MitRis 1.0: DEVS Engine and DEVSML Studio

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

Jose Luis Risco Martin
Universidad Complutense de Madrid
Madrid, Spain
jlrisco@ucm.es

ABSTRACT
The DEVS formalism 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: Mitris 1.0 (a) a high performance DEVS engine implemented in Java, and (b) DEVSML Eclipse Studio that implement DEVSML execution with two DEVS engines. We elaborate on the features of MitRis modeling and simulation APIs, DEVS instrumentation and demonstrate the features of DEVSML Studio with a moderately complex example of a spectroscopy system involving digital shapers.

Keywords
Metamodeling, Eclipse Plugin development environment (PDE), MitRis, DEVSML Studio, digital shapers, DEVS.

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) [1] 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 MitRis 1.0 comprising of, first, another JVM-based DEVS M&S engine that brings some new features capitalizing on the latest JVM release, and second, an Eclipse DEVSML Studio that allows graphical development of atomic DEVS state machine and coupled digraphs. 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 MitRis 1.0 engine provides superior abstraction mechanisms and robust performance.

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:

1. DEVSJAVA
DEVSJAVA has been developed by Bernard P. Zeigler (University of Arizona, U.S.A.) and Hessam Sarjoughian (Arizona State University, U.S.A.)[17]. It is written in Java and 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.

2. 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 [18]. CoSMoS (Component-Based System Modeling and Simulation) is a framework aimed at integrated visual model development, model configuration and automatic simulation data collection [19]. 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.

3. 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 [20].

4. PyDEVS

PythonDEVS (a.k.a. PyDEVS) implements both Classic and Parallel DEVS in the Python language, with a matching simulator [21]. 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.

5. ADEVS

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 [22]. 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.

6. 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 [23].

7. DEVSim++

Developed by Tag Gon Kim and his group at Korea Advanced Institute of Technology (KAIST) [24], 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 [25] for Multi-Agent Systems (MAS), SimStudio [26], PowerDEVS [27] for hybrid systems, MS4Me based on DEVSJAVA [28] and last but not the least, Virtual Laboratory Environment (VLE) [29], 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.

MITRIS 1.0 ENGINE

MitRis, 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. MitRis 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. MitRis is based on the fundamental separation of model and the underlying corresponding simulator [16] and rightly so, provides, the modeling *Application Program Interface (API)* and the simulation API.

Modeling API

Figure 1 depicts an UML diagram of the MitRis modeling layer. DEVS models can either be of Atomic or Coupled type and implements the modeling principle through the Modeling API via *DevsAtomic* and *DevsCoupled* interfaces. As Figure 1 shows, atomic and coupled models are, in turn, *Components*. Each component has a set of input and output ports, defined in *InPort* and *OutPort* classes, respectively, that extend a base class, *PortBase*, which implements a generic *Port* interface. According to the DEVS formalism, a model is composed by a set of components with ports and connections between these ports. The *Coupling* class has been implemented to define these connections. Finally, every single piece in a MitRis DEVS model is considered an entity, inherits from *EntityBase* and implements the *Entity* interface.

Simulation API

Figure 2 shows the UML diagram of the MitRis simulation layer. In this case, the simulation principle of MitRis is twofold: first, corresponding to an atomic or a coupled model, there is corresponding DEVS simulator and a DEVS coordinator interface, and second, MitRis completely encapsulates the complexities related to the simulation clock (virtual time or real time) or the parallelization level (sequential, parallel or distributed). To this end, MitRis implements a very minimalist simulator that implements a DEVS simulation interface: *DevsSimulator*. An abstract class called *AbstractSimulator* implements the basic functionality of a DEVS simulator. The *Simulator* class execute the *time advance*, *output* and *transition* functions of a DEVS atomic model. After that, specialized coordinator classes implementing the *DevsCoordinator* interface, perform a DEVS simulation for different requirements, such as fast-mode, centralized, parallelized, real-time, etc.

