DEVS-Suite: A Simulator Supporting Visual Experimentation Design and Behavior Monitoring

Sungung Kim
Hessam S. Sarjoughian
Vignesh Elamvazhuthi

Arizona Center for Integrative Modeling & Simulation
Dept. of Computer Science and Engineering
Arizona State University
Tempe, AZ 85281, USA

Keywords: DEVSJAVA, DEVS-Suite, experimental design, simulation monitoring, visual complexity

Abstract

Complexity associated with the design of experiments for simulation models can be reduced through visualization. DEVS-Suite, a new generation of the DEVS Tracking Environment which itself was extended from DEVSJAVA, supports visual design of experiments and introduces simulation data visualization. Data generated by the selected models can be collected dynamically and displayed as time-based trajectories. These capabilities complement animation of DEVS model components and their interactions. A service-oriented software system is modeled to illustrate the novel modeling features for DEVS simulations. Another example is developed in Ptolemy II and SimEvents to show the reduced visual complexity afforded by DEVS-Suite.

1 INTRODUCTION

Modeling is commonly used for understanding, engineering, and operations of systems. Simulation tools such as DEVSJAVA [1], SimEvent [8], and Ptolemy [6] can be used for modeling complex discrete systems. Each provides its own unique approach for model specification and simulation visualization. A key capability for simulation studies is automated design of experiments such that models can be easily chosen and their simulated dynamics monitored at run-time [2].

Considering the DEVS modeling framework [16], a variety of simulators have been implemented for it in different programming languages. They are used for simulating discrete event models, and in some cases continuous and discrete, across various application domains. Tools such as CD++ [14], DEVSJAVA, and DEVS Tracking Environment (DTE) [13] conform either to parallel or classic DEVS formalisms and support visualization of pre-built models. For example, DEVSJAVA, a Java-based implementation of Parallel DEVS, supports visualization of hierarchical model components and animation of message exchanges among atomic and coupled model components. However, it does not support on-the-fly selection and monitoring of model components and displaying data trajectories at run-time. Similar to many other tools, models are embellished with code to gather data of interest and are made available to console or written to external files for post processing.

A key advantage of simulators such as SimEvents and Ptolemy II is support for viewing simulation data as time trajectories. SimEvents, which is an extension of Simulink, [8] supports hierarchical activity-based models from pre-built components including queues, servers, switches, gates, timers, and generators for entities, events, and signals. These components can be visually composed to develop bigger models. Model parameters such as congestion, resource contention and processing delays can be monitored at run-time. The pre-built, strongly typed monitoring components such as Signal Scope can be used to plot events or the states of the models along time axes. Monitoring a model's input/output ports may require several plotters.

Ptolemy II supports different types of models (e.g., continuous and discrete) using a graphical user interface called Vergil. Pre-existing code with information about their couplings allows automatic code generation in XML format. Users can visually synthesize hierarchical models from pre-built components which have symbolic representations. The animation feature displays the models that are active at different instances of time. The simulation results can be monitored with plotters which are part of the model layout. Similar to SimEvents, plotters in Ptolemy II are strongly typed.

Given the importance of selecting components of a hierarchical model with support for monitoring and visualizing inputs and outputs of any component, we developed the DEVS-Suite simulator. We begin in Section 2 with a review of the DEVSJAVA and DEVS Tracking Environment simulators and the TimeView, a module that supports plotting input, output, and state trajectories. In Section 3, we describe the DEVS-Suite simulator and use it to model a ser-

vice-oriented computing system. In Section 4, we consider a simple assembly line system. We model this system in DEVS-Suite, Ptolemy II, and SimEvents and then compare them to show their differences for design of experiments and monitoring of simulation results with respect to a visual complexity metric. In Section 5, we summarize and discuss future research.

