A UNIFIED BENCHMARKING FRAMEWORK FOR EVALUATING DISCRETE EVENT SIMULATION ENGINES

Beatriz Herguedas-Pinedo[a], Román Cárdenas[b c], Patricia Arroba[b c],
Eva Besada-Portas[a], and José L. Risco-Martín[a]

[a]Department of Computer Architecture and Automation, Universidad Complutense de Madrid, Spain
[b]Department of Electronic Engineering, Universidad Politécnica de Madrid, Spain
[c]Center for Computational Simulation, Universidad Politécnica de Madrid, Spain

ABSTRACT

Benchmarks are essential tools in computer science for measuring hardware and software performance. In the Discrete-Event System (DES) community, improving simulation performance is crucial. However, comparing different DES simulators is difficult due to the lack of standardized metrics, complicating direct comparisons. A well-defined benchmark suite would help researchers evaluate simulation engine improvements. This paper proposes a set of metrics for assessing DES performance across sequential, parallel, and distributed architectures. Our approach is based on the Discrete Event System Specification (DEVS) formalism, defining parameters such as model structure and event count, with a focus on total simulation time. This framework provides a consistent method for evaluating DES performance, facilitating direct simulator comparisons. We tested the benchmark suite on various discrete event simulators, including both DEVS-based and non-DEVS-based engines, demonstrating its versatility and effectiveness in assessing performance across different platforms. Our results highlight the benchmark's ability to improve simulation evaluation.

Keywords: Discrete-event simulation, DEVS, DEVStone, performance, benchmarking.

1 INTRODUCTION AND RELATED WORK

Modeling and Simulation (M&S) has become an essential tool in scientific research, complementing theoretical and experimental studies. It allows researchers to create and analyze models of complex systems, providing insights that are often difficult to obtain through traditional methods. Computational science treats science as modeling, where executing scientific models on a computer constitutes simulation [1]. Consequently, M&S is now recognized as the third pillar of science [2]. The design and analysis of complex systems rely heavily on M&S, aiding in training, forecasting, and decision support.

Over the past decades, numerous M&S techniques and tools have emerged, providing robust Application Programming Interfaces (APIs) for system modeling and simulation. These tools are designed to cater to a wide range of applications, from simple process modeling to complex system simulations. Given the abundance of M&S frameworks, selecting an optimal Discrete-Event System (DES) M&S tool is challenging.

Several commercial options are available, each offering unique features and capabilities. **AnyLogic** is a versatile tool that integrates multiple modeling paradigms, allowing users to simulate complex systems using a combination of discrete event, agent-based, and system dynamics models [3]. **Arena** is another popular choice, known for its ability to model both discrete and continuous processes, making it suitable for a variety of industrial applications [4]. **MS4 Modeling Environment** leverages the Discrete Event System Specification (DEVS) formalism to support hybrid models, providing a flexible platform for simulating systems

Proc. of the 2025 Annual Modeling and Simulation Conference (ANNSIM'25)
May 26-29, 2025, Universidad Complutense de Madrid, Madrid, Spain
J.L. Risco-Martín, G. Rabadi, D. Cetinkaya, R. Cárdenas, S. Ferrero-Losada, and A. Bany Abdelnabi, eds.

with both discrete and continuous components [5]. **Plant Simulation** focuses on optimizing production systems, offering tools for simulating and analyzing manufacturing processes to improve efficiency and reduce costs [6]. **SimEvents**, integrated into MATLAB/Simulink, allows users to incorporate DES into their existing MATLAB workflows, facilitating the simulation of event-driven systems [7].

In addition to commercial tools, several open-source DES simulators are available, providing cost-effective alternatives for researchers and practitioners. **aDEVS** is a C++ library that supports DEVS-based simulations, offering a high-performance platform for modeling discrete event systems [8]. **Ptolemy II** is designed for actor-oriented design, supporting the modeling and simulation of embedded systems with a focus on concurrent execution [9]. **Cadmium** is another DEVS-based simulator that facilitates the modeling of complex systems and supports the Cell-DEVS formalism for spatially distributed systems [10]. **SystemC** provides an event-driven simulation kernel, widely used for modeling hardware systems and embedded software [11]. **xDEVS** is a multi-platform DEVS modeling tool that offers a flexible environment for simulating discrete event systems across different platforms [12].

Despite the diversity of available tools, evaluating their performance remains a complex task. A standardized benchmark suite is crucial for fair comparisons, enabling researchers to assess the strengths and weaknesses of different simulation engines effectively. The DES community has developed several synthetic benchmarks to address this challenge, such as PHOLD [13], LA-PDES [14], and EPHOLD [15]. These benchmarks are designed to assess the performance of simulators, particularly in parallel or distributed computing environments. PHOLD and EPHOLD, for instance, are primarily used to evaluate the efficiency of parallel and distributed simulations by focusing on the communication and synchronization overheads inherent in such systems. LA-PDES allows for parameterized performance evaluation, enabling controlled experimentation with various parameters to understand their impact on simulation performance. Despite their utility, these benchmarks have limitations. They often focus on specific aspects of computing architecture rather than the intrinsic characteristics of DES models. This focus can lead to evaluations that do not fully capture the computational demands of different simulation engines. Moreover, the absence of standardized performance metrics complicates comparisons between studies, as different researchers may use varied criteria, leading to inconsistencies in performance assessments.

