A Contextualized Web-Based DEVS Tutorial System

Zhibo Wang, Xiaolin Hu, Inthira Srivrunyoo
Department of Computer Science
Georgia State University
Atlanta, GA, USA 30303

Abstract
DEVS-based modeling and simulation have been applied to many different fields and used by researchers from all over the world. Much progress has been made in developing DEVS-based modeling and simulation environments. However, less work exists in developing systems that help beginners to learn DEVS models and DEVS-based simulation. This paper presents an interactive web-based tutorial system for learning DEVS models. The tutorial system uses a traffic light application as a learning context to help users engage in the learning process. In this paper, we present the models and the user interface of this tutorial system, and a stepwise learning process based on the developed system for supporting incremental learning of the DEVS concepts.

Keyword Contextualized, Web-based, DEVS education, Tutorial

1. INTRODUCTION

The Discrete Event Systems Specification (DEVS) is a timed, modular and hierarchical formalism which provides a means of specifying a mathematical object called a system [1]. It has been applied to many different fields and used by researchers from all over the world. Much progress has been made in developing DEVS-based modeling and simulation environments. However, less work exists in developing systems that help users to learn DEVS models and DEVS-based simulation. It is known that DEVS has a steep learning curve for beginners, especially in the initial stages of the learning. Traditional learning methods of DEVS include lectures and conference tutorials. While these methods are effective in an enforced learning environment such as a lecture, their usage and influence are limited by the presence of an instructor and the limitation of a physical environment. Web-based learning has the potential to influence a wider range of learners through the Internet. Current approaches of web-based learning of DEVS models mainly include online papers, online PPT slides, and online introduction of different types of DEVS applications (See e.g, ACIMS' website: www.acims.arizona.edu). Without an instructor to guide through the learning process, these approaches are ineffective for beginners.

In this paper, we present an effort to develop an interactive web-based tutorial system for learning DEVS models. The tutorial system uses a traffic light application as a learning context to help engaging users in the learning process. This approach of contextualized learning is useful for users, especially new beginners, to become interested in the learning subject and to better understand unfamiliar concepts. The developed tutorial system is web-based and interactive. This provides a convenient and interactive environment so a wider range of users can access the system and play with the system online in learning the DEVS models. Based on this system, we also developed a stepwise learning process that guides users to learn the various elements of DEVS modeling in an incremental manner. We note that the contribution of this work is not on developing new theory, methods, or applications of DEVS. Instead, it is on engineering and developing a contextualized web-based system that can be useful for beginners to learn DEVS modeling concepts.

Contextualized learning is becoming a popular learning paradigm in recent years. Many systems have been developed that put a learning process within an application context. For example, the Alice software programming tutorial [2] is a popular and representative contextualized tutorial system that allows students to learn traditional programming concepts such as "object", "method" and "functions". It addresses both the mechanical and sociological barriers that prevent many students from successfully learning to develop computer programs. An important type of contextualized learning is game-based learning, which uses computer games to engage learners in a learning activity through a storyline. For example, SimSE [3] is a computer-based simulation environment for teaching the software engineering concepts using a "virtual" software engineering process in a fully graphical, interactive, and fun game-type of setting. Based on the contextualized learning paradigm, our system allows users to learn DEVS models using a traffic light application context.

Developing materials and systems for learning DEVS models have been addressed by several lines of work in the M&S community. The ACIMS lab did tremendous work in

developing and popularizing the DEVS modeling and simulation framework. It published many materials to introduce the DEVS component-based modeling and simulation (see [3] for an example). More recently, a web-enable DEVSJAVA simulation environment called DEVS-Suite [4] is developed. DEVS-Suite is a new generation of DEVS simulator which combines the capabilities of DEVSJAVA and DEVS Tracking Environment. It supports the capabilities for monitoring simulation dynamics as time-based trajectories and automated data collection, and thus makes it easier for users to develop and to learn the DEVS models. Many other researchers also made efforts in helping learning DEVS-based modeling and simulation. These include, for example, the many conference tutorials given by various researchers, and the materials shown on the DEVS Standardization Group website (http://www.sce.carleton.ca/faculty/wainer/standard/)) [5]. The DEVS tutorial system presented in this paper differs from previous works by developing an online interactive system to support DEVS learning in a contextualized environment.