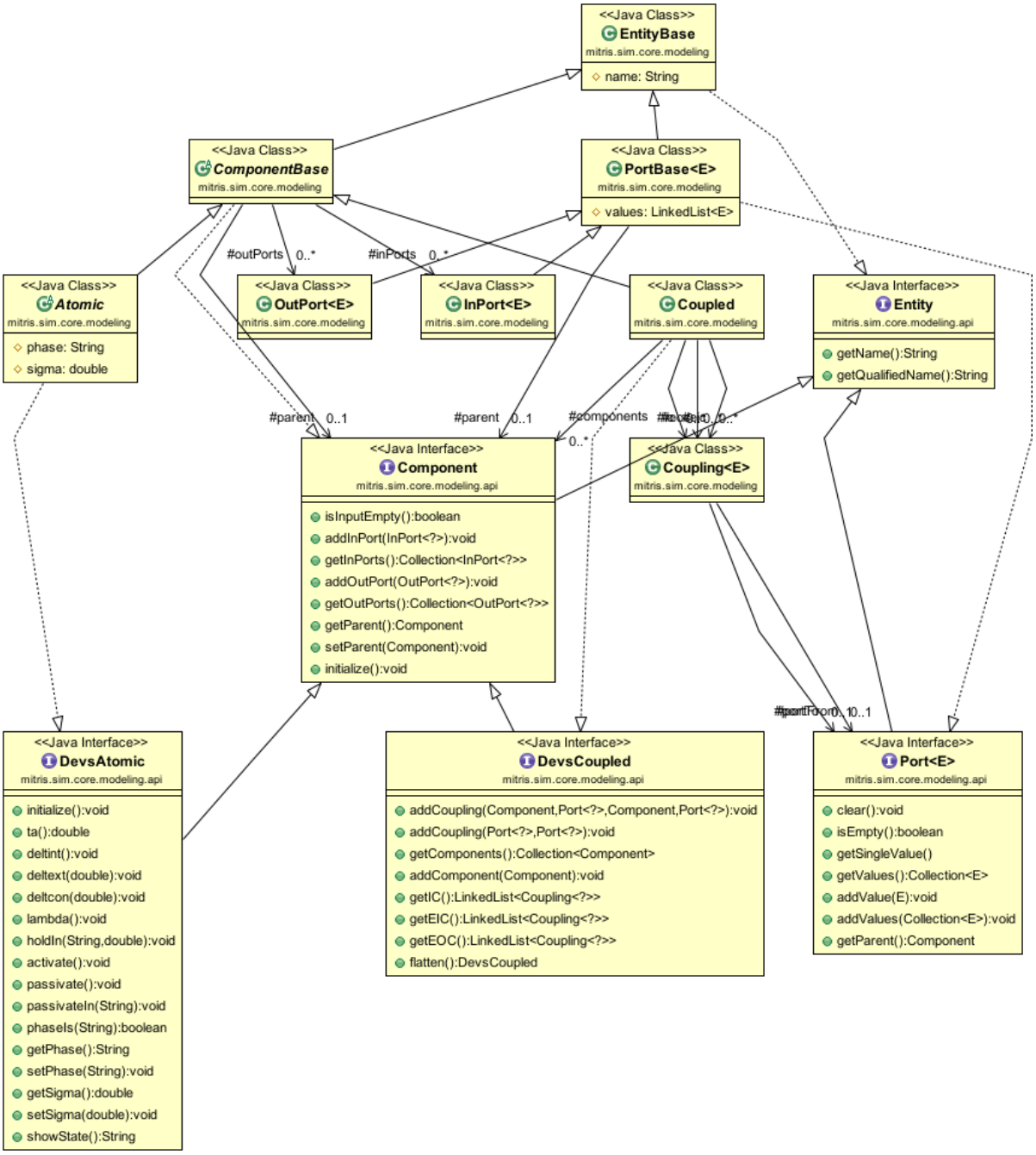

Figure 1. General architecture of the MitRis modeling layer.

The *Coordinator* implements a sequential simulation algorithm (also known as DEVS simulation protocol) using virtual time. *CoordinatorParallel* implements a parallel DEVS simulation using virtual time and a given number of threads. The *RTCentralCoordinator* performs the same operations as *CoordinatorParallel* but uses real-time. Our future work includes the implementation of a *CoordinatorDistributed* class for the execution of distributed simulations, and *RTCoordinator* and *RTSimulator* classes for the execution of real time reactive DEVS simulations [1] (see Chapter 8). Finally, the *CoordinatorProfile* class implements a proxy pattern to extract a set of metrics from any of the specialized coordinators.

Features

Port-identity and Port-extensibility

MitRis 1.0 provides a set of classes to define DEVS ports. Every Port is an *Entity*, as described above and can be either

an input or an output port. MitRis DEVS input or output port implements the *Port* interface (see Figure 1). Each port is also associated with a specific Entity i.e. a message data-type and has a *fullyQualifiedName* (per Eclipse Plugin development environment [PDE] parlance). From its *fullyQualifiedName*, a port can be traced back to its containing component (atomic or coupled) in the model hierarchy. *Port* interface allows the software engineer the rapid implementation of different communication alternatives between models, using for example, Java Message Services (JMS), Apache Camel, Websphere, etc.