A new methodology is needed to address the limitations of existing benchmarks and provide a comprehensive evaluation framework for DES engines. We propose a unified framework that combines the DEVStone benchmark [16] with standard performance metrics to offer a consistent and reliable method for assessing DES simulators. Our approach focuses on using total simulation time as a universal metric, which simplifies the comparison of different simulators by providing a single, consistent measure of performance. We use the geometric mean to aggregate simulation times, ensuring that the evaluation is platform-independent and representative of the simulator's capabilities. Additionally, our framework incorporates the order of complexity of DEVStone models as a weighting factor in the aggregation process. This ensures that more complex models have a greater impact on the overall performance metric, accurately reflecting their computational demands. This facilitates straightforward comparisons between different simulators, whether they are DEVS-based or not.

The remainder of the paper is structured as follows: Section 2 provides background, Section 3 examines performance metrics, Section 4 details our experimental approach, Section 5 presents a use case scenario, and Section 6 concludes our study.

2 BACKGROUND

DES is a modeling approach in which the state of a system changes instantaneously in response to events within a continuous time base. Popular DES techniques include Markov chains, Petri nets, and DEVS.

This section focuses on DEVS and the DEVStone benchmark, used in this work as the standard modeling language and benchmarking tool, respectively.

2.1 The DEVS Formalism

DEVS provides a formal framework for discrete-event modeling and simulation. Systems are represented as atomic or coupled models. Atomic models define autonomous behavior using states, transitions, and event reactions:

$$AM = \langle X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle, \tag{1}$$

where X and Y are input and output sets, S is the state set, δ_{int} and δ_{ext} are internal and external transition functions, δ_{con} handles event collisions, λ generates outputs and ta is the time advance function.

On the other hand, coupled models connect atomic or other coupled models to form larger systems:

$$CM = \langle X, Y, C, EIC, EOC, IC \rangle, \tag{2}$$

where X and Y are the input and output sets, C is the set of DEVS submodels, EIC defines external input coupling, EOC defines external output coupling, and IC defines internal coupling between components.

DEVS supports multi-formalism integration, representing hybrid systems such as differential equations and automata, making it suitable for applications in decision support, logistics, and traffic analysis [17].

2.2 The DEVStone Benchmark

DEVStone is a synthetic benchmark designed to evaluate DES simulators using DEVS. One of its key characteristics is that, once the parameters are selected, the sequence of events in a simulation is deterministic. This is an advantage because it provides absolute control over the experiment's complexity, allowing precise assessment of the computational effort required.

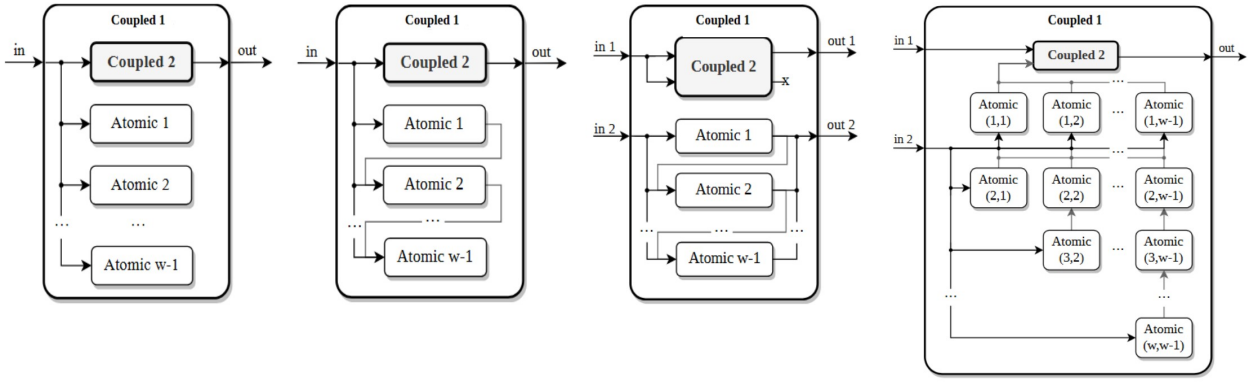
It generates models to test specific simulation aspects, defined by five parameters:

- **Type**: Model layout.
- **Depth** (d): Number of nested coupled levels.
- **Width** (w): Number of atomic models per level.
- **Internal delay** (Δ_{int}): CPU time for internal transitions.
- **External delay** (Δ_{ext}): CPU time for external transitions.

Execution times are measured using Central Processing Unit (CPU) time to ensure adherence to delays. Atomic models are activated on input, process events, and return to a passive state after executing transitions. DEVStone's flexibility and parameterization make it a suitable candidate for assessing simulator performance [18].

DEVStone includes four types of coupled models: LI, HI, HO, and HOmod (Figure 1). Simulations begin with an event generator sending an integer to all input ports of the top-level coupled model. The simulation ends when no pending events remain.

LI models have a simple structure, with d levels where the innermost level contains one atomic model. The number of internal and external transitions of atomic models that are triggered in a simulation is described as follows:



(a) LI model. (b) HI model. (c) HO model. (d) HOmod model.

