shows the percentage of success for LI and HO models when depth is variable and width is fixed, and also when width is variable and depth is fixed, whereas Figure 11 (b) illustrates the worst-case response time for each case. We observe that the RT simulator was able to deliver all the outputs on time for models with 30 or fewer components, regardless the model type. However, when the size of the model is 35, HO models started missing a few deadlines. The pace of external events and the complexity of the model prevent a timely response to some of the external events. For a HO model with 35 components, the success was 79% and 86%. In contrast, LI models always met all their deadlines under the same circumstances (a result of the greater complexity of HO models). Figure 11 (b) illustrates that the worst-case response time is gradually deteriorated for HO models as a result of the increased number of messages exchanged. LI models always had a worst-case response time of 10 ms in these conditions, despite the size of the model. These results provide a reference about the conditions in which the engine is capable of meeting the deadlines, focusing on a particular scenario (inter-event periods and associated deadlines) and the characteristics of the model (size and model type). In the next set of experiments, we executed larger LI and HO models (with between 25 to 50 components). We focused on the performance of the RT simulator under a more stressful scenario. This test focuses on the cases where the simulator is unable to meet its deadlines due to highly overloaded conditions. External events ($e_i$) arrive every 30 milliseconds and deadlines ($d_i$) are set at 60 ms after their arrival.

An alternative analysis can be performed focusing on the surrounding environment; for instance, studying the effect of different inter-event periods (*i.e.*, frequency of event arrivals) on the execution performance. In the following set of experiments, events arrive at different pace (20 to 180 ms), and we analyze the behavior of the simulator under such circumstances. The charts show the results for 8x8 HO models receiving 100 events with deadlines set at 1000 ms.

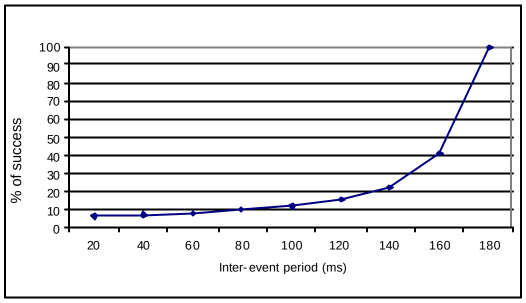
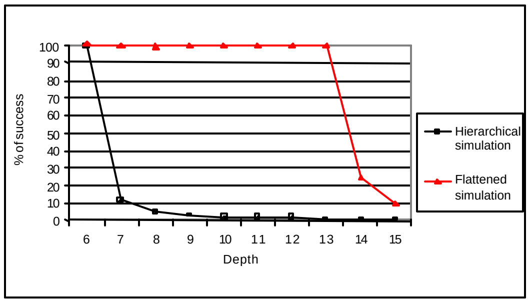


**Fig. 12. Worst-case execution time for HO models with variable inter-event period:**

Figure 12 shows that larger inter-event periods result on greater percentages of success. When the intervals between events become greater than 180 milliseconds, the simulator meets all the associated deadlines for the execution of this 8x8 model.

In the last set of experiments, we contrasted the RT performance of the flat simulator against that of the hierarchical simulator. We analyzed both simulators executing HO models with variable depth and width, focusing on complex models and examining their performance under demanding conditions. The next chart shows the results for models with fixed width of 9 and variable depth (from 6 to 15 levels) that received 100 events at a fixed arrival rate.



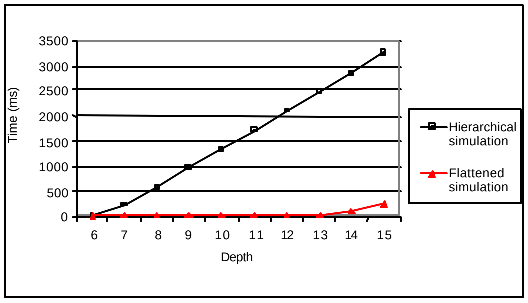


**Fig. 13. Comparison of hierarchical and flat: (a) % of success (b) Worst-case response time**

The flat approach allowed the simulation of a 13x9 HO model (97 components) with minimum worst-case response times, while meeting all the deadlines. When using the flat approach, performance degradation is first noticed in the execution of a 14x9 model (105 components). In contrast, even when simulating a smaller 7x9 model, the hierarchical approach had an inferior percentage of success (87%) and showed worse response times. Figure 13 (b) shows that in larger models, the difference becomes more noticeable in terms of the worst-case response times, as a result of the greater overhead incurred in the simulation of deeper models. Our analysis showed similar results when models with variable width were executed. Similarly to what was illustrated before, the flat coordinator simulated wider models more effectively.

In general, the flat technique outperforms the hierarchical one, reducing the incurred overhead up to 50% and therefore providing improved response times and better percentage of success in the execution. Thus, the use of the non-hierarchical approach allows the simulation of larger models with better performance results. These results are a consequence of the reduced number of messages exchanged in the flat simulation mechanism.