Flattening

Flattening is a mechanism that applies the *Closure under Coupling* DEVS principle which states that every coupled model can be replaced by an atomic model. Since each coupled model behaves exactly as the model that it contains, a DEVS engine can replace all the coupled models by their corresponding sub-models to the irreducible atomic models. Flattening a coupled model to the basic atomic components reduces the depth of the entire model, thereby, improving performance as the messages need not travel up the coupled-coordinator hierarchy. Thus, the corresponding DEVS coordinators are no longer needed. As can be seen in Figure 1, the *DevsCoupled* interface declares this simplification mechanism in the *flatten* member function, which is implemented in the *Coupled* class. By default, all the MitRis coordinators flatten the simulation model just before the simulation is initialized. However, this is an optional parameter in the *Coordinator* constructor method.

Parallelism

It is well known that the Java Development Kit (JDK) incorporates many facilities to develop parallel and distributed programs: Threads, *Remote Method Invocation (RMI)* or Sockets are some examples [2]. In the current version of MitRis, parallelism is included using Executor services based on pool of threads. These threads concurrently execute all the *output functions* in parallel and all the *transition functions* in parallel. To this end, we group all the output functions in a list called *lambdaTasks* in the *CoordinatorParallel* class (see Figure 2), and all the transition functions in the *deltfcnTasks* list of the same class. It is worthwhile to mention that the parallelism is implemented in the simulation layer. Thus, every MitRis DEVS model can be simulated sequentially or in parallel by just changing the coordinator. Finally, distributed simulations are included as part of our immediate future work, using DEVS/SOA as the base of our design [3].

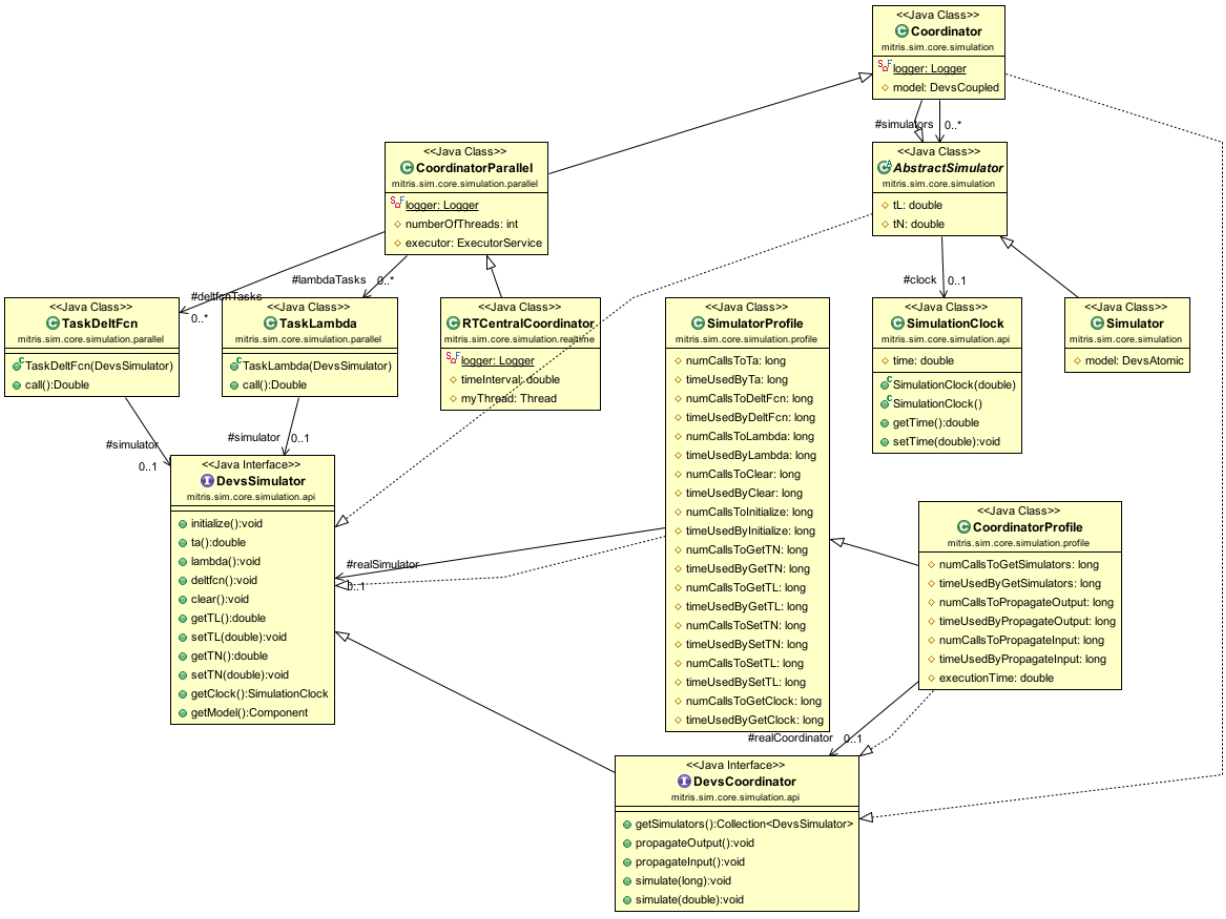

Figure 2. General architecture of the MitRis simulation layer.