2 BACKGROUND

As we mentioned in the previous section, a number of simulators have been built for simulating modular, hierarchical DEVS models. Among these, we consider DEVSJAVA and DEVS Tracking Environment (DTE) since they are used for developing the DEVS-Suite simulator [1]. Here, we review the basic concepts for the DEVSJAVA and DTE. The DEVS-Suite simulator offers new and integrated features that can help users devise experiments and conduct simulations more easily. There is no need to delve into the details of the simulator protocol design and implementation since the simulation engine used in DEVS-Suite is the same as the one used in DEVSJAVA and DTE.

2.1 DEVSJAVA

A commonly used tool for simulating parallel DEVS models is DEVSJAVA. It displays a view of the entire hierarchy of the simulation model using components-within-components style. For any coupled model, one or more messages simultaneously travel along coupling paths that connect atomic to atomic, atomic to coupled, and coupled to atomic model components [7]. The design of the DEVSJAVA separates execution control from the tightly integrated simulator kernel and view. That is, visualization of models and their animations are supported by simView, a module that supports user interactions and control of simulation execution. The control supports logical- and soft real-time simulation execution. The states for every model components can be individually examined (viewed) when the simulator is stopped or paused (i.e., at the end of a simulation cycle). The visualization of the messages is allowed for the entire model. Furthermore, animation of the messages is not synchronized against the simulation's execution speed.

2.2 DEVS Tracking Environment

To support automated design of experiments through observing inputs and outputs and as applicable common state variables phase and sigma, the DEVSJAVA Tracking Environment was developed. Its design is based on the classical Model-View-Control (MVC) architecture [9]. In addition to the Model, View, and Control modules, another module called Façade layer is also used. The data available in the Façade module can be accessed by the Controller and the View. With the MFVC architecture, the simulation data sets can be displayed with one or more views. The data generated by the Model module are obtained during simulation execution and made available to the View module. The separation between the Model and the View modules and the presence of the Façade module, model components can be arbitrarily selected for monitoring and their dynamics displayed at run-time or exported for external use. The encapsulation and modularization through the Façade module significantly reduce dependencies between the Model and the Control and View modules.

DTE's key capability is simplifying design of experiments for simulation models. Its graphical user interface allows a user to select model components to be monitored and thus design experiments in terms of components' inputs/outputs and state variables. Simulation model data sets, which include states such as Time of Next Event, Time of Last Event, and user selected input/output ports, can be dynamically tracked. The user, therefore, is able to observe simulation data for any number of atomic and coupled models without any code development. The data can also be displayed in a tabular format using a Tracking Logger or exported to CSV files.

2.3 TimeView

The TimeView is a module developed for run-time display of data sets as two dimensional plots (every plot has x and y coordinates). Its operation is similar to an oscilloscope. The TimeView is a passive module. It can display sets of (x, y) values where x (or y) values are plotted with respect to y (or x). In order to use it for plotting time-based simulation data, the x-coordinate for all plots is defined to represent time. The allowed values for the x-coordinate are integers (e.g., natural numbers) and the unit can be alphanumeric (e.g., seconds). The y-coordinate can be numbers (integer and real) and string. The values for the y-coordinates are generated inputs and outputs that are generated by atomic or coupled models. As an example, size of a queue can be plotted at time instances 0, 1, 2, ..., 100. The time increment duration and the units for time and variable to be plotted can be set by user.

For every atomic and coupled model component, one or all of its input and output ports can be viewed independently. The time trajectories for all variables of a model that are selected to be tracked are combined into a single view. There is no support for overlaying multiple variables into a single plot and multiple time trajectories from different models cannot be combined into a single trajectory (e.g., given model output port out_A from model A and output port out_B from model B, a single trajectory cannot be created to display data from both out_A and out_B ports).

3 DEVS-SUITE SIMULATOR

The architecture of the DEVS-Suite simulator is the same as that of the DTE. The architecture separates simulation models (i.e., the source of data) from how they are controlled and viewed. As shown in Figures 1 and 2, the DEVS-suite package diagram consists of Model, Façade, Controller, and View packages and their sub-packages [5]. The design and implementation of the DTE's Façade layer is extended. Mainly, its connection to the Model is altered in order to include DEVSJAVA's simView to the View. Therefore, the View module has the simView, TimeView, and TrackingLog packages. Given the presence of the Façade, the View allows combined animation and tracking (i.e., viewing at run-time trajectories and tabulation of simulated data). Users, therefore, may choose simView and/or Tracking. Similarly, the Controller design is extended to handle simulation animation speed (i.e., the speed at which messages are exchanged among atomic and coupled model components).

