DEVSML Studio: A Framework for Integrating Domain-Specific Languages for Discrete and Continuous Hybrid Systems into DEVS-Based M&S Environment

Saurabh Mittal
Dunip Technologies, LLC
Littleton, CO, USA
smittal@duniptech.com

José L. Risco-Martín
Complutense University of Madrid
Madrid, Spain
jlrisco@ucm.es

ABSTRACT

The Discrete EVent System (DEVS) specification has been implemented in various platforms and languages over the years. However, each implementation has been tightly coupled with the underlying syntactical language. The DEVS Modeling Language (DEVSML) is based on meta-modeling concepts that provide a Domain-Specific-Language (DSL) for DEVS model description. In this paper, we introduce DEVSML Eclipse Studio that implement DEVSML execution with two DEVS engines: DEVSJAVA and xDEVS. We demonstrate the features of DEVSML Studio with a moderately complex example of a spectroscopy system involving digital shapers. We emphasize the importance of metamodeling concepts and their implementation in the DEVSML Studio that enables model-driven engineering practices for bringing other DSLs to a DEVS-based Modeling and Simulation (M&S) environment. We also establish the robustness and correctness of the xDEVS simulation engine, integrated within DEVSML Studio for a hybrid discrete and continuous system such as a digital shaper.

Author Keywords

Metamodeling; Eclipse Plugin Development Environment (PDE); DEVSML Studio; xDEVS; Digital Shapers; DEVS

ACM Classification Keywords

C.3 SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS: *Real-time and embedded systems*; I.6.1 SIMULATION AND MODELING: Simulation Theory; J.6 COMPUTER-AIDED ENGINEERING: *Computer-aided design (CAD)*

INTRODUCTION

DEVS formalism has been in existence for over four decades and has been implemented in major Object-oriented languages, e.g. Lisp, Scheme, C++, Java, Python, SmallTalk, etc. We have ourselves tried the next generation of Java Virtual Machine (JVM)-based languages such a Groovy, Scala and Xtend. There exist many DEVS development environments. However, model creation is largely in the implementation language of the simulation engine implementation. Model-Driven Engineering (MDE) has started to make way into DEVS Integrated Development Environments (IDEs) [22] and various DEVS metamodels have started to appear in the community. However, the abstraction levels at the DEVS Modeling API have not been clearly defined or standardized for that matter. Efforts are underway for a standardized DEVS modeling and simulation API for interoperability at both the modeling and simulation layers.

In this article, we explore the state of the art in DEVS M&S IDEs, DEVS metamodeling and abstract DEVS Modeling Language (DEVSML). We will introduce an Eclipse DEVSML Studio that allows graphical development of atomic DEVS state machine and coupled digraphs. Currently, DEVSML Studio is able to execute DEVS using two simulation engines: DEVSJAVA [5] and xDEVS [24]. We will highlight the capabilities of the DEVSML Studio through a moderately complex example of a spectroscopy system involving digital shapers. We will demonstrate that the DEVSML Studio is capable of handling complex models and illustrate how the novel xDEVS engine provides excellent abstraction mechanisms and robust performance.

The reminder of this paper is organized as follows. Firstly, we shows a brief survey on existing DEVS tools. Secondly, we present the architecture of DEVSML Studio. Next, we present an example with moderate complexity that can be modeled using the DEVSML Studio. Finally, we show our conclusions.

SURVEY OF EXISTING DEVS TOOLS

In the last decade, many DEVS M&S engines have come into existence. Almost all of them offer a programmer-friendly Application Programming Interface (API) to define new models using a high level language and with an exception of one or two, most are bound to an implementation language. Alternatively, none is based on a DEVS metamodel. Further, only a few of them provide a user-friendly Graphical User Interface (GUI) for model specification. In the following, we describe some of the most referenced DEVS M&S simulation frameworks:

DEVSJAVA

DEVSJAVA has been developed by Bernard P. Zeigler (University of Arizona, U.S.A.) and Hessam Sarjoughian (Arizona State University, U.S.A.) [5]. It is written in Java and

SummerSim-SCSC 2016 July 24-27 Montreal, Quebec, Canada

supports virtual time, real time, and sequential and parallel execution. The definition of new models is performed through an API. Several M&S tools have been defined around DEVSJAVA (GUIs for results visualization, GUIs for models definition, etc.), as DEVSJAVA is one of the primary DEVS M&S reference simulators in the community.

DEVS-Suite and COSMOS
DEVS-Suite is a simulator built based on the Parallel DEVS formalism, design of experiment concepts, and simulation visualization techniques consisting of displaying static structure of models, animation of models, and run-time viewing of time-based trajectories [4]. CoSMoS (Component-Based System Modeling and Simulation) is a framework aimed at integrated visual model development, model configuration and automatic simulation data collection [3]. The CoSMoS environment supports component-based modeling with direct support for DEVS formalism and XML Schema. DEVS-Suite's core is largely DEVSJAVA. It is bundled within the CoSMoS distribution and thus enables both modeling and simulation of Parallel DEVS models.