Engine Footprint

The performance of a given DEVS model mainly depends on the simulator used. This, translated to MitRis, implies that the performance of a given model depends on the coordinator selected to simulate that model. In MitRis, we have implemented a DEVS profiler using two classes, namely, *CoordinatorProfile* and *SimulatorProfile*. Applying a proxy pattern, these classes are able to collect different metrics of the coordinator selected to perform the simulation. These metrics measure various statistics for *DevsSimulator*, *DevsCoordinator*, and by extension, *DevsAtomic* and *DevsCoupled* member functions, as well as their accumulated execution time. In the UML diagram depicted in Figure 2, all the parameters measured are contained in both *CoordinatorProfile* and *SimulatorProfile* classes. Java VisualVM [4] is utilized to measure simulations' performance in terms of percentage of CPU or memory used, time consumed by a given member function, detecting hotspots, etc.

MITRIS DEVSML ECLIPSE DEVELOPMENT STUDIO

The MitRis 1.0 DEVS engine is available for use with any JVM-based languages, such as Groovy, Scala, Xtend, etc. A demonstrative example is available online [6] where various languages can be used at the Modeling layer leveraging the modeling API interface (Figure 3). MitRis 1.0 engine is made available as the platform-specific-model (PSM) of a more abstract DEVS Modeling Language (DEVSML), based on XML-Based Finite Deterministic DEVS [7] and language introduced in [8] and later developed in [1,9]. The DEVSML Stack shown in Figure 4 describes the layered architecture of platform-independent nature of DEVSML [8,10]. 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 [11]. 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 3). 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 [28] and CD++ [20] are already having such repositories..

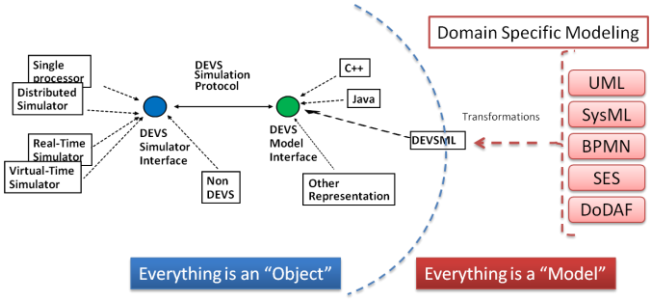

Figure 3. Integrating theory of modeling and simulation framework with MDE

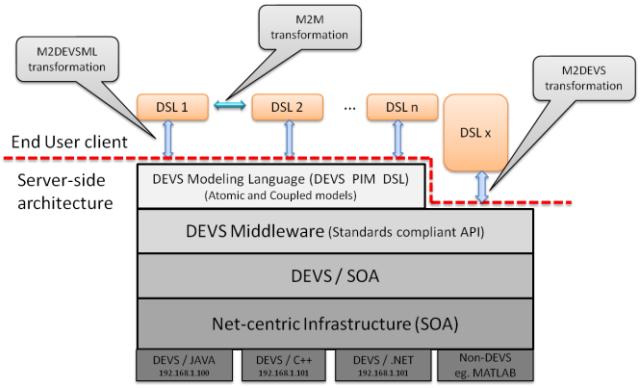

Figure 4. DEVSML Stack showing DEVS modeling language and various transformations

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 [12] 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 [13].
4. It can be configured with different DEVS-engines using DEVSML configuration settings. The default is MitRis M&S engine. The other available engine is DEVSJAVA [14].
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 [14] 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 5.

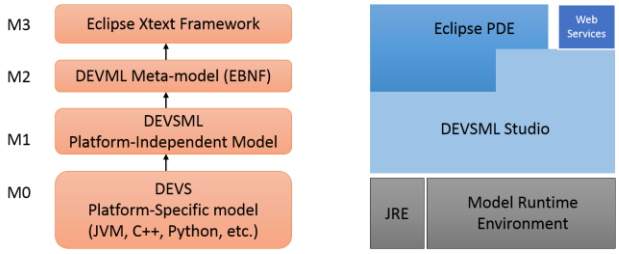

Figure 5. Metamodeling and DEVSML Studio

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 [15]. The MitRis M&S Engine is available both as .jar and as an Eclipse plugin. The DEVSML Studio is available as an installable Eclipse feature.