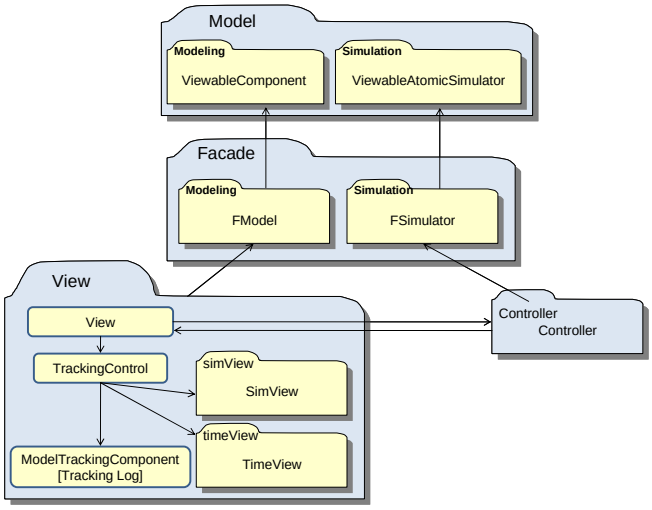

Figure 1: DEVS-Suite MFVC Package Diagram

From the scalability perspective, a representative set of simulation models having 20 to 7000 model components were devised and simulated for a service-oriented software system called voice communication system (VCS) [5][12][15]. We used a desktop machine with Core 2 Duo 2.66 GHz CPU and 4GB RAM. Obviously, the execution speed afforded by the simulator depends on the number of components that are chosen to be tracked and whether or not the model is animated. The simView was turned off and no data was tracked. Initially, the wall-clock simulation time increases proportionally and then slowly becomes exponential.