CD++
CD++ has been developed by Gabriel Wainer and his students (Carleton University, Canada; Universidad de Buenos Aires, Argentina). Written in C++, it allows the definition of DEVS and Cell-DEVS models graphically. These models are also defined using an API. CD++ supports virtual and real time, as well as sequential, parallel and distributed simulations [2].

xDEVS
xDEVS has been recently implemented by José L. Risco-Martín and Saurabh Mittal [24]. Developed in Java, supports both virtual time and real time simulations (in sequential, parallel or distributed environments), as well as flattened simulations. It also allows hardware/software co-simulations and provides a simulation profiler to measure performance. xDEVS is designed using Object-oriented paradigm and is released under the GNU Public License (GPL). This facilitates the rapid development of new components and extensions, and wide adoption of the core engine. xDEVS provides the user with a set of base classes that can be used to develop new DEVS models, or to develop new DEVS simulation engines. xDEVS is based on the fundamental separation of model and the underlying corresponding simulator [27] and rightly so, provides, the modeling Application Program Interface (API) and the simulation API.

PyDEVS
PythonDEVS (a.k.a. PyDEVS) implements both Classic and Parallel DEVS in the Python language, with a matching simulator [25]. Models are defined through the provided API, allowing the execution of virtual time or real time simulations. The latest release of PyDEVS is focused on improving the performance, mainly because Python is an interpreted language. To this end, several schedulers have been defined, obtaining good performance metrics.

aDEVS

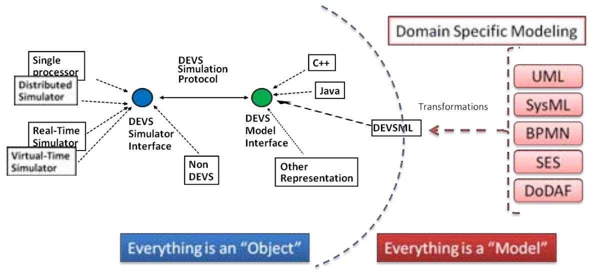


Figure 1. Integrating theory of modeling and simulation framework with MDE [22].

aDEVS (a Discrete EVent System simulator) is a C++ library for constructing discrete event simulations based on the Parallel DEVS and Dynamic DEVS (dynDEVS) formalisms [1]. Developed by Jim Nutaro, it allows the implementation of both sequential and parallel simulations using the provided C++ API. This framework by far, displays the best performance.

JAMES-II
Developed at the University of Rostock, the Java-based Multipurpose Environment for Simulation II (JAMES II) provides support for many different formalisms, including various variants of DEVS formalisms. Besides an API to define models, this framework also provides a GUI to configure experiments and see simulation results. This simulation engine supports sequential and parallel execution [9].

DEVSim++
Developed by Tag Gon Kim and his group at Korea Advanced Institute of Technology (KAIST) [16], this is a C++ based engine and used extensively for large simulations focusing on wargaming and simulation interoperability.

In addition to the above DEVS implementations used widely, there are others with selective adoption such as GALATEA [14] for Multi-Agent Systems (MAS), SimStudio [26], PowerDEVS [12] for hybrid systems, MS4Me based on DEVSJAVA [10] and last but not the least, Virtual Laboratory Environment (VLE) [13], based on C++. VLE is a multiparadigm environment based on several DEVS extensions. Providing a graphical atomic model representation has been a challenge in all the existing DEVS engines and simulation environments, mostly attributed to the lack of a DEVS metamodel and standardized atomic notation. Consequently, while depicting coupled model is easy, depicting an atomic DEVS state machine has proven to be hard and largely unattended. It is worth stating that a DEVS State machine is more expressive than a UML state machine. Consequently, more notations are needed in UML to account for DEVS specifications. This paper provides a way forward to visualize both the atomic and coupled models.

DEVSML ECLIPSE DEVELOPMENT STUDIO
DEVSML Studio is available for use with any JVM-based languages, such as Groovy, Scala, Xtend, etc. A demonstrative example is available online [17], where various languages can be used at the Modeling layer leveraging the

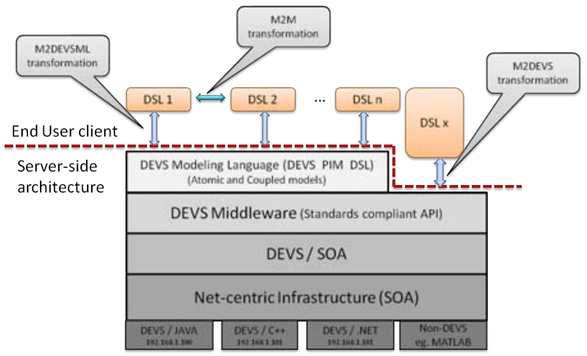


Figure 2. DEVSML Stack showing DEVS modeling language and various transformations [19].