Figure 1: DEVStone model types.

$$\#\delta_{\text{int}} = \#\delta_{\text{ext}} = \#\text{Events} = 1 + (d-1)\cdot(w-1). \tag{3}$$

HI models extend LI by introducing internal couplings (IC), increasing the number of events. Changes in the number of internal and external transitions are reflected as:

$$\#\delta_{\text{int}} = \#\delta_{\text{ext}} = \#\text{Events} = 1 + (d-1)\cdot\sum_{i=1}^{w-1} i. \tag{4}$$

HO models build on HI, coupling all atomic components in a module to the output port. This increases external couplings. The changes in the number of internal and external transitions are defined as:

$$\#\delta_{\text{int}} = \#\delta_{\text{ext}} = \#\text{Events} = 1 + (d-1)\cdot\sum_{i=1}^{w-1} i. \tag{5}$$

HOmod models exponentially increase coupling and outputs. Rows and columns in $d-1$ levels follow a decremental structure, with characteristics for internal and external transitions defined as follows:

$$\#\delta_{\text{int}} = \#\delta_{\text{ext}} = 1 + (d-1)\cdot(w-1)^2 + \left((d-1) + (w-1)\cdot\sum_{i=1}^{d-2} i\right)\cdot\left((w-1) + \sum_{i=1}^{w-1} i\right). \tag{6}$$

DEVStone is easily extensible, allowing new topologies to be incorporated, as shown in [19]. This adaptability improves its utility for DES simulator performance evaluation.

3 PERFORMANCE METRICS ANALYSIS

The DEVStone model set simplifies performance evaluation by focusing on execution time as the primary metric, limiting parameters such as the number of components, events triggered, and computation density. However, a method to aggregate simulation times across multiple models into a single value is still required.

This section proposes measuring the order of complexity of a functioning DES model based on the number of state transitions generated in a simulation. This approach makes sense because state transitions directly reflect the computational effort required by the simulation, providing a meaningful indicator of model be-

havior. The analysis is conducted using DEVStone configuration parameters, incorporating the order of complexity as a weighting factor in the aggregation process.

3.1 Order of complexity in DEVStone Models

For evaluating the performance of discrete event simulation (DES) engines, we define the order of complexity of a model, denoted as $\mathcal{C}(m_z^T)$, to represent the computational demand required to simulate a given model. Here, $T \in \mathbf{T} = \{\text{LI}, \text{HI}, \text{HO}, \text{HOmod}\}$ refers to the model type or topology, and $z \in Z_T = \{w,d\}_T$ a particular size or shape. This concept is similar to the big-O notation used in computer science to describe the upper bound of an algorithm's running time. $\mathcal{C}(m_z^T)$ is then defined as:

$$\mathcal{C}(m_z^T) = min\{g(\mathbf{x}) | f(\mathbf{x}) \in O(g(\mathbf{x})), \text{ and g(x) is asymptotically the tightest bound}\}. \tag{7}$$

For the DEVStone set of models, the order of complexity $\mathcal{C}(m_{w,d}^T)$ is determined by the number of transitions that occur during the simulation. These transitions are influenced by the model's type (T), width (w), and depth (d), which are parameters that define the structure and size of the model (see equations (3) to (6)).

For **LI** models, the order of complexity is directly proportional to the product of width and depth. This reflects the linear increase in transitions as the model's size grows:

$$\mathcal{C}(m_{w,d}^{LI}) = \mathcal{C}(1 + (d-1)\cdot(w-1)) = w \cdot d. \tag{8}$$

HI models introduce additional internal couplings. The number of transitions grows quadratically with the width and linearly with the depth, leading to a quadratic relationship between order of complexity and model parameters:

$$\mathcal{C}(m_{w,d}^{HI}) = \mathcal{C}\left(1 + (d-1)\cdot\sum_{i=1}^{w-1} i\right) = \mathcal{C}\left(1 + (d-1)\cdot\frac{w^2-w}{2}\right) = w^2 \cdot d. \tag{9}$$

HO models share the same order of complexity as HI models, due to their similar structure and coupling patterns. The additional output couplings in HO models do not alter the overall complexity order compared to HI models:

$$\mathcal{C}(m_{w,d}^{HO}) = \mathcal{C}(m_{w,d}^{HI}). \tag{10}$$

HOmod models exhibit the highest order of complexity, which reflects the exponential increase in transitions due to the extensive coupling. This makes HOmod models the most computationally demanding, with a cubic relationship to width and a quadratic relationship to depth:

$$\begin{aligned}
\mathcal{C}(m_{w,d}^{HOmod}) &= \mathcal{C}\left(1 + (d-1)\cdot(w-1)^2 + \left((d-1) + (w-1)\cdot\sum_{i=1}^{d-2} i\right)\cdot\left((w-1) + \sum_{i=1}^{w-1} i\right)\right) \\
&= \mathcal{C}\left(w^2 \cdot d + (w\cdot d^2)\cdot(w^2)\right) \\
&= w^3 \cdot d^2.
\end{aligned} \tag{11}$$

These order of complexity relationships highlight the importance of model structure in determining computational load. They also allow us to better interpret the performance metrics and their implications for different simulation engines.

3.2 Metrics for Performance Analysis

Next, we consolidate the methodology for measuring the performance of a DES simulation engine using a Pythagorean metric applied to a set of benchmark models. The selected metric incorporates features such as using a simulator as a reference and including weights in the aggregation. In addition, we discuss experimental setups for sequential and parallel computing architectures.