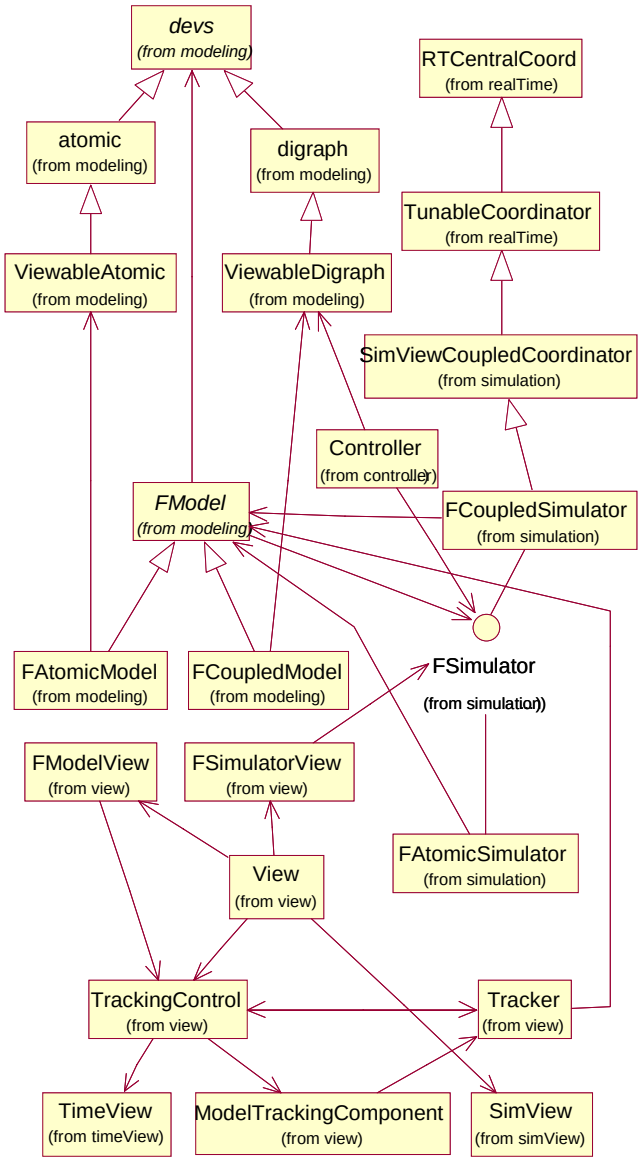

Figure 2. DEVS-Suite Class Diagram

3.1 Interface and Monitoring of Simulation Data

DEVS-Suite user interface consists of four parts: (1) Model Viewer at the top left corner, (2) Simulator Control at the bottom left corner, (3) simView at the top right hand corner, and (4) TimeView at the bottom right hand corner (see Figure 3). In order to make better use of available display space, the Model Viewer and Simulator Control are combined to form a part which we call MVSC. A user, therefore, can choose to view any one of the TimeView, SimView, or MVSC parts within the DEVS-Suite interface since any two of the three parts can be hidden. A user may also view MVSC with either TimeView or SimView. Alternatively, the user can hide the MVSC part and only view the TimeView and SimView while executing the model using the execution buttons provided in the menu bar.

Both block and tree views of hierarchical model components are available. It provides flexibility in that a user can select animation and/or tracking of simulation model components as time trajectories. The tree view is used for choosing model components and deciding which input and output ports to monitor. For atomic models, pre-defined state variables and basic simulator variables can also be chosen and tracked. The block model is used for animation.

The dynamics of every atomic and coupled model can be individually displayed with TimeView. The semantics of the data generated by the Model module in DEVS-Suite is applied to the TimeView. Therefore, to display time-based state and input/output data, simulation time is used to synchronize generation of the time trajectories. Users have the flexibility to select animation and tracking view options for any number of atomic/coupled models. They can set the unit for data that is to be monitored as well as the time axis. The time increment, units, and the selection of data to be observed can be set as shown in Figure 4.

Figure 3: DEVS-Suite UI with Model Viewer, Simulator Control, simView, and TimeView

If the Tracking option is selected and one or more model components are also chosen to be viewed via TimeView and TrackingLog, a tracker collects simulation data sets for each selected model. For example, we consider an SOA-complaint DEVS model shown in Figure 3. This model represents Voice Communication Service (VCS) and Travel Agency Service (TAS). Voice communication service called VoiceComm is an atomic service that broadcasts voice data streams in response to Subscriber1 and Subscriber2 atomic services. The system is devised to support audio data that can be sampled at any of the following rates – 44.1, 88.2, 136.4, 176.4, 220.5 KHz. For the VoiceComm service, its input and output ports can be tracked. Figure 4 shows the VoiceComm requested and provided data rates that are selected to be tracked.

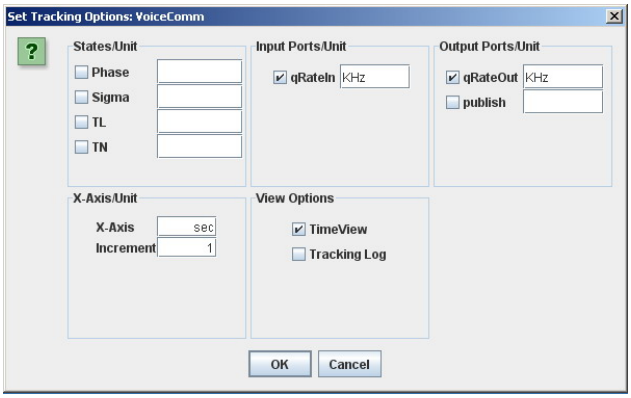

Figure 4: Tracking VoiceComm Model

3.2 Simulation Control Logics

The DEVS-Suite simulator supports the control logics that are provided by DEVSJAVA and DTE. The simulation execution can be controlled using Run, Step, Step(n), Request Pause, and Reset (see Figure 3). The other controls are Show Coupling, real-time factor, and a new control called Animation speed. The control logic manages the simulator execution and the data that is provided to SimView. DEVS-Suite handles the simView animation and TimeView trajectories independently. This is advantageous since the speed of animation for messages may not be the same as the speed at which time trajectories can be displayed. The Controller supports compilation of models at run-time. This is important to allow users to configure the path to packages of model classes and source files as well as model package names as in DEVSJAVA. After the configuration, a user selects a package name and then the list of available models in the selected package can be displayed.