The remainder of this paper is organized as follows. Section 2 reviews the background of DEVS. Section 3 introduces the traffic light systems based on Atomic model and coupled model. Section 4 indicates the implementation of the DEVS tutorial system; Section 5 provides an example to guide users to build a coupled DEVS model. Section 6 concludes this work.

2. DEVS BACKGROUND

Since our system is about learning DEVS models, it is necessary to review the basic DEVS concepts and the DEVS formalism in this section. At the end of this section we also describe some of the major developments in DEVS-based modeling and simulation. We note that the first part of this section follows closely from an earlier paper that introduces DEVS modeling and simulation [6].

The DEVS (Discrete Event System Specification) formalism [1] is derived from mathematical generic dynamic systems theory and has been applied to both continuous and discrete phenomena. It provides a formal modeling and simulation (M&S) framework with well-defined concepts of coupling of components, hierarchical, modular model construction, support for discrete event approximation of continuous systems and an object-oriented substrate supporting repository reuse. There are two types of DEVS models: Atomic model and Coupled model. An Atomic model is a basic component which has input ports, output ports, states, internal, external and confluent transition functions, and output function. A coupled model is composed from multiple atomic models or coupled models. By coupling one model's output port to another model's input port, a message can flow from one model to another model. The "closed under coupling" property [1] of DEVS makes it possible to treat a coupled model as an atomic model, and thus gives rise to hierarchical construction of DEVS models.

A DEVS atomic model includes a time base, inputs, states, and outputs, and state transition functions for determining next states and an output function based on current state. An atomic model of a standard DEVS is a structure:

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

where,

X : set of external input events;

S : set of sequential states;

Y : set of outputs;

δ_{int}: $S \to S$: internal transition function

δ_{ext} : $Q \times X^b \to S$: external transition function

δ_{con}: $Q \times X^b \to S$: confluent transition function

X^b is a set of bags over elements in X,

$\lambda : S \to Y^b$: output function generating external events at the output;

$ta : S \to R^+_{0,\infty}$: time advance function;

$Q = \{ (s,e) \mid s \in S, 0 \le e \le ta(s) \}$ is the set of total states where e is the elapsed time since last state transition.

An *atomic model* template captures the following information:

- the set of input ports through which external events are received
- the set of output ports through which external events are sent
- the set of state variables and parameters
- the time advance function which controls the timing of internal transitions
- the internal transition function which specifies to which next state the system will transit after the time given by the time advance function has elapsed
- the external transition function which specifies how the system changes state when an input is received. The next state is computed on the basis of the present state, the input port and value of the external event, and the time that has elapsed in the current state.
- the confluent transition function which decides the next state in cases of collision between internal and external events.
- the output function which generates an external output just before an internal transition takes place.

Atomic models may be coupled in the DEVS formalism to form a *coupled model*. A coupled model specifies how to couple (connect) several component models together to form a new model. Two major activities involved in *coupled* models are specifying its component models and defining the couplings which create the desired communication networks. A *coupled* model is defined as follows:

DN = <X, Y, D, {M_i}, {I_i}, {$Z_{i,j}$}>
where,
X : set of external input events;
Y : a set of outputs;
D : a set of components names;
for each i in D,
M_i is a component model
I_i is the set of influences for i
for each j in I_i,
$Z_{i,j}$ *is* the i-to-j output translation function

A *coupled model* template captures the following information:

- the set of components
- for each component, its influences
- the set of input ports through which external events are received
- the set of output ports through which external events are sent
- the coupling specification consisting of:
  - the external input coupling (EIC) connects the input ports of the coupled to one or more of the input ports of the components
  - the external output coupling (EOC) connects the output ports of the components to one or more of the output ports of the *coupled* model
  - internal coupling (IC) connects output ports of components to input ports of other components

The classic DEVS formalism presented above has been extended in different ways to support modeling various types of systems. Among them the RTDEVS [7] formalism was developed for supporting real-time system specification. The Cell-DEVS formalism [8,9] was developed to support discrete event cell space modeling (see [10] for an application of Cell-DEVS to the forest fire spread simulation). To model systems with dynamics structure capabilities, the dynamic structure DEVS formalism was developed and studied (see, e.g., [11,12,13]). The DEVS formalism was also extended to support continuous system modeling based on the quantization concepts [14,15]. More recently, the Finite & Deterministic DEVS (FD-DEVS) was proposed for supporting formal verification using DEVS models [16]. The tutorial system presented in this paper deals with the classic DEVS and focuses on the modeling aspect of DEVS atomic and coupled models.