CASE STUDY: TRAPEZOIDAL PULSE SHAPER

In recent years, 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 MitRis DEVSML Studio and analyze 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 [5]:

$$d^k(n) = v(n) - v(n-k) \quad (1)$$

$$d^{k,l}(n) = d^k(n) - d^k(n-l) \quad (2)$$

$$p(n) = p(n-1) + m_2 \cdot d^{k,l}(n), n \geq 0 \quad (3)$$

$$r(n) = p(n) + m_1 \cdot d^{k,l}(n) \quad (4)$$

$$s(n) = s(n-1) + r(n), n \geq 0 \quad (5)$$

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:

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

According to [5], 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 6 shows a block diagram of the digital trapezoidal shaper. $\text{DELAY}_{\{1,2\}}$ are the delay pipelines, $\Sigma_{\{1,2,3\}}$ are adders/subtractors, $\text{ACC}_{\{1,2\}}$ are accumulators and $\text{X}_{\{1,2\}}$ are multipliers. We can also find coupled models like 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 6 and the corresponding MitRis DEVSML model, the following classes are designed:

Atomic models:

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

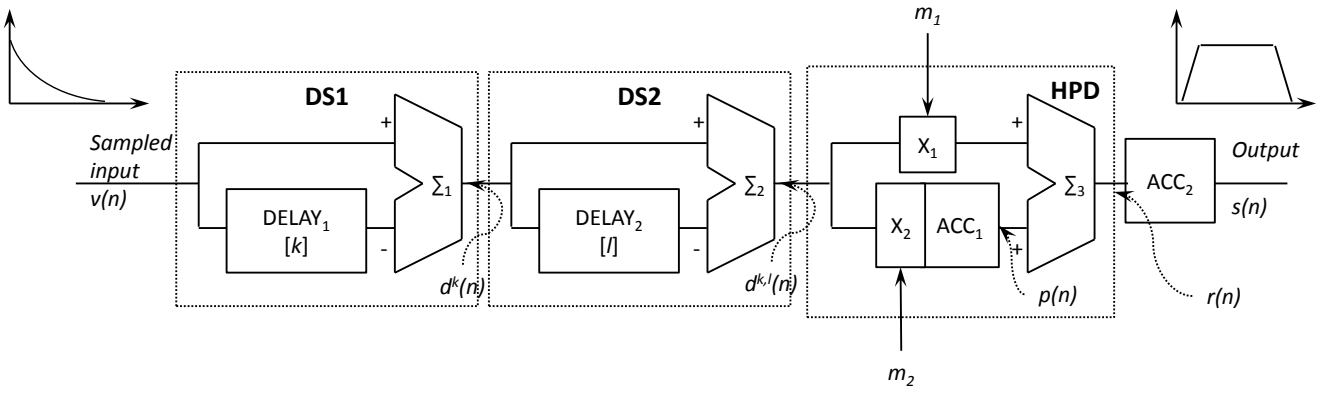

Figure 6. Diagram of the digital trapezoidal shaper. The elements are: DELAYn - a delay pipeline, Σn – an adder/subtracter, ACCn – an accumulator, Xn – a multiplier. DSn is a delay-subtractor unit, HPD is a high-pass filter deconvolver.

- **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 m1 or m2 in Figure 6) 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 $X_{\{1,2\}}$.
- **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 $DELAY_{\{1,2\}}$.

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 6 contains a *ShiftRegister* atomic model and an *AdderSubtractor* atomic model.
- **HighPassDeconvolver (HPD)**: The HPD coupled model in Figure 6. This model includes two *Constant*, two *Multiplier*, one *Accumulator* and one *AdderSubtractor* atomic models.
- **Trapezoidal**: This is the trapezoidal digital shaper, including *DS1*, *DS2*, *HPD* and the final *Accumulator* in Figure 6.
- **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 7 and coupled model is shown in Figure 8. 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 "^". In the coupled model, each port displays the fullyQualifiedName and its data-type enclosed within "<< >>". 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 100,000 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:

- $T_{clk} = 2\times10^{-5}$ 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 9 shows the detection of one of these 100,000 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 7 Operating System.