4 SIMULATION MODEL AND COMPLEXITY EVALUATION

To evaluate the tracking and viewing supported by DEVS-Suite, we compared it with Ptolemy II and SimEvents simulators. The Assembly Line model [4] shown in Figure 5 is included in Ptolemy II. In this model, jobs are generated by a Generator model at predefined intervals and are serviced by three processors P_1, P_2, and P_3 in a cascade fashion. The service time for each job is specified by a Processor. We developed the same model in SimEvents and DEVS-Suite simulation tools (see Figure 6) [3].

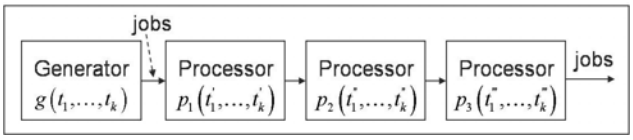

Figure 5: Conceptual Assembly Line Model

We define a simple visual complexity metric as the numbers of components that are displayed for a given model. The total number is equal to the sum of the number of block components (i.e., models that represent dynamics of the modeled system) and their ports and couplings. The block components are categorized into logical and monitoring components. The number of logical component models among SimEvents, Ptolemy II, and DEV-Suite vary due to their underlying modeling approaches. A consequence of the modeling differences is that the number of ports and couplings for SimEvents and Ptolemy II are higher as compared with DEVS-Suite.

A key difference among these tools is the presence or absence of monitoring components. As shown in Table 1, the number of the displayed components for the Assembly Line model varies significantly even for such a small model. The visual complexity metrics for SimEvents is 58 vs. 23 for DEVS-Suite. The visual complexity for Ptolemy II is better as compared with SimEvents, but not with respect to DEVS-Suite. The complexity metric also depends on flat vs. hierarchical models. In DEV-Suite, the hierarchy of a model does not impact the visual complexity metric. In comparison, the visual complexity metric for hierarchical models developed in Ptolemy II and SimEvents may be affected given the necessity of couplings monitoring components to logical components.