modeling API interface (see Figure 1). The xDEVS engine is made available as the platform-specific-model (PSM) of a more abstract DEVS Modeling Language (DEVSML), which is based on XML-Based Finite Deterministic DEVS [20] and language introduced in [19] and later developed in [22][21]. The DEVSML Stack shown in Figure 2 describes the layered architecture of platform-independent nature of DEVSML [19][18]. The idea of including other domain-specific languages (DSLs) and the following transformations at the top layer of the stack brings in model-driven engineering (MDE) concepts with the DEVS M&S framework. Three transformations are defined that allows various DSLs to be transformed into DEVSML or directly to DEVS:

1. Model-to-Model (M2M)
2. Model-to-DEVSML (M2DEVSML)
3. Model-to-DEVS (M2DEVS)

DEVSML and DEVS Unified Process (DUNIP) are focused towards interoperability at the application level, specifically, at the modeling level and hiding the simulator engine as a whole, making it transparent [23]. Our vision and solution development is along the lines of Model-as-a-Service (MaaS), Simulation-as-a-Service (SimaaS), DEVS-as-a-Service (DevaaS) and ultimately, System-as-a-Service (SysaaS). We would like the user or designer to code the behavior in any of the programming languages, ideally a DSL of his choice and let the DEVSML stack develop the transformations (Figure 1). The DEVS/SOA architecture is responsible for taking a DSL or a coupled DEVSML model with the associated transformations and delivering us with an executable model that can be simulated on any parallel-distributed netcentric DEVS platform. The realization of netcentric DEVS has the following pieces:

1. DEVSML Stack: the central concept.
2. Distributed simulation using SOA.
3. Netcentric DEVS VM (both client and server).
4. Design, development and deployment of netcentric systems with DEVS.

The user can integrate his model from models stored in any Web/Cloud repository, whether it contained public models of legacy systems or proprietary standardized models. This will prove beneficial for both the industry as well as to the user, thereby truly realizing the model-driven paradigm. MS4Me [10] and CD++ [2] are already having such repositories.

We introduce a DEVSML Modeling Studio in this article that demonstrates MDE principles and core ideas in the DEVSML paradigm.

Features

The DEVSML Studio provides the following features:

1. It is based on Eclipse PDE with Xtext [8] EBNF grammar underneath as DEVSML metamodel.
2. It provides textual templates for Atomic and Coupled DEVSML models, rich with code-completion and DEVS model validation.
3. It provides a visualization plugin for rapid visual inspection of both the atomic and coupled DEVS. The visualization plugin is based on open-source PlantUML plugin [11].
4. It can be configured with different DEVS-engines using DEVSML configuration settings. The default is xDEVS M&S engine [24]. The other available engine is DEVS-JAVA [5].
5. It can be configured with various platform-specific implementations. Currently only JVM-based languages are supported and efforts are underway to generate C++ (aDEVS) and Python (PyDEVS).
6. It provides compiled JAVA code for ready execution of DEVSML.
7. It integrates EclEmma Code Coverage plugin [7] for JVM executable platform-specific code.
8. It provides explicit port-interfaces for rapid prototyping to message-based netcentric systems using Oracle JMS, Apache Camel, IBM Websphere MQ and Event-driven Architectures using TIBCO, Esper, etc.
9. Code-snippets are provided as String and when a model runtime is configured for a DEVSML project, the platform-specific model shows errors in the generated platform-specific code.
10. It shows the hierarchical structure of a DEVS file in the Eclipse Outline View.

Architecture

The architecture of DEVSML Studio is based on metamodeling concepts and is shown in Figure 3. DEVSML Studio inherits from the Eclipse Workbench Plugin architecture and integrates various views (PlantUML, EclEmma) into a DEVSML Perspective. The execution of DEVSML Studio will be demonstrated through a moderately complex example in the sections ahead. The Studio is available for download at [6]. The xDEVS M&S Engine is available both as .jar and as an Eclipse plugin. The DEVSML Studio is available as an installable Eclipse feature.

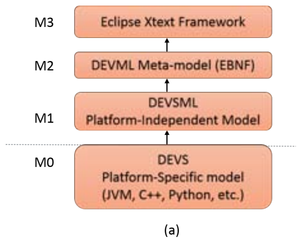
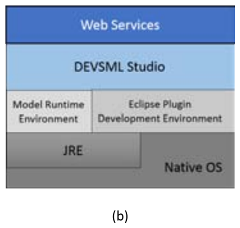

M3
Eclipse Xtext Framework
M2
DEVML Meta-model (EBNF)
M1
DEVSML Platform-Independent Model
M0
DEVS Platform-Specific model (JVM, C++, Python, etc.)
(a)

Web Services
DEVSML Studio
Model Runtime Environment
Eclipse Plugin Development Environment
JRE
Native OS
(b)

Figure 3. (a) Metamodeling layers and (b) DEVSML Studio software architecture.

