From UML State Charts to DEVS State Machines using XML

José L. Risco-Martín[a] Saurabh Mittal[b] Bernard P. Zeigler[b]
Jesús M. de la Cruz[a]

[a] *Arquitectura de Computadores y Automática*
Universidad Complutense de Madrid
Madrid, Spain

[b] *Arizona Center for Integrative M&S*
ECE Department, University of Arizona
Tucson, AZ 85721

Abstract

Modeling and Simulation (M&S) for system design and prototyping is practiced today both in the industry and academia. Modeling and Simulation are two different areas altogether and have specific objectives. However, most of the times these two separate areas are taken together. The developed code is tightly woven around and model and the underlying simulator that executes it. This constraints both the development of model as well as the simulation engine and impacts scalability of the developed code. Modeling is more art than science and the level of abstraction plays a crucial role in model's performance. Furthermore, it is difficult to develop a model because it needs both domain knowledge and simulation techniques, which also requires communication among users and developers. UML is the preferred method used today in industry, but it lacks simulation power. Discrete Event Specification (DEVS) based modeling that separates the model and the simulator, provides a cleaner methodology to develop models. However DEVS today is used by engineers who understand discrete event modeling at a much detailed level and are able to translate requirements to DEVS modeling code. It is oblivious to industry practitioners that use UML due to its graphical expressive power. There have been earlier efforts to integrate UML and DEVS but they haven't succeeded in providing a transformation mechanism due to inherent differences in these two modeling paradigms. DEVS state machines are more expressive than UML state machines, require more information, specifically finite timeouts for each state. This paper presents an integrated approach towards using UML state machines transformed as DEVS models. The transformation mechanism is available as an upcoming standard, State Chart XML (SCXML) that provides a generic execution environment based on CCXML and Harel State tables. We will illustrate the transformation by taking a UML state machine and augmenting it with information during the process using SCXML to make it DEVS capable. Further, we will also show that the obtained DEVS models belong to a specific class of DEVS models called Finite Deterministic DEVS (FD-DEVS) that is available as a W3C XML schema.

Keywords: DEVS, FD-DEVS, UML-based Modeling, XML, Model Transformation

1 Introduction

It is difficult to develop a simulation model in the early phase of system development since it requires a high level knowledge of modeling techniques, the system domain and the model execution paradigms. It is no longer sufficient to study the diverse components separately, using the specific formalisms these components were modelled in. Rather, it is necessary to answer questions about properties (most notably behaviour) of the whole system [26]. It requires intensive cooperation among

domain experts and modeling experts as both demand totally different expertise. We need a practical and efficient way of applying Modeling and Simulation (M&S) to the development of system under design in early phases of system development. The process starts with elicitation of system requirements and translating them into executable modeling code. The Department of Defense (DoD) has strongly recommended to apply M&S techniques to validate the requirements during the system development [6].

Unified Modeling Language (UML) is one of the preferred means of communication between the domain experts and modeling experts. UML is very powerful in terms of its graphical representation but diminishes in quality when it comes to execution of the UML model. Executable UML (xUML) [16] is a working draft and is not operational in its current state today. We propose a set of transformations to use such graphic models into an executable simulation model. Discrete Event Specification (DEVS) Formalism provides a system theoretic foundation to execute models using DEVS simulation protocol. DEVS models are inherently hierarchical in nature and consists of two class of models i.e. atomic and coupled. Atomic modes contain the state machine of a component and coupled models, that acts as a collection box, contains many atomic and coupled models leading to hierarchical design. Many different paradigms like System Entity Structure (SES) and DEVS hierarchical modeling can very well be used to interface with UML structure diagrams to provide an executable model. However, the problem comes at the level of atomic models that contains DEVS finite state machines. We would like to emphasize that this work is based on the behavior of such models, since the model structure is well studied and has multiple solutions, such as block diagrams, UML class diagrams, etc. We will be using the State Chart XML (SCXML) [24] methodology as an intermediate step in development of DEVS atomic models from UML state machines.

The proposed UML-based M&S method takes three steps. First, we synthesize the UML SM and generate its corresponding SCXML model. Second, we transform the SCXML file to a finite deterministic DEVS State Machine (FD-DEVS SM) model [17]. Finally, we transform FD-DEVS XML model into a simulation model which can be executed by the DEVS simulation engine [1], [18], [19].