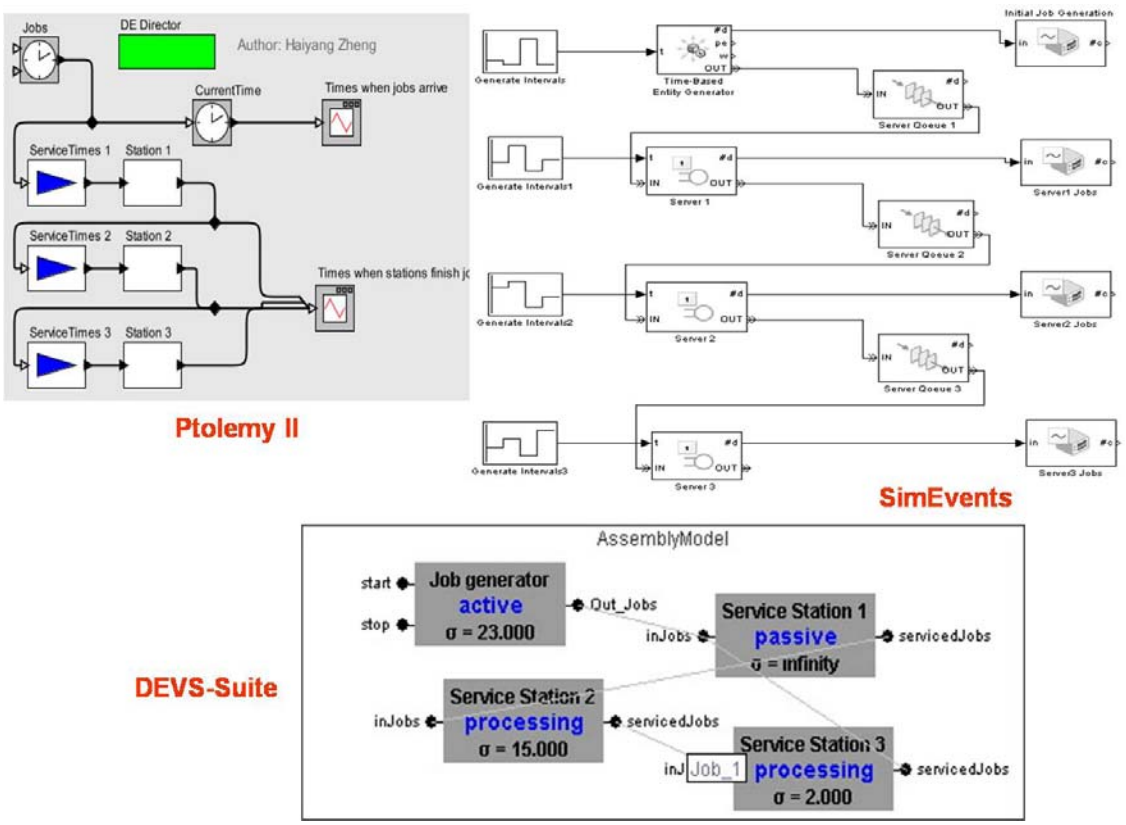

Figure 6: Models of the Assembly Line System in DEVS-Suite, Ptolemy II, and SimEvents Simulators

The main cause for the different visual complexity metrics is the use of monitoring components. They increase the number of ports and couplings for SimEvents and Ptolemy II. It can be easily seen that TimedPlotter for Ptolemy II and SignalScopes for SimEvents are required for monitoring inputs and outputs of components. They, therefore, add to the visual complexity of model display. In contrast, in DEVS-suite, the user selects model components that are to be monitored using dialogue boxes. It is noted that the monitoring components in SimEvents and Ptolemy II collect data, but in order to observe this data, separate windows must be created.

Thus, it is straightforward to observe that the DEVS-Suite visual model complexity is not affected by the number of models to be monitored. However, the DEVS-Suite's block model component view suffers from the overlapping of the couplings. This problem also can be seen to a lesser degree in the other tools. Some research aimed at overcoming the crossing of couplings has results in the visual modeling environment CoSMoS [11], which supports simulating models using DEV-Suite.

Some factors that are related to monitoring components are shown in Table 2. All three tools support plotting numbers with DEVS-Suite also supporting strings. In contrast, SimEvents and Ptolemy II plotters have user-friendly features such as zoom in and zoom out. Ptolemy II supports combing multiple variables which could be from multiple logical components into a single plot (i.e., overlaying of trajectories). DEVS-Suite allows viewing together independent plots for a single model, but has no support for overlaying time trajectories. The plotters for Ptolemy II and SimEvents are specialized and efficient. In contrast, while the TimeView for DEVS-Suite is generic, it lacks flexibility for the resizing of plots. Finally, as noted above, the monitoring components for SimEvents and Ptolemy must be coupled with logical components.