3 The Traffic Light System as a Learning Context

In the developed tutorial system (available online at: http://www.cs.gsu.edu/DEVSTutorial/), we exemplify the DEVS atomic and coupled model formalism using a simple traffic light controller system. The traffic system we consider here is a simplified version of real-world traffic. For example, we do not allow cars to turn, and all cars on the same street move at the same speed. Figure 1 shows a screen shot of the traffic system window of the tutorial (detailed descriptions of this figure will be given in section 3.1). In this traffic system, there are two one-way streets that intersect with each other: a main street (the horizontal street) and a small street (the vertical street). Cars on the main street run from east to west; and cars on the small street run from north to south.

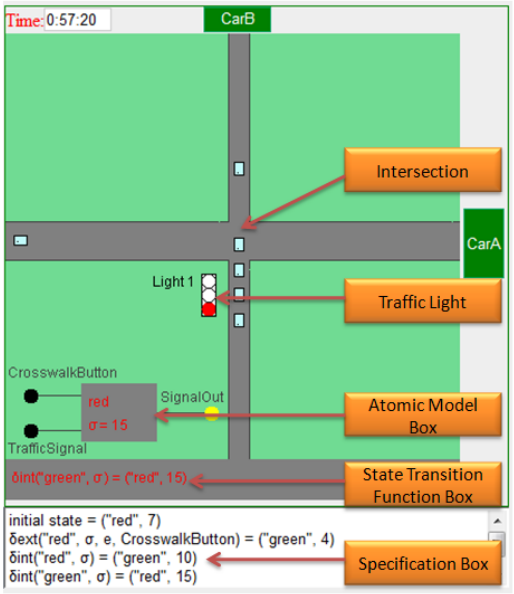

Figure 1: The Traffic System as a Learning Context

The DEVS model that users can define and interact with is for controlling the traffic light of the intersection. We limit the traffic light to have three signal states: *red*, *green*, and *flashing red*. Red means "stop" for the main street and "go" for the small street; green means "go" for the main street and "stop" for the small street; flashing red means cars on both streets should stop first before moving ahead. The tutorial system provides an interface for users to define the behavior of the traffic light (following the DEVS formalism), and then see how it works right away through the traffic system window as shown in Figure 1. Note that as a tutorial system, it is our intention to focus only on the traffic

light (with relative simple behaviors) through which users can learn the basic elements of DEVS modeling.

In DEVS modeling, one must specify 1) atomic models from which larger ones are built, and 2) how these models are coupled together to form coupled models. Corresponding to these two types of models, two main tutorial pages are developed: one for atomic model tutorial, the other for coupled model tutorial. The atomic model tutorial page allows users to learn and experiment with the dynamical behaviors of an atomic model. These include specifying the initialize function, external transition function, internal transition function, confluent transition function, and output function of the atomic model. The coupled model tutorial page allows users to learn and experiment with adding components and coupling components together to form a system. For example, users can define an intelligent traffic light control system by coupling traffic lights together so they can send signal to each other. Furthermore, we developed a stepwise learning process that guides users to learn DEVS modeling in an incremental manner. Each step of this process focuses on a single modeling element, such as internal transition function or external transition function of DEVS atomic model, and provides discussions and exercises related to that element.

Below we describe the atomic model tutorial page, and coupled model tutorial page, and the stepwise learning process in detail.

3.1 Atomic Model Tutorial Page

The atomic model tutorial paper allows users to learn the DEVS atomic model based on the traffic system context. It allows users to define the behavior of the traffic light, watch how it works according to the defined behavior, interact with the model (e.g., injecting an external input to the model), and check out the DEVSJAVA code of the defined model.