There are many UML Computer Aided Software Engineering (UML CASE) tools such as IBM Rational Rose, Poseidon, etc, and all of them provide simulation functionalities, tracing states change or signal invocation. All these tools have proprietary simulation engines. Further, most of the simulation engines are not extensible towards performance related requirements. For example, a typical Opnet model takes days to complete an execution. A parallel simulation engine would be needed but due to the proprietary nature, such extensions are not possible. For our purpose, we need an open-source specialized simulation engine which can take over the details of the simulation process (event management, simulation time management, etc.) and provides extensibility. Consequently, in addition to the benefits DEVS formalism [27] provides with respect to system theoretic foundation, we use the DEVSJAVA version 3.0 [1] to develop our case.

The paper is organized as follows. Section 2 introduces related work. In Section 3, we introduce the underlying technologies used, DEVS, DEVSJAVA, UML SM,

SCXML SM and FD-DEVS SM. In Section 4, we describe the method proposed. Section 5 demonstrates the application of this approach to the development of an atomic model, JTAC, which is taken from the complete example of Joint Close Air Support (JCAS [17]). Section 6 summarizes the main results of this paper and gives plan for future work.

2 Related Work

In our earlier effort [23], we proposed a W3C XML schema for DEVS coupled models that enables the user to provide the system component structure in hierarchical DEVS notation. This describes the structure and a limited behavior of any DEVS model. However, it is not a graphic solution, because the model must be written in XML to be transformed to the simulation engine.

In [13] and [18], another meta-model is defined to represent a DEVS system by means of XML. In both cases, JavaML [14] is used to describe the behavior of an atomic model. This solution is good for transforming platform specific models (PSMs) to platform independent models (PIMs), but it is not a graphic solution for modeling such systems.

There are other approaches to represent DEVS models, such as the Scalable Entity Structure Modeler with Complexity Measures (SESM/CM) [2]. SESM/CM is suitable for developing component-based hierarchical models. It offers a basis for modeling behavioral aspect of atomic models by providing the structural specification and storage of the model using XML. But, this approach is very close to the simulation expert instead of the domain expert. Further, there is no means to develop the atomic state machine or behavioral model explicitely. This is largely a structure tool and needs further work to represent atomic models using XML.

In the UML-based M&S domain, [5] propose a UML-based M&S method, making use of UML sequence diagrams to define the system behavior. In [9], eight steps are introduced in order to make DEVS models using UML, but in this case they need many human decisions to transform the model. Zinoviev [28] proposes a mapping of DEVS models onto UML state and component diagrams. In [3], they develop a methodology to transform hierarchical statecharts into DEVS, but we are making use of SCXML, which fixes the problem about the definition of time of events. In [7], the formal transformation of timed input/output automata into a DEVS simulation model is provided. However, the timed automata is hard to communicate and develop the simulation model. Extending the domain to the Model Driven Architecture (MDA) paradigm, Tolk and Muguira show how the complementary ideas and methods of the High Level Architecture (HLA) and DEVS can be merged into a well-defined M&S application domain within the MDA framework [25].

3 Background

3.1 UML State Machine Diagrams

State diagrams are used to graphically represent finite state machines. State transition tables are another possible representation. There are many forms of state

diagrams that differ slightly and have different semantics, such as Moore machine, Mealy machine, Harel statecharts, etc. For a more detailed description, see [10].

UML is officially defined at the Object Management Group (OMG) [22] by the UML metamodel, a Meta-Object Facility metamodel (MOF). Like other MOF-based specifications, the UML metamodel and UML models may be serialized in XMI. UML was designed to specify, visualize, construct, and document software-intensive systems. UML is not restricted to modeling software. It is also used for business process modeling, systems engineering modeling and representing organizational structures. The UML SM diagram is essentially a Harel statechart with standardized notation that can describe the behavior of computer programs, business processes, etc.

State diagrams may be used to represent the behavior of DEVS atomic models. Furthermore, state diagrams can be represented graphically by means of UML State Diagrams. In the field of software engineering, UML is a standardized specification language for object modeling. UML is a general-purpose modeling language that includes a graphical notation used to create an abstract model of a system, referred to as a UML model [9].