To obtain the characteristic of our DEVS coupled model, we simulate the model using the *CoordinatorProfile* class depicted in Figure 2. 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

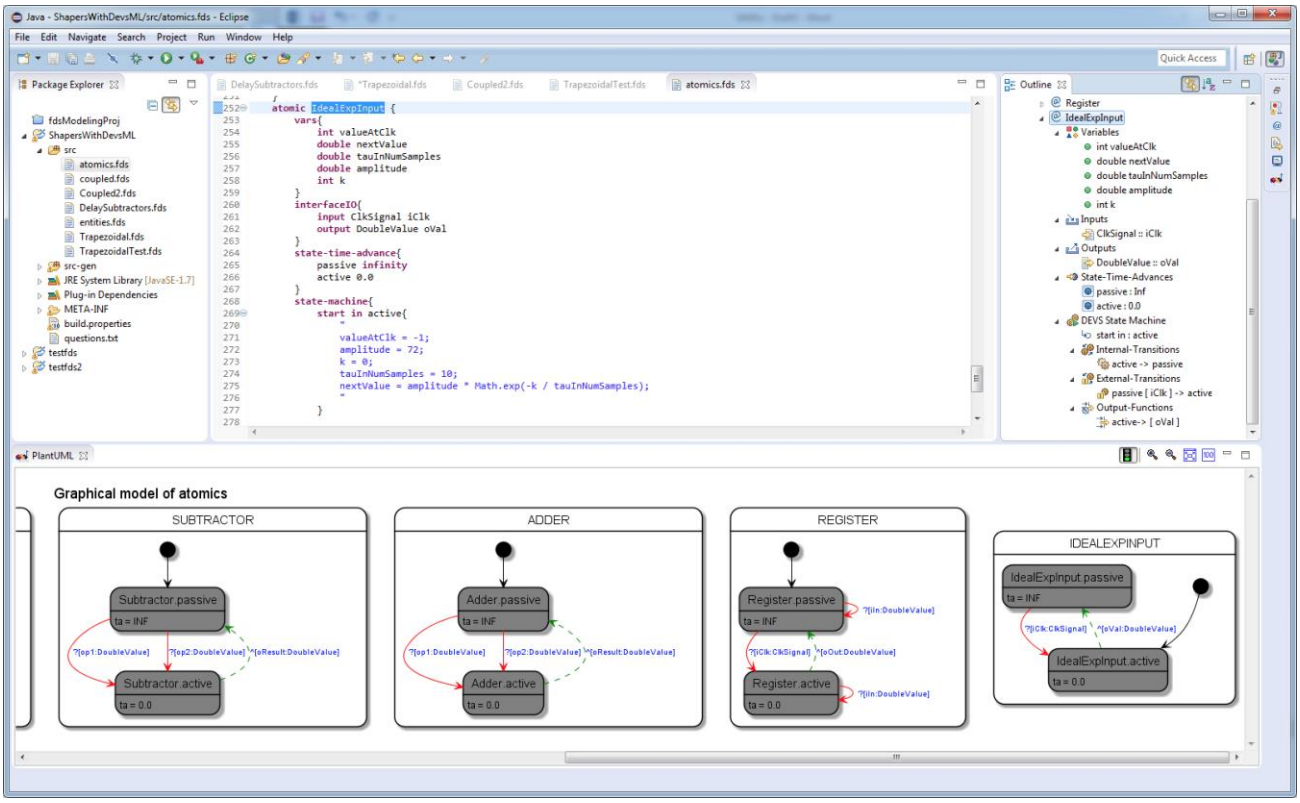

Figure 7. MitRis DEVSML Studio showing auto-generated DEVS Atomic visual and hierarchical representation