As stated above, a benchmark model is denoted as m_z^T. The total execution time for a simulation engine S running m_z^T, denoted as $t_S(m_z^T)$, includes the loading, configuration, and simulation times:

$$t_S(m_z^T) = t_S^{\text{load}}(m_z^T) + t_S^{\text{conf}}(m_z^T) + t_S^{\text{sim}}(m_z^T). \tag{12}$$

We suggest using the geometric mean to combine execution times from different models for performance evaluation. This method helps avoid the problems of the arithmetic mean, which can make slower models seem more significant when combining results from different setups. The geometric mean gives a better understanding of simulator performance, even when the data isn't standardized. Additionally, we use order of complexity $\mathcal{C}(m_z^T)$ as a weighting factor. This means that more complex models have a bigger impact on the overall metric, accurately showing their effect on performance, i.e., our proposed metric ensures a fair and precise combination of results:

$$\hat{t}_S^T = \left(\prod_{z \in Z_T} t_S(m_z^T)^{\mathcal{C}(m_z^T)} \right)^{\frac{1}{\sum_z \mathcal{C}(m_z^T)}}. \tag{13}$$

Equation (13) presents some issues that, under certain circumstances, make it impossible to compute $\hat{t}_S^T$ due to problems with floating-point arithmetic. For example, when $t_S(m_z^T)$ is less than 1 second or when there are total times with different orders of magnitude. Furthermore, an exponential operation over a number close to zero or a large number may lead to execution exceptions. To avoid these issues, we represent the geometric mean in the following form:

$$\hat{t}_S^T = exp\left(\frac{\sum_{z \in Z_T} \mathcal{C}(m_z^T) \cdot ln(1 + t_S(m_z^T))}{\sum_{z \in Z_T} \mathcal{C}(m_z^T)} \right). \tag{14}$$

Eq. (14) is the aggregated metric selected to evaluate the performance of $|Z_T|$ sequential simulations of DES benchmark models of a given topology T using a DES simulator S.

When benchmarking performance results, it is common to normalize results relative to a reference system. In this work, the normalization uses a DES engine R, known for its superior sequential performance due to its efficient event scheduling, minimal computational overhead, and well-optimized architecture. The Reference Performance Unit (RPU) metric, defined like a traditional speed-up over $\hat{t}_{\text{R}}^T$:

When evaluating performance results, it's common to use a reference system R for normalization. We define the RPU metric, which is similar to a traditional speed-up metric, to compare the performance of a given simulator against this reference. The RPU is calculated as follows:

$$\hat{s}_S^T = \frac{\hat{t}_R^T}{\hat{t}_S^T}. \tag{15}$$

This equation allows us to assess how much faster or slower a simulator is compared to the reference system for a specific set of benchmark models. To summarize performance across different model types, we use the geometric mean:

$$\hat{t}_S = \left(\prod_{T \in \mathbf{T}} \hat{t}_S^T \right)^{\frac{1}{|\mathbf{T}|}}. \tag{16}$$

This approach provides a single, comprehensive metric that reflects the overall performance of a simulator across various model types. We then normalize this aggregated metric to obtain a global RPU value:

$$\hat{s}_S = \frac{\hat{t}_R}{\hat{t}_S}. \tag{17}$$

For parallel or distributed simulations, we suggest using (17) when evaluating multiple models. Alternatively, for a single model, the traditional speedup metric can be applied:

$$s_S = \frac{t_R(m_z^T)}{t_S(m_z^T)}. \tag{18}$$

In these equations, $\hat{t}_R$ (or $t_R(m_z^T)$) and $\hat{t}_S$ (or $t_S(m_z^T)$) represent the sequential and parallel performance metrics, respectively.

Together, these equations provide a clear and consistent framework for evaluating simulation performance across both sequential and parallel computing environments.

4 EXPERIMENTAL METHODOLOGY

We apply the methodology of Section 3 to the DEVStone benchmark and various DES simulation engines. This involves aligning DEVStone with the general notation introduced from Eqs. (12) to (18), detailing the simulators, and describing the sequential and parallel setups used in the experiments. The goal is not to compare a plethora of existing DES simulators, but to present a unified methodology to analyze the performance of the simulator.

4.1 Nomenclature Correspondence

The benchmark models used in this work are DEVStone models, with topologies defined by model types $T \in \mathbf{T} = \{\text{LI}, \text{HI}, \text{HO}, \text{HOmod}\}$, and sizes specified by widths and depths as $z \in Z_T = \{w, d\}_T$.

The aDEVS simulator [8], known for its high sequential performance in previous DEVStone benchmarks [18], is selected as the reference DES simulator R in these equations. This choice is based on its proven efficiency in sequential execution, its stability across different configurations, and its frequent use in DES simulator performance evaluations, making it a solid and reliable benchmark for comparison.

4.2 Simulation Engines Under Study

To evaluate the proposed metrics ($\hat{t}_S^T$, $\hat{s}_S^T$, $\hat{t}_S$, $\hat{s}_S$, and s_S), three simulation engines are used: aDEVS as a reference, xDEVS as a DEVS simulator, and SystemC as a non-DEVS simulator. The SystemC DEVStone models are implemented based on the method in [20], ensuring an equivalent structure and behavior without replicating the DEVS protocol.

Table 1 summarizes the characteristics of the engines and environments, including versions, programming languages, and compilers or interpreters. In particular, xDEVS is implemented in three programming languages (C++, Java and Python), enabling an analysis of language effects under the same interface.

Table 1: Engine versions and environments used for the DEVStone simulations.