3.2 SCXML

The State Chart eXtensible Markup Language (SCXML) provides a generic state-machine based execution environment based on Call Control eXtensible Markup Language (CCXML) [4] and Harel State Tables [8]. The last call working draft of the specification was released by the W3C in February 2007 [24].

SCXML is a general-purpose event-based state machine language and can be used in many ways, including high-level dialog language, state machine framework for a future version of CCXML, etc. Since UML State Diagrams can be transformed to XMI or SCXML, our purpose is to use SCXML because its syntax is more comprehensible and easier to manage from an external application than XMI.

There are some ways to transform a given UML SM into a SCXML SM. We have used a plug-in for IBM Rational Software Architect (IBM RSA) [12].

3.3 DEVS and DEVSJAVA

DEVS formalism consists of models, the simulator and the Experimental Frame. It allows representing any system by three sets and four functions: Input Set, Output Set, State Set, External Transition Function, Internal Transition Function, Output Function, and Time Advanced Function. DEVS formalism provides the framework for information modeling which gives several advantages to analyze and design complex systems [27].

DEVS model processes an input event based on its state and condition, and it generates an output event and changes its state. Finally, it sets the time during which the model can stay in that state.

We will focus our attention to the specified two types of models i.e. atomic and coupled models. The atomic model is the irreducible model definition that specify the behavior for any modeled entity. The coupled model is the aggregation/composition of two or more atomic and coupled models connected by explicit

couplings.

The coupled model can itself be a part of component in a larger coupled model system giving rise to a hierarchical DEVS model construction. Detailed descriptions about DEVS Simulator, Experimental Frame and of both atomic and coupled models can be found in [27].

The DEVSJAVA [1] is a Java based DEVS simulation environment. It provides the advantages of Object Oriented framework such as encapsulation, inheritance, polymorphism, etc. DEVSJAVA manages the simulation time, coordinates event schedules, and provides a library for simulation, a graphical user interface to view the results, and other utilities.

3.4 DEVS State Machine

Saurabh Mittal has recently proposed a novel way to automate the DEVS state machine specification process [17]. In such approach, any state machine can be looked upon as the superposition of two behaviors. The first cycle is the default execution of the machine, wherein it receives no external inputs. These are the "internal" transitions of the component. The second behavior, which can spawn multiple cycles stems from the actions resulting from reception of various inputs in various states. These are the induced "external" transitions of the component. DEVS categorically separates these two behaviors in its formal δ_{int} and δ_{ext} specification respectively. Mittal also proposed a W3C XML Schema [19] which belongs to a sub-class of verifiable DEVS models called finite deterministic FD-DEVS [11], [20]. Consequently, a template based state requirement process is developed where the designer can specify these two behavior cycles. In addition, Mittal developed a framework to transform DEVS SM XML models to DEVSJAVA models.

In this effort, we developed a way to transform an SCXML SM into a DEVS SM, without using the DEVS template based finite state machine tool developed by Mittal.

4 From UML to DEVS

4.1 Overview

Modeling is an art of abstraction of real systems to generate the behavior by specifying a set of instructions, rules and equations to represent the structure and the behavior of the system. The structural elements of a system include the components of the system and their interactions (inputs, outputs, connections, etc). The behavioral elements include the sequence of interactions, the timing constraints of the interactions, and the operations of each component. Modeling provides the means of specifying the structure of a system, behavior of a system over time, and the mechanism for executing the instructions, rules, or equations.

We provide a new approach towards behavioral specifications, since the approach to represent the system component structure is easier to understand between the domain experts and modeling experts. Such approaches include UML Class Diagrams, block diagrams, etc. While in [21], Mittal provided a complete mapping of UML diagrams to correspong DEVS elements, [23] provides a Platform Indepen-

dent Model (PIM) of DEVS coupled systems. The current effort is continuation of these efforts. Behavioral models are developed using finite state machines or UML statecharts time-sequence diagrams that illustrate interactions between various components over time. We concentrate on taking the finite state machine using UML approach and developing the DEVS model from it.