CASE STUDY: TRAPEZOIDAL PULSE SHAPER

The integration with other discrete systems like Business Process Modeling Notation (BPMN), Unified Modeling Language (UML), Statecharts, System Entity Structure (SES), etc., has already been demonstrated and reported in the book by Mittal and Risco-Martín [22]. We have selected an example of trapezoidal pulse shapers to demonstrate the integration of discrete and continuous hybrid systems. In this section, we demonstrate the capabilities of DEVSML Studio with an example of moderate complexity: M&S of trapezoidal pulse shapers.

The performance of nuclear spectroscopy systems has been considerably improved by replacing the conventional analogue electronics modules by modern digital systems. The detector-preamplifier configuration of a common spectroscopy system produces a pulse with an initial short rise time followed by a long exponential tail. Using a trapezoidal digital shaper, the exponential signal is transformed into a trapezoid by series of differentiators and integrators, and the pulse energy is measured as the difference between the fat top of the trapezoid and its base. The design of these trapezoidal shapers requires the definition of a set of parameters based on the characteristic of the input signal (timing, noise, etc.). Hence, M&S is essential to analyze these parameters. In the following, we model a common spectroscopy system using DEVSML Studio and xDEVS as the simulation engine, analyzing its behavior.

Model description

A recursive algorithm that converts a digitized exponential pulse $v(n)$ into a symmetrical trapezoidal pulse $s(n)$ is given by equations 1 to 5, borrowed from [15]:

$$\begin{aligned} d^k(n) &= v(n) - v(n-k), && (1)\\ d^{k,l}(n) &= d^k(n) - d^k(n-l), && (2)\\ p(n) &= p(n-1) + m_2 \cdot d^{k,l}(n), n \geq 0, && (3)\\ r(n) &= p(n) + m_1 \cdot d^{k,l}(n), && (4)\\ s(n) &= s(n-1) + r(n), n \geq 0 && (5) \end{aligned}$$

In the equations above, $v(n)$, $p(n)$ and $s(n)$ are equal to zero for $n < 0$. Parameters m_1 and m_2 only depend on the decay time constant of the exponential pulse, τ, and the sampling period, T_{clk}, given by:

$$\frac{m_1}{m_2} = \left(e^{\frac{T_{\text{clk}}}{\tau}} - 1\right)^{-1} \qquad (6)$$

According to [15], the duration of both the rising and falling edge of the trapezoidal shape is defined by $min(k, l)$, whereas the duration of the flat part of the trapezoid is given by $|k - l|$. Parameter m_2 determines the digital gain of the shaper.

Figure 4 shows a block diagram of the digital trapezoidal shaper. $\texttt{DELAY}_{\{1,2\}}$ are the delay pipelines, $\Sigma_{\{1,2,3\}}$ are adders/subtractors, $\texttt{ACC}_{\{1,2\}}$ are accumulators and $\texttt{X}_{\{1,2\}}$ are multipliers. We can also find coupled models like $\texttt{DS}_{\{1,2\}}$, which is a Delay-Subtractor unit or HPD, which is a High-Pass filter Deconvolver. In the following, we describe the implementation of this trapezoidal shaper using MitRis DEVSML Studio.

Model code

In order to deploy a direct mapping between Figure 4 and the corresponding DEVSML model, the following components are designed:

Atomic models:

AdderSubtractor An adder/subtractor combinational model, to implement $\Sigma_{\{1,2,3\}}$.

Clock It is the digital clock system. It sends a clock square signal to all the sequential components. All the sequential components react to a rising clock edge.

Constant A combinational atomic model designed to send a given value (like m_1 or m_2 in Figure 4) just at the beginning of the simulation.

IdealExpInput This class implements the sequential exponential input as a discrete function where each value is triggered by a clock signal.

Multiplier A multiplier combinational model to implement $\texttt{X}_{\{1,2\}}$.

Register This is a register sequential atomic model, needed to implement both accumulators.

ShiftRegister Implements a sequential shift register. This component is used to simulate the two delay pipelines $\texttt{DELAY}_{\{1,2\}}$.

Coupled models

Accumulator A coupled model that contains an AdderSubstractor atomic model and a Register atomic model.

DelaySubtractor (`DS`) The DS coupled model in Figure 4 contains a ShiftRegister atomic model and an AdderSubtractor atomic model.

HighPassDeconvolver (`HPD`) The HPD coupled model in Figure 4. This model includes two Constant, two Multiplier, one Accumulator and one AdderSubtractor atomic models.