Engine	Version	Programming Language	Interpreter/Compiler
aDEVS	3.3	C++17	g++ 7.5 (-o3)
xDEVS/C++	1.0	C++11	g++ 7.5 (-o3)
xDEVS/JAVA	2.0	JAVA	OpenJDK 11.0.7
xDEVS/Python	2.0	Python3	CPython 3.6.9
SystemC	2.3.4	C++17	g++ 7.5 (-o3)

4.3 Sequential and Parallel Environments

For sequential simulations, a wide range of DEVStone models with varying depth and width was tested, specifically 100 to 1200 with steps of 50 for LI, HI, and HO topologies. HOmod models were excluded due to their high memory demands, which hinder fair comparisons between simulation platforms. All sequential simulations did not use extra workload, i.e. $\Delta = 0$ inside transition functions, as transition times do not significantly impact sequential performance but only extend execution times. Experiments were conducted on an n1-standard-1 Google Cloud Platform virtual machine (Intel Xeon CPU @ 2.30GHz, 3.75 GiB RAM). xDEVS/Python encountered memory heap issues with the most complex models.

Following state-of-the-art recommendations for parallel simulations [21], the HO model set ($w = d = 15$) was selected due to its balanced CPU and memory usage, and transition computation times were set to high values ($\Delta_{\text{int}} = \Delta_{\text{ext}} = 2$), which evaluates the simulator's ability to handle complex, computationally intensive models, and emphasizing workload distribution rather than synchronization or communication costs. Experiments were conducted using the xDEVS/Java engine on a high-performance computing cluster. Our configurations used 2, 4, 8, and 16 cores from Intel Xeon Gold 6230 CPUs @ 2.10GHz, with access to 375 GiB of shared memory. A single-task per CPU-core allocation strategy was employed to control hyperthreading and thread usage, optimizing parallel performance evaluation.

5 RESULTS

5.1 Sequential Analysis

Figure 2 compares the total simulation times of aDEVS, xDEVS, and SystemC for different DEVStone implementations (LI, HI and HO). Each row represents a simulation engine, and each column corresponds to a model type. The plots show simulation times based on DEVStone width and depth, with the color intensity indicating higher times. The total execution time includes model loading, initialization, and simulation.

Figure 2 highlights key performance insights. aDEVS emerges as the fastest simulator overall, while xDEVS/C++ achieves comparable times and better peak speedup (1.27 for $\text{HO}_{w=d=1200}$) due to efficient memory management using smart pointers.