Figure 2 shows the traffic light model that a user can specify and interact. As can be seen, the traffic light model has two input ports: *CrosswalkButton* and *TrafficSignal* and one output port: *SignalOut*. The traffic light has three states: *red*, *green* and *flashing red*. These different states of the traffic light indicate different control signals for the cars on the streets. When the model transitions from one state to another, the cars on the street will respond to the change of signal accordingly. A display of this model (named as the Atomic model Box in Figure 1) is also shown in the traffic system window. The atomic model box always displays the current state of the model and the remaining sigma of that state. The remaining sigma is continuously updated every second, that is, it decreases by 1 in every second. This allows users to easily see how the time elapses, and thus help them understand the concept of elapse time, which is critical in DEVS atomic model. At any time, a user can inject an external input to the atomic model by clicking one of the input ports in the Atomic model box. Also, whenever the model generates an output, the output port of the Atomic model box will change its color and flashes several times to visualize the output.

Figure 2: The Traffic Light Atomic Model and State

The dynamic behavior of the traffic light is defined by the user following the DEVS specification. This is accomplished through a popup window when the user clicks the "define behavior" button in the atomic model tutorial page. Figure 3 shows the interface for specifying the traffic light model's behavior. Through this interface, a user can define the initialization function, the external transition function, internal transition function, confluent transition function, and the output function. For example, in the external transition function form, a user can select an input (e.g., CrosswalkButton, meaning an input from the CrosswalkButton input port), a current state (e.g., red), a next state (e.g., green), and type a sigma value (e.g., 10), and then click "Add" to add that behavior as part of the external transition function of the traffic light model. Then based on that specification, if the traffic light is in the red state and a user clicks the CrosswalkButton, the traffic light will transition to the green state with sigma of 10 seconds. Other behaviors of the traffic light model can be defined in a similar way.

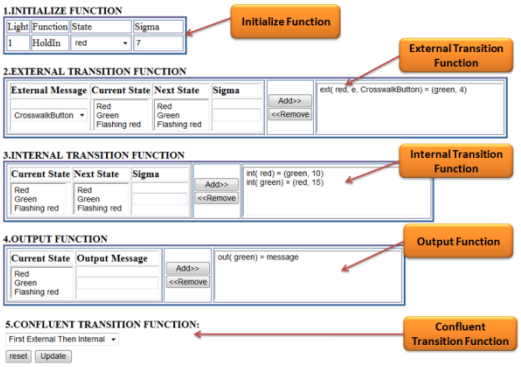

Figure 3: The traffic light specification page

After the user finishes defining the behavior, the traffic system window (Figure 1) will carries out the defined behavior and visualizes its effect.

This includes displays the traffic light with the right color (red, green, or flashing red), and move or stop the cars on the two streets according the traffic light signal. Furthermore, the traffic system window has two more boxes as shown in Figure 1: a State Transition Function Box and a Specification Box, to help users to understand how the traffic light atomic model works. The State Transition Function Box shows the more recent state transition of the model And the Specification Box lists all the specifications defined by the user. They help users to understand what was happened and what will be happening next, and observe the changes of light states in real time. Finally, the atomic model tutorial page also provides a function for users to see the DEVSJAVA code of the defined model.