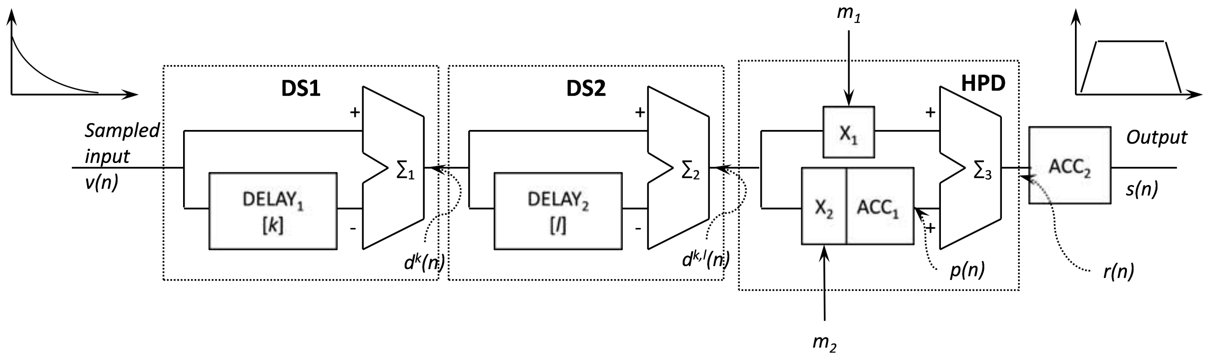


Figure 4. Diagram of the digital trapezoidal shaper. The elements are: $\mathtt{DELAY}_n$ - a delay pipeline, Σ_n an adder/subtracter, $\mathtt{ACC}_n$ an accumulator, $\mathtt{X}_n$ a multiplier. $\mathtt{DS}_n$ is a delay-subtractor unit, `HPD` is a high-pass filter deconvolver.

Trapezoidal This is the trapezoidal digital shaper, including $\mathtt{DS}_1$, $\mathtt{DS}_2$, `HPD` and the final Accumulator in Figure 4. [TrapezoidalTest] This is the coupled model that runs the experiment. It consists of Clock, IdealExpInput and Trapezoidal components.

It can be easily seen that it is hierarchical model (TrapezoidalTest) with depth of 4. A graphical representation in DEVSML Editor for some of the atomic models is shown in Figure 5 and coupled model is shown in Figure 6. In the graphical atomic model, external transitions are shown by red, internal transition are shown by green, an external input is shown as a label prefixed by ? on the red arrow and the output is shown as label prefixed by $\wedge$. In the coupled model, each port displays the fullyQualifiedName and its data-type enclosed within $<< \ldots >>$. The external input couplings (EIC) are shown in green, the internal couplings (IC) in blue and the external output couplings are shown in red. The flows are shown by directed arrows. In order to run the simulation, TrapezoidalTest is executed by a specific coordinator.

Model Execution

A set of 100000 synthetic particle impacts (also called events), which represents up to 4 hours of particle detection in a real satellite, is generated using the following parameters:

- $T_{\text{clk}} = 2 \times 10^{-5}$s
- Amplitude of the exponential input is randomly generated in the interval $[70, 74]$
- τ is also randomly generated in the rage of $[9, 11]$ clock ticks.
- $\{k, l, m_1, m_2\} = \{8, 64, 19, 2\}$

Figure 7 shows the detection of one of these 100000 events, with amplitude equal to 72 and τ equal to 10 clock ticks. The left plot shows the input event and the right plot shows the trapezoid generated by the digital shaper. As can be seen, the set of parameters selected are fine to detect the range of particle impacts generated.