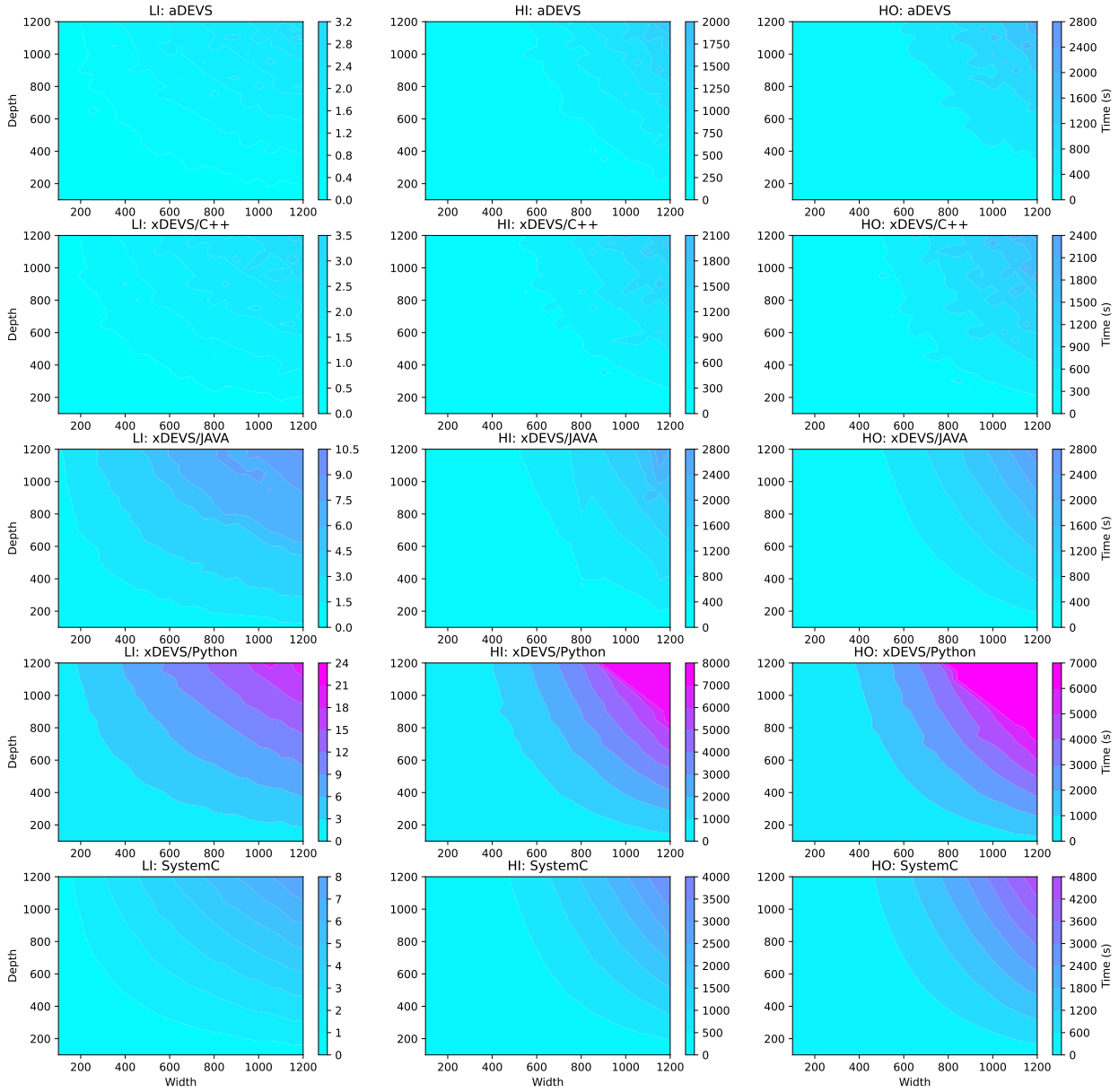


Figure 2: DEVStone simulation times for aDEVS and the different DES simulator implementations.

Table 2 details total simulation times for specific configurations. xDEVS/Java is slower than aDEVS and xDEVS/C++ due to the overhead of the Java Virtual Machine, but it benefits from parallelization capabilities, already included in its API. SystemC starts with better performance than xDEVS/Java but slows down for complex models (e.g. $HI_{(w,d)>300}$). xDEVS/Python is the slowest engine due to its interpreted nature, though it provides advantages like Python's extensive libraries for data analysis.

Table 3 presents the $\hat{t}_S^T$ metric for each DEVStone model class and simulator, along with the general $\hat{t}_S$ metric. The results align with earlier findings but provide a more concise comparison. aDEVS is the fastest simulator, followed by xDEVS/C++, xDEVS/Java, SystemC, and xDEVS/Python.

Table 2: Simulation times (in seconds) for balanced DEVStone models (Size=Width=Depth).

Model	Size	aDEVS	xDEVS/C++	xDEVS/JAVA	xDEVS/Python	SystemC
LI	400	0.27	0.35	1.60	2.16	0.82
	600	0.63	0.79	3.17	4.80	1.97
	800	1.12	1.83	4.84	8.49	3.56
	1000	1.74	2.21	7.57	13.08	5.58
	1200	2.55	3.27	9.24	18.97	7.89
HI	400	36.34	49.37	72.09	332.21	99.14
	600	128.06	258.26	249.30	1088.55	391.53
	800	338.94	431.46	795.57	2649.91	986.74
	1000	745.52	826.01	1272.91	5312.32	2038.17
	1200	1462.11	1463.87	2340.96	7879.57	3608.29
HO	400	60.43	62.12	81.12	351.48	126.8
	600	185.54	213.82	274.17	1197.56	507.05
	800	392.61	558.46	712.02	2864.07	1293.23
	1000	1252.72	1030.42	1401.38	6554.49	2636.28
	1200	2553.18	2004.62	2613.55	6554.49	4679.24

Table 3: $\hat{t}$ metric values.

Model	aDEVS	xDEVS/C++	xDEVS/JAVA	xDEVS/Python	SystemC
$\hat{t}_S^{\mathrm{LI}}$	2.13	2.41	5.61	8.66	4.26
$\hat{t}_S^{\mathrm{HI}}$	445.80	504.15	739.32	3087.57	1126.78
$\hat{t}_S^{\mathrm{HO}}$	525.85	623.01	808.34	3227.87	1475.46
$\hat{t}_S$	79.33	91.14	149.67	441.93	192.04

Table 4 presents the $\hat{s}$ metric, which provides performance comparisons relative to the aDEVS simulator. xDEVS/C++ achieves the best score with 0.87 RPUs, followed by xDEVS/Java (0.53), SystemC (0.41), and xDEVS/Python (0.18). As a relative metric, $\hat{s}$ is independent of the M&S formalism, allowing comparisons between different DEVS and non-DEVS simulation engines using the same model set and computing architecture.

5.2 Parallel analysis

In the parallel case, multiple HO runs were not performed, as the models exhibit uniform behavior, and repeated runs did not show a significant variation in total time. As described in Section 4.3, the HO model with $w = d = 15$ and $\Delta = 2$ was used. The standard speed-up metric in (18) with $R =$ aDEVS and $S =$ xDEVS/Java was calculated without a geometric mean, as only a single model was evaluated for different hardware resources.

Figure 3 illustrates the speedup achieved by the xDEVS/Java simulation engine using 2, 4, 8, and 16 CPU cores. Furthermore, the effect of thread communication and saturation was analyzed by varying the number of threads from 2 to 20.

The speedup achieved by xDEVS/Java is nearly proportional to the number of cores, provided there are sufficient threads. However, slight degradation occurs as threads increase due to communication overhead.

Table 4: $\hat{s}$ metric values (RPUs).