3.2 Coupled Model Tutorial Page

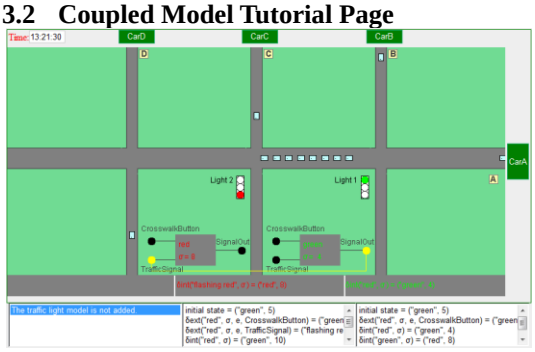

Figure 4 Traffic light coupled model

The coupled model tutorial page allows users to define a DEVS coupled model consisting of up to three traffic light atomic models (corresponding to the three intersections) as shown in Figure 4. It allows users to add traffic light models to the coupled model system, define model behaviors, and add couplings between the traffic light models. To visualizing the effect of adding components into the coupled model, initially the system has no traffic light and all cars moves ahead without stopping. A user can select which traffic lights (up to three) to add. Once a traffic light is added, it will be displayed on the applet window as shown in Figure 4, and begins to controls the traffic on the streets. A user can also remove already added traffic lights. Each traffic light is an atomic model and is the same as described in the previous section. For a particular traffic light, a user can define its behavior and interact with it just as described before. Meanwhile, a user can add couplings among the added traffic lights and thus allow them to influence each other. When a coupling is added, a link will be dynamically added by connecting the source model's output port with the destination models' input port as shown in Figure 5. We note that a user can also remove existing added couplings. Figure 5 shows an example where two traffic lights are added into the system and two couplings are added between the two lights. To visualize the effect of message passing between traffic lights, whenever a traffic light generates an output through its output port, all the coupling lines and the input ports coupled to that output port will flash several times. This makes it very easy for users to see the effect of message passing between models. Similar as in the atomic model tutorial page, the coupled model tutorial page also has the State Transition Function Boxes and the Specification Boxes to show the specifications of the defined models.

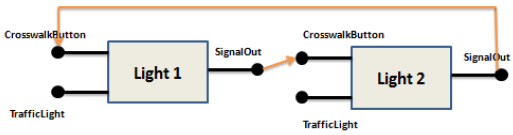

Figure 5 A Coupled model Example

3.3 The stepwise learning process

Based on the functionalities of the atomic mode tutorial and the coupled model tutorial, we developed a stepwise learning process that guides users to learn the various elements of DEVS modeling in an incremental manner. The basic idea is that each step of this process focuses only on one element of DEVS modeling, and a user can stop, move backward, and move forward, or jump to a specific step during the learning process. We think this is useful especially for beginners to learn the various elements of DEVS modeling. The major steps of this learning process for the atomic model and the coupled model are given below.

- Step 1: Set the model's initial state – learn the initialize function. In this step, a user defines an initial state and its sigma through the initialize function, and then sees how the sigma elapses over time. Note that whenever the sigma expires and the model has no associated internal transition defined by the user, the model will atomically come to an "undefined" state and will be displayed in gray color. This leads naturally to the next step for specifying the internal transition function so the model can transition to a new state. Two special cases in this step that users can exercise are setting the sigma to be 0 or to be infinity.
- Step 2: Define state transition when time expires – learn the internal transition function. In this step, a user can define the internal transition of the model (need to set an initial state first) and see how the model will transition to a new state when time expires.
- Step 3: respond to external input – learn the external transition function. In this step, a user

can specify external transition functions of the model, and then click the input port to see how the model responds. This step also introduces concepts such as input ports, external events.

- Step 4: Generate an output – learn the output function. In this step, a user can specify an output for the model, and then watch how an output will be generated (the output port will flash several times when there is an output) when the sigma expires.
- Step 5: Handle the confluent situation – learn the confluent transition function. This step will instruct users to specify the confluent transition function, and then click the input ports when sigma expires. This will trigger the confluent transition function and allow users to see the effect.
- Step 6: All together – learn how the atomic mode works. In this step, a user can define a full-scale atomic model and interact with it. Some pre-defined sample models will also be provided for users to load and play with.

The stepwise process for learning the coupled model is straightforward. It includes the following step:

- Step 1: Add components. In this step, a user can add selected lights (up to three) to the coupled model system, and once a traffic light controller is added, it displays on the applet window and starts to control the traffic of the street. Existing traffic light can be removed by the "Remove" button.
- Step 2: Define model behavior. In this step, user can define the selected traffic light controllers as described in section 3.1. Each traffic light has individual behaviors based on a user's specification.
- Step 3: Add couplings between models. In this step, users can couple the selected lights by connection one's output port with others' input port. Thus, models can not only own their individual behaviors, but also have the ability to influent others in the coupled model system.
- Step 4: Closed under coupling – support hierarchical model construction. In this step, we explain one of the most important features of DEVS model, the closed under coupling feature that make it possible for hierarchical model construction. We note that currently our tutorial system does not support user to interact with the system to learn this concept.