Table 1: Visual Complexity Metric

	SimEvents	Ptolemy II	DEVS-Suite
Logical components	11	9	5
Ports	29	15	10
Couplings	14	11	4
Monitoring components	4	2	0
Total number of components	58	37	23

Table 2: Comparison of Component-based Simulators

Plotters		SimEvents	Ptolemy II	DEVS-Suite
Data Types	Numbers	Yes	Yes	Yes
	String	No	No	Yes
Consolidate into a single plotter		No (single plotter per port)	Yes	Yes (per model)
Specialized		Yes	Yes	No
Generic		No	No	Yes
Require couplings or an output port		Yes	Yes	No

5 CONCLUSION AND FUTURE WORK

In this paper, we presented the DEVS-Suite simulator and described its capabilities for designing experiments and visualization of simulation executions both as time trajectories and animation. The simulator simplifies designing simulation experimentations and observing their dynamics without unnecessarily complicating the display of models. Systems such as service-oriented software systems can be more easily configured for colleting simulation data and run-time examination of the model behavior via time trajectories. The simulator can also be used for education [10]. This is because DEVS-Suite simplifies creation of simulation scenarios (i.e., experimental designs) which in turn aids verification and validation of systems. In terms of future work, it is useful for the DEVS-Suite to support real-time visualization when useful. It is desirable for animation and visualization of time trajectories to be synchronized. Better support for manipulating views of trajectories at varying levels of details is also important.

Acknowledgement

This research was partially supported by NSF Grant numbers #BCS-0140269 and #CCF-0725340. The TimeView module was developed by Robert Flasher as part of his undergraduate senior project in the Computer Science and Engineering department at Arizona State University.

References

[1] Arizona Center for Integrative Modeling and Simulation. 2007. http://www.acims.arizona.edu/SOFTWARE.

[2] Dalle, O. 2007. An Instrumentation Framework for Component-Based Simulations based on the Separation of Concerns Paradigm. 6th EUROSIM Congress, Ljubljana, Slovenia.

[3] Elamvazhuthi, V. 2008. Visual Component-Based System Modeling with Automated Simulation Data Collection and Observation, MS Thesis, Department of Computer Science and Engineering, Arizona State University, Tempe, AZ, USA.

[4] Jayadev, M. 1986. Distributed discrete-event simulation. ACM Computing Surveys, 18(1): 39-65.

[5] Kim, S. 2008. Simulator for Service-based Software Systems: Design and Implementation with DEVS-Suite. MS Thesis. Computer Science and Engineering. Arizona State University, Tempe, AZ, USA.

[6] Lee, E. A. 2003. Overview of the Ptolemy Project. University of California, Berkeley, CA, USA.

[7] Mather, J. 2003. The DEVSJAVA Simulation Viewer: A Modular GUI that Visualizes the Structure and Behavior of Hierarchical DEVS Models, MS Thesis, Electrical and Computer Engineering. University of Arizona, Tucson, AZ, USA.

[8] Mathworks. 2007. http://www.mathworks.com.

[9] Reenskaug, T. M. H. 1978. The Model-View-Controller (MVC). http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html.

[10] Sarjoughian, H.S., Y. Chen, K. Burger. 2008. A Component-based Visual Simulator for MIPS32 Processors, Frontiers in Education, TIA1-TIA6, October, Saratoga Spring, NY, USA.

[11] Sarjoughian, H. S. and V. Elamvazhuthi. 2009. CoSMoS: A Visual Environment for Component-based Modeling, Experimental Design, and Simulation, Proceedings of International Conference on Simulation Tools and Techniques, Rome, Italy.

[12] Sarjoughian, H., S. Kim, M. Ramaswamy, S. Yau. 2008. An SOA-DEVS Modeling Framework for Service-Oriented Software System Simulation, Proceedings of the Winter Simulation Conference, 845-853, December, Miami, FL, USA.

[13] Sarjoughian, H. S. and R. Singh. 2004. Building Simulation Modeling Environments Using Systems Theory and Soft-ware Architecture Principles. Proceedings of the Advanced Simulation Technology Conference, 235-240, Washington DC, USA.

[14] Wainer, G. 2002. CD++: A Toolkit to Develop DEVS Models, Software - Practice and Experience, 32: 1261-1306.

[15] Yau, S. S., N. Ye, H. S. Sarjoughian and D. Huang, 2008. Developing Service-based Software Systems with QoS Monitoring and Adaptation. Proceedings of the 12th IEEE Int'l Workshop on Future Trends of Distributed Computing Systems, 74-80, October, Kunming, China.

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