Model	xDEVS/C++	xDEVS/JAVA	xDEVS/Python	SystemC
$\hat{s}_S^{\mathrm{LI}}$	0.88	0.38	0.25	0.50
$\hat{s}_S^{\mathrm{HI}}$	0.88	0.60	0.14	0.40
$\hat{s}_S^{\mathrm{HO}}$	0.84	0.65	0.16	0.36
$\hat{s}_S$	0.87	0.53	0.18	0.41

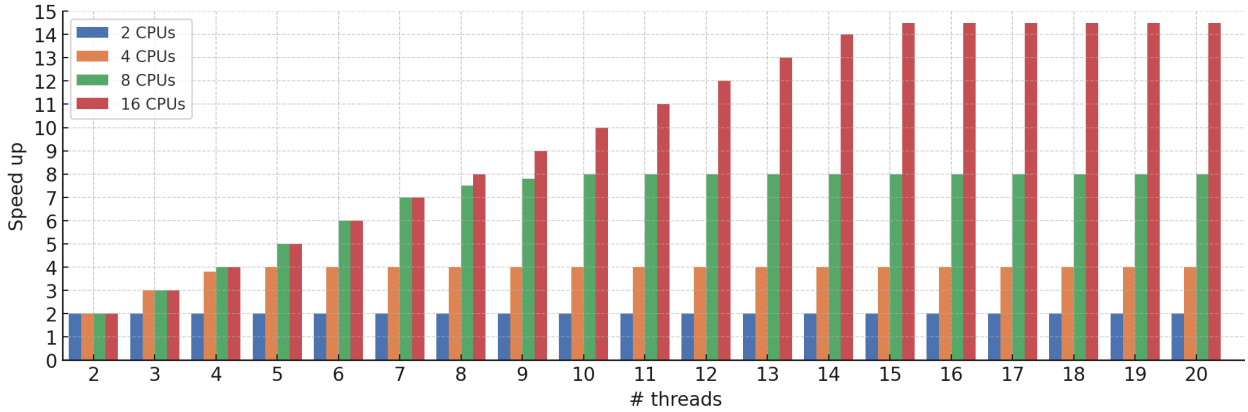


Figure 3: Speedup of the HO model with $d = w = 15$ in the XDEVS/Java simulator in relation to aDEVS.

For more than 4 cores, the speedup grows sub-linearly, influenced by CPU context switching. Maximum speedups of 2.00, 3.98, 7.86, and 15.02 RPUs were obtained for 2, 4, 8, and 16 cores, respectively. This shows that effective parallelization can significantly improve performance, with xDEVS/Java achieving a maximum speedup of 15.02 RPUs on 16 cores, outperforming even the fastest sequential simulator.

6 CONCLUSION AND FUTURE WORK

Evaluating the performance of DES engines is often challenging due to the lack of standardized benchmarks and metrics. This paper introduces DEVStone as a comprehensive benchmark suite for assessing DES simulators. DEVStone follows the DEVS formalism, which is widely recognized for its ability to model discrete systems effectively. By focusing on total simulation time as a universal metric, DEVStone provides a consistent framework for comparing different simulation engines.

Our study demonstrates that the geometric mean is an effective method for aggregating simulation times in sequential experiments, offering a platform-independent measure of performance. For parallel and distributed simulations, traditional speedup metrics remain appropriate. These approaches ensure that performance evaluations are both reproducible and representative, allowing researchers to make informed decisions when selecting simulation tools.

Despite its strengths, DEVStone currently lacks explicit mechanisms for controlling communication patterns among simulation objects, which could limit its applicability in certain scenarios. Addressing this limitation is a key area for future development. We also plan to enhance DEVStone by integrating modern transition algorithms that provide more accurate assessments of CPU and memory usage. This will replace the current Dhrystone-based approach, which primarily serves to occupy CPU resources. Additionally, we aim to establish a public repository where the modeling and simulation community can access and contribute open data from performance experiments. This initiative will foster collaboration and standardization across the

field. Further comparative studies will explore alternative weighting methods and metrics, offering deeper insights into the strengths and limitations of our proposed approach.

Another aspect for future research is the propagation of events within simulations. Although the HO model has the same number of state transitions as other models, its order of complexity is higher due to the increased propagation of events. Investigating how different event propagation patterns affect simulation performance, as well as developing benchmarks that specifically account for these effects, could provide valuable insights into optimizing simulation efficiency. Future work will explore the impact of event propagation on execution time and resource utilization, potentially leading to more refined benchmarking methodologies.

ACKNOWLEDGMENTS

This work has been supported by the Research Projects SMART-BLOOMS (TED2021-130123B-I00) by MCIN/AEI/10.13039/501100011033 and the European Union NextGenerationEU/PRTR, and INSERTION (PID2021-127648OB-C33) by the Knowledge Generation program of the Spanish Ministry of Science and Innovation.

REFERENCES

[1] A. Tolk, "Simulation and modeling as the essence of computational science," in *Proceeding of the 2019 Summer Simulation Conference (SummerSim'2018)*, 2018, pp. 1–12.

[2] R. Siegfried, *Modeling and simulation of complex systems: A framework for efficient agent-based modeling and simulation*. Springer, 2014.

[3] D. Muravev, H. Hu, A. Rakhmangulov, and P. Mishkurov, "Multi-agent optimization of the intermodal terminal main parameters by using anylogic simulation platform: Case study on the ningbozhoushan port," *International Journal of Information Management*, vol. 57, p. 102133, 2021.

[4] T. Wang, A. Guinet, A. Belaidi, and B. Besombes, "Modelling and simulation of emergency services with aris and arena. case study: the emergency department of saint joseph and saint luc hospital," *Production Planning and Control*, vol. 20, no. 6, pp. 484–495, 2009.