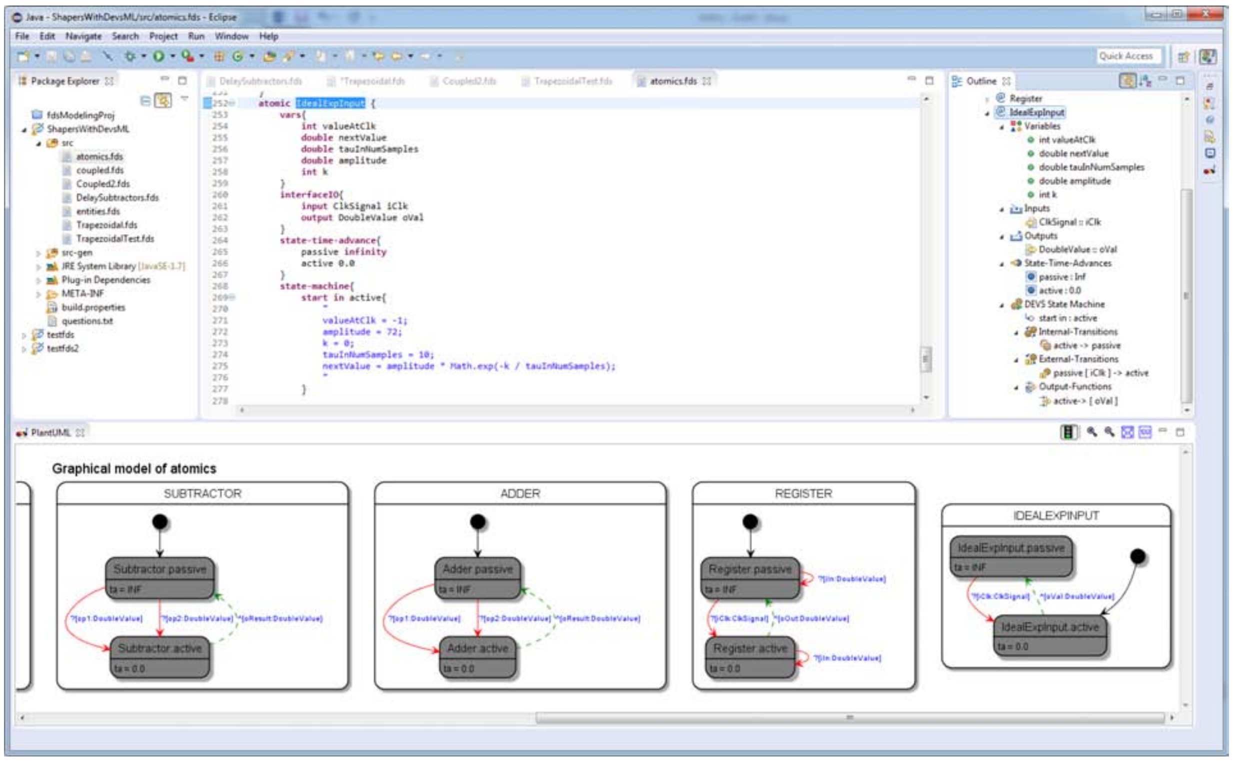
To perform this experiment, we have used an Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz with 16 GB RAM memory, with a GNU/Linux Debian 8 Operating System.

To obtain the characteristic of our DEVS coupled model, we simulate the model using the xDEVS CoordinatorProfile class. As a result, the number of calls to the transition or output functions was equal to 6.66×10^7, whereas the most time consuming function was the Accumulator transition function of the High Pass Deconvolver coupled model with a total of 75.56 s. The atomic model with the lowest performance was the adder of the accumulator 1 ($\mathtt{ACC}_1$ in Figure 4), with an execution time equal to 3.44 s for the transition function (including the external, internal and confluent transitions).

Per the case study, we have, in total, 14 atomic models in TrapezoidalTest. In terms of performance, Figure 8 shows a comparative study of different xDEVS coordinators. A sequential execution takes the maximum amount of time with 151.54 seconds. After flattening the model, the same coordinator improves its performance to 93.97 seconds i.e. a speedup of 1.61. For the parallelized xDEVS simulator, the simulation with the flattened model is run on 8 processors and the parallelized Coordinator execution is varied from using 1 thread to 10 threads. With a single thread, the execution time taken is 96.56 seconds. A little jump in execution time (From 93.97 secs) may be attributed to the thread management in JVM. When the number of threads is increased to 8, the execution time is reduced to 34 seconds, i.e. a total speedup of 4.45 over sequential (without flattened) and 2.76 with flattened. As the threads are increased (from 8 to 10) beyond the number of available cores, more time is spent in managing the thread execution resulting in increased execution time.

CONCLUSIONS

DEVS formalism has been in existence for around 40 years and many implementations of the formalism exist in various programming languages. We surveyed the state of the art in DEVS tools and found that very few use MDE and metamodeling approaches to deliver a workbench for DEVS modeling. In almost all the approaches, the user is forced to program using a computer language and is tied to the execution platform. We explored the DEVS DSL called DEVSML in more detail and implemented the DEVSML metamodel in Eclipse PDE. We have developed DEVSML Studio containing several DEVS M&S engines and an Eclipse editor. We have also presented the xDEVS engine, which is based on latest



```
/
atomic IdealExpInput {
    vars{
        int valueAtClk
        double nextValue
        double tauInNumSamples
        double amplitude
        int k
    }
    interfaceIO{
        input ClkSignal iClk
        output DoubleValue oVal
    }
    state-time-advance{
        passive infinity
        active 0.0
    }
    state-machine{
        start in active{
            "
            valueAtClk = -1;
            amplitude = 72;
            k = 0;
            tauInNumSamples = 10;
            nextValue = amplitude * Math.exp(-k / tauInNumSamples);
            "
        }
```

```
Register
IdealExpInput
    Variables
        int valueAtClk
        double nextValue
        double tauInNumSamples
        double amplitude
        int k
    Inputs
        ClkSignal :: iClk
    Outputs
        DoubleValue :: oVal
    State-Time-Advances
        passive : Inf
        active : 0.0
    DEVS State Machine
        start in : active
        Internal-Transitions
            active -> passive
        External-Transitions
            passive [ iClk ] -> active
        Output-Functions
            active-> [ oVal ]
```

Figure 5. DEVSML Studio showing auto-generated DEVS Atomic visual and hierarchical representation.