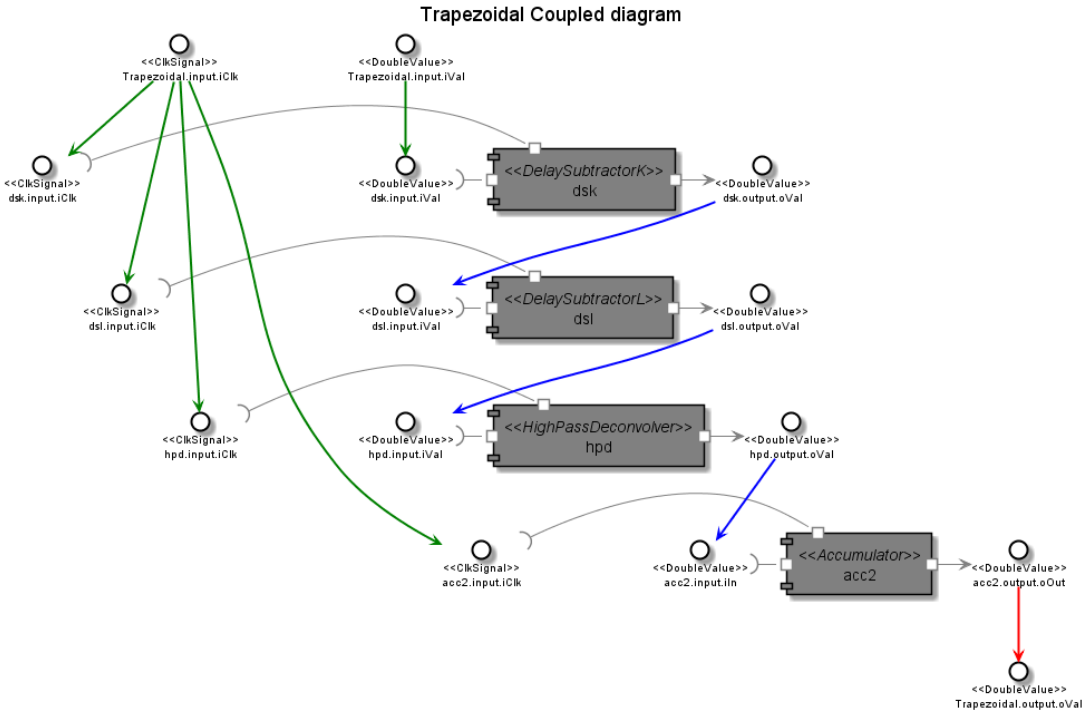

Figure 8. Auto-generated Coupled model representation with port data-types. Green arrows represent External Input Coupling (EIC), blue arrows represent Internal Coupling (IC), and red arrows represent External Output Coupling (EOC)

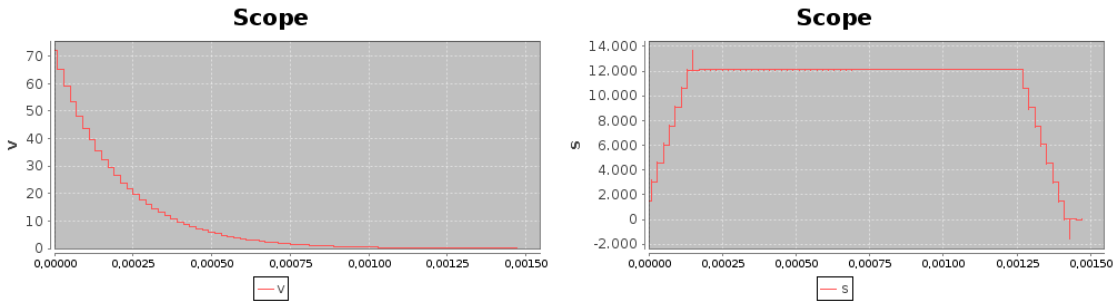

Figure 9. Exponential input (left) and trapezoidal output (right) generated by the MitRis shaper model for one single event.

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 (ACC_1 in Figure 6), with an execution time equal to 3.44 s for the transition function (including the external, internal and confluent transitions).

Figure 10 shows the performance comparison of three coordinators implemented in MitRis 1.0. *CoordinatorProfile* is not included in the comparison because of the overhead (up to 457 seconds of wall clock time).

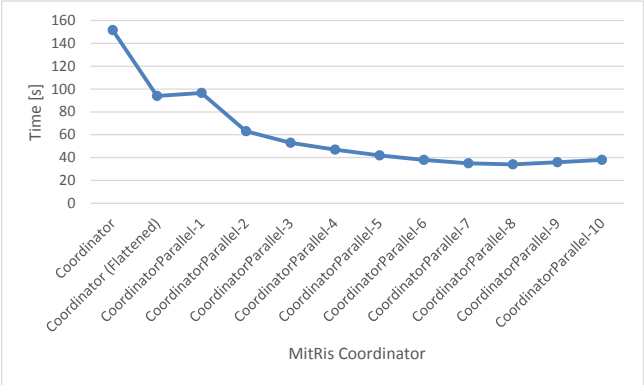

Figure 10. Performance comparison of Coordinators

Per the case study, we have, in total, 14 atomic models in *TrapezoidalTest*. The Sequential Coordinator 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 version, 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 (Figure 10).

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 introduced MitRis 1.0 containing both the DEVS M&S engine and an Eclipse editor. The engine is based on latest JVM features and implements advanced executor framework for a parallel and distributed execution on JVM. The engine also introduces a new port-identity and –extensibility framework for netcentric execution, flattening capability for the coupled model and a profiler for inspecting the performance of any particular model during a simulation execution. The MitRis DEVSML Eclipse 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. Finally, we examined the performance of the engine for Sequential and Parallel execution with multiple thread and achieved a maximum speedup of 4.45 for a parallelized flattened model with 100,000 events.

We would like to integrate various other DSLs as shown in Figure 3 in our next release of MitRis and as described in [1].