4 Implementation of the Tutorial System

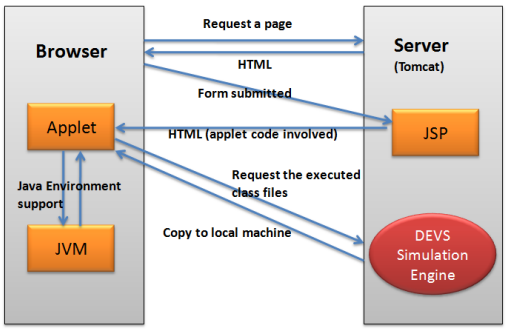

Figure 6 Structure diagram of the tutorial system

This contextualized web-based tutorial system is based on the B/S (Browser / Server) model as shown in figure 6. For such a design way, any user with only a local computer with Browsers (IE, Firefox .etc.) supported by Java environment and Internet access can study DEVS knowledge through our system anywhere. Tomcat as a web-server software is engaged and served for the simulation applications. In the Client side, applet is used for the simulation display and to support it, Java Virtual Machine (JVM) needs to be installed in the local machine.

When connection is established between users' local machine and the server, users send requests and the server responses users' requests and return a HTML file. Once users submit the form (e.g. light specification page) to the server, JSP will load it and also return HTML file but the applet codes involved. The browser in the local machine reads the returned HTML file and finds the applet codes. Then it loads the applet plug-in and starts the JVM in the local machine. The applet concurrently checks the path in the returned HTML file and according the path (told in the HTML file) request the requisite class files stored in the simulation engine. Then the server makes a copy to the local machine. When clients complete the downloading work, the simulation starts running locally. If the users submit another form to the server, server will return it in a new HTML file to the client and the browser will read the loaded content in the applet. The action that local machine copies the class files from Server only for the first time.

5 An Illustrative Example

This section provides an illustrate example to demonstrate how to set up a coupled model in a step-wise fashion.

Step1: Add components

In this example, two lights are employed in our coupled model. Only select Light1 and Light2 and then click the add to add them into the system. And now, two traffic light controllers should displays in the display window.

Step 2: Define the selected light controller

In this step user specified the traffic lights' specifications follow section 3.1. Below is the DEVS formalism generated by users' definitions.

LIGHT1:

- Initialization Function
  - Initial state is in green for 5 seconds.
- External Transition Function
  - Current state is in red, and next state is in green. Sigma time is 6 seconds. External message is "CrosswalkButton".
- Internal Transition Function
  - Current state is in red, and next state is in green for 7 seconds.
  - Current state is in green, and next state is in flashing red for 5 seconds.
  - Current state is in flashing red, and next state is in red for 6 seconds.
- Output Function
  - When the phase is in red, a message is sent.
- Confluent Function
  - System will schedule the executed sequence as First External Then Internal

Below is the DEVS formalism for LIGHT1.

initial state = ("green", 5)

δ_{int} ("red", σ) = ("green", 7)

δ_{int} (" green", σ) = ("flashing red", 5)

δ_{int} (" flashing red", σ) = ("red", 6)

δ_{ext} ("red", σ, e, "CrosswalkButton") = ("green", 6)

δ_{con} (s,ta(s),x) = $\delta_{ext}(\delta_{int}(s),0,x)$

λ ("red", σ) = "message1"

ta (phase, σ) = σ

LIGHT2:

Under same setting methods with LIGHT1, formalisms generated for LIGHT2 is as follows.

initial state = ("red", 5)

δ_{int} ("red", σ) = ("green", 8)

δ_{int} (" green", σ) = ("red", 3)

δ_{ext} ("red", σ, e, "CrosswalkButton") = ("green", 4)

δ_{con} (s,ta(s),x) = $\delta_{ext}(\delta_{int}(s),0,x)$

λ ("red", σ) = "message2"

ta (phase, σ) = σ

Step 3: Establish the couplings

Below are two coupling formalisms between Light1 and Light 2.