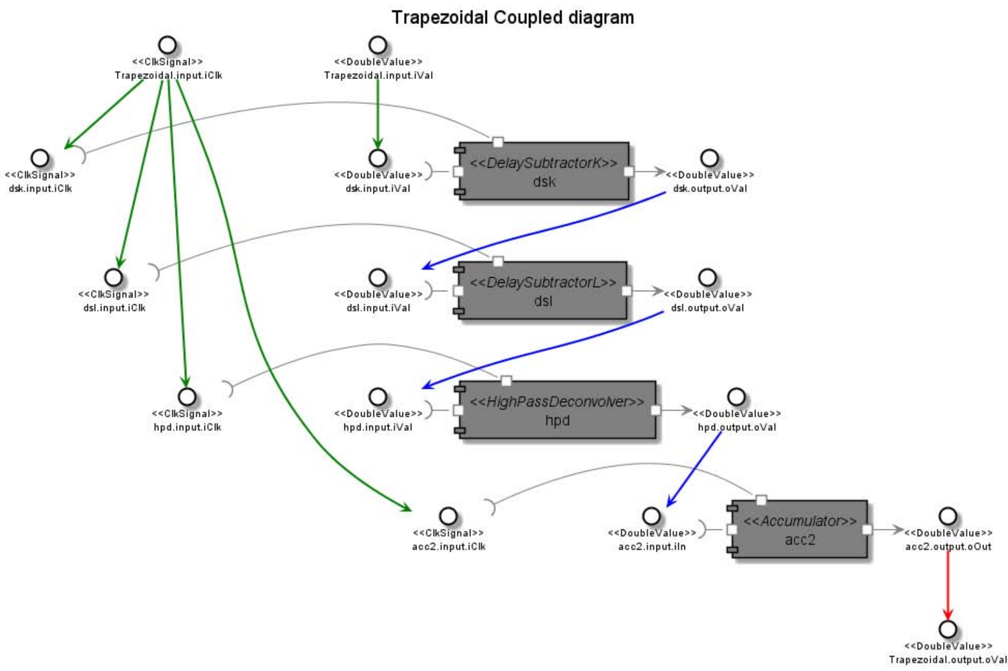


Figure 6. Auto-generated Coupled model representation with port data-types. Green arrows represent External Input Coupling (EIC), blue arrows represent Internal Coupling (IC), and red arrows represent External Output Coupling (EOC).

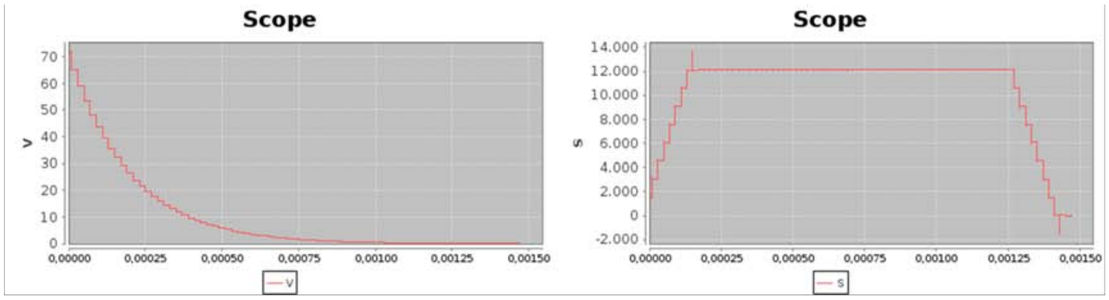


Figure 7. Exponential input (left) and trapezoidal output (right) generated by the xDEVS shaper model for one single event.

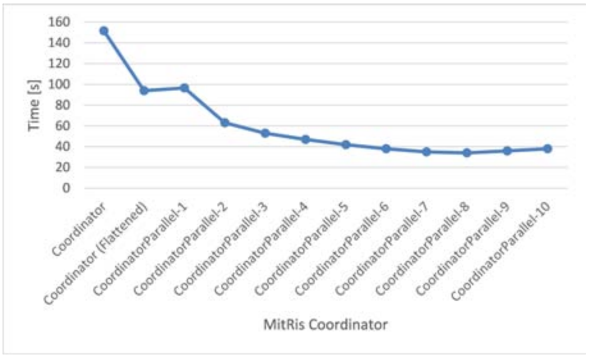


Figure 8. Performance comparison of xDEVS Coordinators.