REFERENCES

1. Mittal, S. and Risco-Martín, J. L. Netcentric System of Systems Engineering with DEVS Unified Process, CRC Press, 2013.
2. Garg, V. K. Concurrent and Distributed Computing in Java Wiley-IEEE Press, 2004.
3. Mittal, S.; Risco-Martín, J. L. and Zeigler, B. P. DEVS/SOA: A Cross-Platform Framework for Net-centric Modeling and Simulation in DEVS Unified Process SIMULATION, 85, 419-450, 2009.
4. Java Visual VM, http://visualvm.java.net, last accessed Dec. 6, 2014
5. 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 Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 353, 261-264, 1994.
6. EFP Samples, available at: http://duniptech.com/wp/languages/efp, last accessed Dec. 6, 2014
7. Mittal, S., Hwang, M.H., Zeigler, B.P., XFD-DEVS: XML-Based Finite Deterministic DEVS, available at: http://www.duniptechnologies.com/research/xfddevs/, last accessed Dec. 6, 2014
8. Mittal, S., and S. A. Douglass. DEVSML 2.0: The Language and the Stack. Symposium on Theory of Modeling and Simulation, Spring Simulation Multiconference. Orlando, FL, 2012.
9. Mittal, S., Martin, J.L.R, Model-driven Systems Engineering for Netcentric Systems of Systems with DEVS Unified Process, Winter Simulation Conference, 2013
10. Mittal, S., and S. A. Douglass. From Domain Specific Languages to DEVS Components: Application to Cognitive M&S, Proceedings of the Workshop on Model-driven Approaches for Simulation Engineering -- Spring Simulation Multiconference. Boston, MA., 2011
11. Mittal, S., J. L. R. Martin, and B. P. Zeigler, DEVSML: Automating DEVS Simulation over SOA using Transparent Simulators, DEVS Symposium, Spring Simulation Multiconference. Norfolk, VA., 2007
12. Eclipse Xtext, available at: http://www.xtext.org, last accessed Dec. 6, 2014
13. PlantUML, available at: http://plantuml.org, last accessed Dec. 6, 2014
14. EclEmma Java Code Coverage, available at: http://eclemma.org, last accessed Dec. 6, 2014
15. MitRis DEVSML Studio Update Site: http://duniptech.com/modeling/studio/releases/update, last accessed Dec. 6, 2014
16. Zeigler, B.P., Praehofer, H., Tim, T.G., Theory of Modeling and Simulation, Academic Press, 2000
17. DEVSJAVA, available at: http://acims.asu.edu/software/devsjava, last accessed Dec. 6, 2014
18. DEVS-Suite, available at: http://devs-suitesim.sourceforge.net/, last accessed Dec. 6, 2014
19. CoSMoS: available at; http://acims.asu.edu/software/cosmos, last accessed Dec. 6, 2014
20. CD++, available at: http://cell-devs.sce.carleton.ca, last accessed Dec. 6, 2014
21. Tendeloo, Y.V., Vangheluwe, H., The modular architecture of the python(P)DEVS simulation kernel, Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium, 2014.
22. ADEVS, available at: http://web.ornl.gov/~1qn/adevs/, last accessed Dec. 6, 2014
23. JAMES II, available at: http://wwwmosi.informatik.uni-rostock.de, last accessed Dec. 6, 2014
24. Kim, T.G., Sung, C.H., et.al, DEVSim++ Toolset for Defense Modeling and Simulation and Interoperation, Journal of Defense M&S, 8(3), 129-142, 2011.
25. J. DAVILLA, M. UZCATEGUI, GALATEA: A Multi-agent simulation platform. In International Confer- ence on Modelling, Simulation and Neural Networks MSNN'2000, Me´rida, Venezuela, 2000.
26. Traore, M.K., SimStudio: A Next Generation modeling and simulation framework, Proceedings of the 1st international conference on Simulation Tools and techniques for Communications, Networks and Systems and Workshops, 2008.
27. PowerDEVS, available at: http://powerdevs.sourceforge.net, last accessed Dec. 6, 2014
28. MS4Me, available at: http://www.ms4systems.com/pages/main.php, last accessed Dec. 6, 2014
29. Virtual Laboratory Environment, available at: http://www.vle-project.org/wiki/Main_Page, last accessed Dec. 6, 2014

AUTHOR BIOGRAPHY

SAURABH MITTAL is the founder and principal of Dunip Technologies LLC, USA. He is also affiliated with National Renewable Energy Lab, Department of Energy, Golden, Colorado USA. He received both a PhD (2007) and MS (2003) in Electrical and Computer engineering from the University of Arizona, Tucson. He can be reached at smittal@duniptech.com

JOSE LUIS RISCO MARTIN is Chief Technology Officer of Dunip Technologies, LLC USA and is associate professor at the Department of Computer Architecture and Automation of Universidad Complutense de Madrid, Spain. He received both a PhD (2004) and MS (1998) in Physics from Univer-sidad Complutense de Madrid. He can be reached at jlrisco@dacya.ucm.es