- addCoulping(LIGHT1, SignalOut, LIGHT2, CrosswalkButton)
- addCoupling(LIGHT2, SignalOut, LIGHT1, TrafficSignal)

Based on three steps above, we visualize the result as follows:

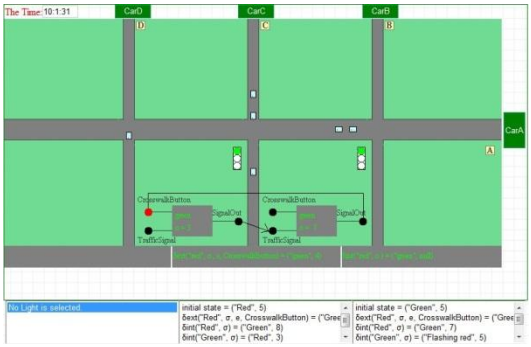

Figure 7 Display of two light controllers coupled model

6 Conclusion

This paper presents an interactive web-based tutorial system for learning DEVS models. The tutorial system uses a traffic light application as a learning context to help users engage in the learning process. We present the models and the user interface of this tutorial system, and the steps of the learning process. Further extensions of this work include getting feedback from users for improving the user interface, and integrating with other systems such as the DEVS Suite system for adding more functionality into the system.

Reference:

[1] B. P. Zeigler, T. G. Kim, H. Praehofer, "Theory of Modeling and Simulation", 2nd ed. New York: Academic, 2000.

[2] The Alice software programming Tutorial, Pittsburgh, Pennsylvania: Carnegie Mellon University, 2007. [Online]. Available: www.alice.org

[3] SimSE, Irvine, California: University of California Irvine, 2007. [Online]. Available: www.ics.uci.edu/~emilyo/SimSE

[4] Arizona Center of Integrative Modeling and Simulation, DEVS-Suite Simulator. Tucson, Arizona: Univ. Arizona, 2009 [Online]. Available: devs-suitesim.sourceforge.net

[5] DEVS Standardization Group, [Online]. Available: cell-devs.sce.carleton.ca/devsgroup

[6] B. P. Zeigler, H. S. Sarjoughian, "DEVS Component-based M&S Framework: An Introduction", AIS 2002.

[7] J.S. Hong, and T.G. Kim, "Real-time

Discrete Event System Specification Formalism for Seamless Real-time Software Development," Discrete Event Dynamic Systems: Theory and Applications, vol. 7, pp.355-375, 1997.

[8] Wainer, G. and N. Giambiasi. 2002. N-Dimensional Cell-DEVS. Discrete Events Systems: Theory and Applications 12(1), 135–157.

[9] Wainer, G. A. Modeling and simulation of complex systems with Cell-DEVS. 2004. Proceedings of the 36th conference on winter simulation 1, 45-56.

[10] Muzy, A., and Innocenti E. and Aiello, A., and Santucci, J.F., and Wainer, G. Specification of Discrete Event Models for Fire Spreading, SIMULATION, Vol. 81, No. 2, 103-117 (2005)

[11] Barros, F.J. 1997. Modeling Formalisms for Dynamic Structure Systems. ACM Transactions on Modeling and Computer Simulation 7(4), 501-515.

[12] Uhrmacher, A.M. 2001. Dynamic Structures in Modeling and Simulation – A Reflective Approach. ACM Transactions on Modeling and Simulation 11(2), 206-232.

[13] X. Hu, B. P. Zeigler, and S. Mittal, Variable Structure in DEVS Component-Based Modeling and Simulation, SIMULATION, Vol. 81, No. 2, pp. 91-102, 2005

[14] Kofman, E. and Junco, S. (2001)."Quantized State Systems. A DEVS Approach for Continuous Systems Simulation". Transactions of SCS. 18(3). pp.123-132

[15] Nutaro, J., Zeigler, B.P., "On the Stability and Performance of Discrete Event Methods for Simulating Continuous Systems ", Journal of Computational Physics, Vol 227, Issue 1, 10 November, 2007

[16] Hwang, M. H.; Zeigler, B. P., Reachability Graph of Finite and Deterministic DEVS Networks, IEEE Transactions on Automation Science and Engineering, Page(s): 468-478, Volume 6, Issue 3, 2009

[17] I. Srivrunyoo, “A Contextualized Web-based Learning Environment for DEVS Models”, Thesis of Master Degree, 2007.