[5] C. Seo, B. P. Zeigler, R. Coop, and D. Kim, "DEVS modeling and simulation methodology with MS4 Me software tool," in *Proceedings of the Spring Simulation Multiconference (SpringSim'2013)*, 2013, p. 33.

[6] J. Siderska *et al.*, "Application of tecnomatix plant simulation for modeling production and logistics processes," *Business, Management and Education*, vol. 14, no. 1, pp. 64–73, 2016.

[7] M. I. Clune, P. J. Mosterman, and C. G. Cassandras, "Discrete event and hybrid system simulation with simevents," in *Proceedings of the 8th International Workshop on Discrete Event Systems*. Citeseer, 2006, pp. 386–387.

[8] J. Nutaro, "adevs: a discrete event system simulator," https://web.ornl.gov/ nutarojj/adevs/, 2022.

[9] C. Ptolemaeus, *System design, modeling, and simulation: using Ptolemy II*. Ptolemy. org Berkeley, 2014, vol. 1.

[10] L. Belloli, D. Vicino, C. Ruiz-Martin, and G. Wainer, "Building DEVS models with the Cadmium tool," in *2019 Winter Simulation Conference (WSC'19)*, 2019, pp. 45–59.

[11] W. Müller, W. Rosenstiel, and J. Ruf, *SystemC: methodologies and applications*. Springer Science & Business, 2007.

[12] J. L. Risco-Martín, S. Mittal, K. Henares, R. Cardenas, and P. Arroba, "xDEVS: a toolkit for interoperable modeling and simulation of formal discrete event systems," *Software: Practice and Experience*, pp. 1–42, 2023.

[13] R. M. Fujimoto, "Exploiting temporal uncertainty in parallel and distributed simulations," in *Proceedings of the Thirteenth Workshop on Parallel and Distributed Simulation (PADS'1999)*. IEEE, 1999, pp. 46–53.

[14] E. Park, S. Eidenbenz, N. Santhi, G. Chapuis, and B. Settlemyer, "Parameterized benchmarking of parallel discrete event simulation systems: communication, computation, and memory," in *2015 Winter Simulation Conference (WSC'2015)*. IEEE, 2015, pp. 2836–2847.

[15] V. Bonnet, "Benchmarking parallel discrete event simulations," Master's thesis, Utrecht University, 2017. [Online]. Available: https://studenttheses.uu.nl/handle/20.500.12932/26960

[16] G. Wainer, E. Glinsky, and M. Gutierrez-Alcaraz, "Studying performance of devs modeling and simulation environments using the devstone benchmark," *Simulation*, vol. 87, no. 7, pp. 555–580, 2011.

[17] H. Vangheluwe, "DEVS as a common denominator for multi-formalism hybrid systems modelling," in *Proceedings of the IEEE International Symposium on Computer-Aided Control System Design*. Anchorage, AK, USA: IEEE, 2000, pp. 129–134. [Online]. Available: http://ieeexplore.ieee.org/document/900199/

[18] R. Cárdenas, K. Henares, P. Arroba, J. L. Risco-Martín, and G. A. Wainer, "The DEVStone Metric: Performance Analysis of DEVS Simulation Engines," *ACM Transactions on Modeling and Computer Simulation*, 2022.

[19] J. L. Risco-Martín, S. Mittal, J. C. Fabero Jiménez, M. Zapater, and R. Hermida Correa, "Reconsidering the performance of DEVS modeling and simulation environments using the DEVStone benchmark," *Simulation*, vol. 93, no. 6, pp. 459–476, 2017.

[20] F. Madlener, H. G. Molter, and S. A. Huss, "SC-DEVS: an efficient SystemC extension for the DEVS model of computation," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 1518–1523.

[21] J. L. Risco-Martín, K. Henares, S. Mittal, L. F. Almendras, and K. Olcoz, "A unified cloud-enabled discrete event parallel and distributed simulation architecture," *Simulation Modelling Practice and Theory*, vol. 118, p. 102539, 2022.

AUTHOR BIOGRAPHIES

BEATRIZ HERGUEDAS is a Ph.D. student in Computer Science at Universidad Complutense de Madrid (UCM), Spain. Her research interests include computer-aided design, modeling, simulation, and optimization of complex systems. She can be reached at behergue@ucm.es.

ROMÁN CÁRDENAS is an Assistant Professor at Universidad Politécnica de Madrid (UPM), Spain. He obtained a Ph.D. in Electronic Systems Engineering in Cotutelle modality at UPM and Carleton University. His research interests include M&S in the IoT domain. His email address is r.cardenas@upm.es.

PATRICIA ARROBA is an Associate Professor at Universidad Politécnica de Madrid (UPM), Spain, where she received her Ph.D. in Telecommunication Engineering. Her research interests include energy and thermal-aware M&S&O for Cloud and Edge infrastructures. She can be reached at p.arroba@upm.es.

EVA BESADA-PORTAS is an Associate Professor of Systems Engineering and Automation at UCM. She also holds a PhD in Computer Systems from UCM. Her research interests include uncertainty modeling and simulation, optimal control and planning of unmanned vehicles. Her email address is ebesada@ucm.es.

JOSÉ L. RISCO-MARTÍN received his Ph.D. from Universidad Complutense de Madrid (UCM), Spain, where he currently is a Full Professor. His research interests include computer-aided design and modeling, simulation, and optimization of complex systems. His email address is jlrisco@ucm.es.