## 6. Conclusions

Evaluating the performance of a simulation tool is typically a tedious and complex process, which requires the execution of a wide variety of models with different characteristics. Our main goal was to provide a means for evaluating the efficiency of existing simulation engines with focus on DEVS-based tools, and facilitating a qualitative and objective comparison of different tools.

We developed DEVStone, a synthetic model generator that supports the process of evaluating the performance of simulation engines. DEVStone produces models that are similar to those existing in the real world, thus making it possible to: (i) create models with different sizes, shapes, and behavior; (ii) generate an arbitrarily large number of such models; and (iii) execute those models using the simulator(s) under study.

DEVStone relies on executing a collection of models with different characteristics. In order to emulate several degrees of complexity in their structures, we identified three types of models that correspond to three interconnection patterns. In addition, each atomic component usually executes code in its transition functions; we proposed the use of Dhrystone code to resemble the task to be performed by these components.

As a result, we have a systematic way to assess the performance of a simulation engine, reducing the time required to measure its efficiency. It is possible to analyze the efficiency of any DEVS simulator with relation to the size, the behavior and the structure of the model under execution. A precise performance characterization of a simulator allows modelers to consider the actual overhead of the tool based on solid results, and then analyze the feasibility of executing timed models with specific timing constraints.

Our framework provides a common metric to compare the results that were obtained using the different simulation tools, and also enables an analysis of the efficiency of successive versions of the same simulator, such as upgrades or fixes. We used the CD++ toolkit to show how to apply the proposed benchmark. These experiments allowed us to test the usefulness of the benchmark, and to thoroughly test CD++ (which is the first systematic effort to characterize the performance of DEVS modeling and simulation environments). Although we restricted our case study to the existing CD++ simulation engines, the same ideas may hold for other DEVS-based simulators. Using DEVStone, we showed that hierarchical simulation techniques are capable to simulate models with low overhead, even for models with complex structure. By means of the proposed framework, the performance of both virtual-time and RT hierarchical simulators was shown to be satisfactory. Moreover, the results demonstrated that the flat simulation technique could improve the efficiency in some cases, especially when model structure is particularly large or complex. Regardless of the size and complexity of the models, the flat simulator outperformed the hierarchical one. In general, the charts illustrate that the overhead incurred by the flat simulator is reduced up to about 55% of the overhead incurred by the hierarchical approach.

## References

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

[2] Wainer, G. "CD++: a toolkit to develop DEVS models". Software - Practice and Experience. vol. 32, pp. 1261-1306. 2002.

[3] Troccoli, A.; Wainer, G. "Implementing Parallel Cell-DEVS". Proceedings of 36th IEEE/SCS Annual Simulation Symposium. Orlando, USA. 2003.

[4] Glinsky, E.; Wainer, G. "Definition of RT simulation in the CD++ toolkit". *Proc. of the Summer Computer Simulation Conference*. San Diego, CA. 2002.

[5] Davidson, A.; Wainer, G. "Specifying truck movement in traffic models using Cell-DEVS". *Proc. of the 33rd Annual Simulation Symposium*. Washington, DC. 2000.

[6] Lo Tartaro, M.; Torres, C.; Wainer, G. "Defining Models of Urban Traffic using the TSC Tool". *Proc. of the Winter Simulation Conference.* Washington, DC. 2001.

[7] Díaz, A.; Vázquez, V.; Wainer, G. "Application of the ATLAS language in models of urban traffic". *Proc. of the Annual Simulation Symposium.* Seattle, WA. 2001.

[8] Ameghino, J.; Troccoli, A.; Wainer, G. "Models of complex physical systems using Cell-DEVS". *Proc. of the Annual Simulation Symposium.* Seattle, WA. 2001.

[9] Ameghino, J.; Wainer, G.; Glinsky, E. "Applying Cell-DEVS in Models of Complex Systems". *Proc. of the Summer Computer Simulation Conference.* Montreal, QC. 2003.

[10] Wainer, G.; S. Daicz, S.; De Simoni, L.; Wasserman, D. "Using the ALFA-1 simulated processor for educational purposes". *ACM Journal on Educational Resources in Computing*. 1(4). pp. 111-151. 2001.

[11] Weicker, R. P. "Dhrystone: A synthetic systems programming benchmark". *Communications of the ACM*, volume 27, pages 1013-1030, 1984.

[12] Chow, A.; Zeigler, B. "Parallel DEVS: A parallel, hierarchical, modular modeling formalism". *Proc. of the Winter Simulation Conference*. Orlando, FL. 1994.

[13] Martin, D.; McBrayer, T.; Radhakrishan, R.; Wilsey, P. "Time Warp Parallel Discrete Event Simulator". Technical report. Computer Architecture Design Laboratory. University of Cincinnati. USA. 1997.

[14] Kim, K.; Kang W.; Sagong, B.; Seo, H. "Efficient Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-Hierarchical One". *Proc. of the 33rd Annual Simulation Symposium.* Washington DC, USA. 2000.