JVM features and implements advanced executor framework for a parallel and distributed execution on JVM, flattening capability for the coupled model and a profiler for inspecting the performance of any particular model during a simulation execution. DEVSML Studio features advanced model checking, validation, graphical inspection of both atomic and coupled models, and advanced code-generation to multiple DEVS platforms and languages (Java, Python, Groovy, etc.). We also demonstrated the execution of both the engine and the editor using a moderately complex example of digital shapers utilized in nuclear spectroscopy establishing that both the engine and editor are robust enough, and the code-gen using MDE principles delivers valid simulation results for a hybrid discrete and continuous system. Finally, we examined the performance of the xDEVS engine for sequential and parallel execution with multiple thread and achieved a maximum speedup of 4.45 for a parallelized flattened model with 100000 events.

ACKNOWLEDGMENTS

The work on DEVSML Eclipse Studio was developed at Dunip Technologies during the period 2012-2013 and was made available as a part of the book [22] by the authors. The xDEVS simulation engine development and research is supported by the Spanish Ministry of Economy and Competitivity under research grant TIN2014-54806-R.

REFERENCES

1. aDEVS. http://web.ornl.gov/∼1qn/adevs, 2016.
2. CD++. http://cell-devs.sce.carleton.ca, 2016.
3. CoSMoS. http://acims.asu.edu/software/cosmos, 2016.
4. DEVS-Suite. http://devs-suitesim.sourceforge.net, 2016.
5. DEVSJAVA. http://acims.asu.edu/software/devsjava, 2016.
6. DEVSML Studio Update site. http://duniptechnologies.com/jm/downloads.html, 2016.
7. EclEmma Java Code Coverage. http://eclemma.org, 2016.
8. Eclipse Xtext. http://www.xtext.org, 2016.
9. JAMES II. http://wwwmosi.informatik.uni-rostock.de, 2016.
10. MS4Me. http://www.ms4systems.com, 2016.
11. PlantUML. http://plantuml.com, 2016.
12. PowerDEVS. http://powerdevs.sourceforge.net, 2016.
13. VLE: The Virtual Laboratory Environment. http://www.vle-project.org, 2016.
14. Davila, J., and Uzcategui, M. Y. GALATEA: A multi-agent, simulation platform. In *International Conference on Modeling, Simulation and Neural Networks (MSNN 2000)* (2000), 52–67.
15. Jordanov, V. T., Knoll, G. F., Huber, A. C., and Pantazis, J. A. Digital techniques for real-time pulse shaping in radiation measurements. *Nuclear Instruments and Methods in Physics Research A 353* (1994), 261–264.
16. Kim, T. G., Sung, C. H., Hong, S.-Y., Hong, J. H., Choi, C. B., and Kim, J. H. DEVSim++ Toolset for Defense Modeling and Simulation and Interoperation. *The Journal of Defense Modeling & Simulation 8*, 3 (2011), 129–142.
17. Mittal, S. EFP Samples. http://duniptechnologies.com/wp/tag/tutorials, 2016.

18. Mittal, S., and Douglass, S. A. From Domain Specific Languages to DEVS Components: Application to Cognitive M&S. In *Proceedings of the Workshop on Model-driven Approaches for Simulation Engineering - Spring Simulation Multiconference* (2011).

19. Mittal, S., and Douglass, S. A. DEVSML 2.0: The Language and the Stack. In *Symposium on Theory of Modeling and Simulation, Spring Simulation Multiconference* (2012).

20. Mittal, S., Hwang, M. H., and Zeigler, B. P. XFD-DEVS. XML-Based Finite Deterministic DEVS. http://www.duniptechnologies.com/research/xfddevs, 2016.

21. Mittal, S., and Risco-Martín, J. L. Model-driven systems engineering for netcentric system of systems with devs unified process. In *Winter Simulation Conference (WSC 2013)* (Washington, DC, 2013), 1140–1151.

22. Mittal, S., and Risco-Martín, J. L. *Netcentric System of Systems Engineering with DEVS Unified Process*. CRC Press, 2013.

23. Mittal, S., Risco-Martín, J. L., and Zeigler, B. P. DEVSML: automating DEVS execution over SOA towards transparent simulators. In *SpringSim '07: Proceedings of the 2007 spring simulation multiconference*, Society for Computer Simulation International (San Diego, CA, USA, 2007), 287–295.

24. Risco-Martín, J. L., and Mittal, S. xdevs. http://www.duniptechnologies.com, 2016.

25. Tendeloo, Y. V., and Vangheluwe, H. The modular architecture of the python(P)DEVS simulation kernel. In *Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium* (2014), 1–6.

26. Traoré, M. K. SimStudio: a Next Generation Modeling and Simulation Framework. In *International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems* (2010).

27. Zeigler, B. P., Praehofer, H., and Kim, T. G. *Theory of Modeling and Simulation. Integrating Discrete Event and Continuous Complex Dynamic Systems*, 2 ed. Academic Press, 2000.