The proposed UML-based M&S method takes three steps. First, we design the UML SM and generate a SCXML SM describing this UML SM. Second, we transform the SCXML file to a DEVS SM model. Finally, we transform DEVS SM model into a simulation model, which can be executed by the simulation engine [1], [18], [19].

Figure 1 shows the overview of the UML SM method. As we have mentioned before, the work developed here corresponds to the behavioral aspects of the system. The structural model could be developed in parallel with the behavior description. In Figure 1, gray boxes describe a possibility to model the system structure through DEVSML [18].

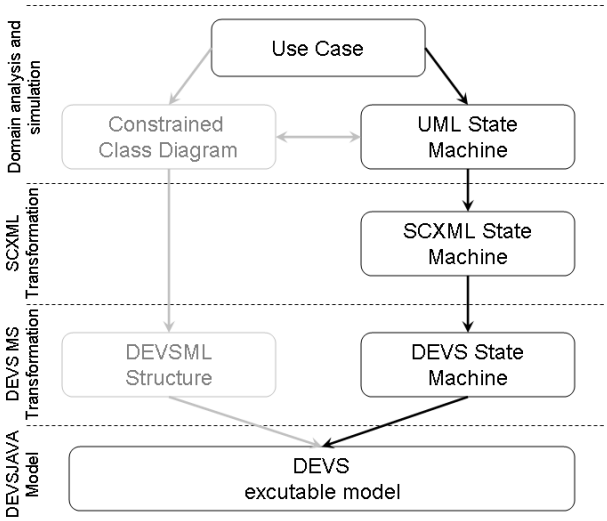

Fig. 1. Overview of the modelling process

4.2 The UML SM

The UML SM is defined as follows. We define a SM in the sense of a subset of the SCXML SM:

Definition: State Machine Our State Machine is described by a 5-tuple $SM =< S, s_0, \Sigma, \delta, S_e >$ where:

- S is a finite nonempty set of states,
- s_0 is the initial state,
- Σ is a finite set of events,
- $\delta : S \times \Sigma \rightarrow S$ is the transition function,
- $S_e : S \rightarrow P(\Sigma)\backslash 0$ is a function assigning to each state a set of delayed output events.

Each element of the 5-tuple has its corresponding SCXML tags. Table 1 shows all of these equivalences.

Table 1
Equivalences between SCXML and SM

SM	SCXML	Description
s_0	`initialstate='s0'`	`at document root`
S	`<state id='S'/>`	`node`
Σ		`set of events`
$\delta : S \times \Sigma \rightarrow S$	`<transition event='`Σ_i`' .../>` `<transition .../>`	`child of state` `child of state`
$S_e : S \rightarrow P(\Sigma) \backslash 0$	`<send event='`Σ_i`' delay='t'/>`	`child of transition`

States, time events, transitions and transitions triggered by events are easy to be modeled in UML SM. Time events are stated in SCXML through "send" tags. It is used to send events to external systems or to raise external events in the current SCXML session. The target of "send" is specified using the "target" and "targettype" attributes (usually empty for our purposes). These attributes control how the platform should dispatch the event to its final destination. The send tag has an attribute named "delay". The character string returned is interpreted as a time interval. The send tag will return immediately, but the event is not dispatched until the delay interval elapses. In order to represent timed events in the UML SM we are adding timed events in the transitions of the sate chart. Such events will include a "send" tag, which will define all the timed events.

The transformation from UML SM to SCXML is automatic; there are libraries for this purpose, such as .a plug-in for IBM RSA [12].

4.3 From SCXML SM to DEVS SM

SCXML allows the definition of timed events, which is a good advantage in our algorithm designed to transform a SCXML model to a DEVS SM model. Returning to our SM definition, the corresponding DEVS elements are illustrated in Table 2. Such transformation is done through XSLT.

Table 2
Rules to transform a SCXML model into a DEVS SM model

SM	SCXML	DEVS SM
s_0	`initialstate='s0'`	`Initial state`
S	`<state id='S'/>`	`State`
Σ		`Set of outputs`
$\delta : S \times \Sigma \rightarrow S$	`<transition event='`Σ_i`' .../>` `<transition .../>`	`External transition` `Internal transition`
$S_e : S \rightarrow P(\Sigma) \backslash 0$	`<send event='`Σ_i`' delay='t'/>`	`Output function`

5 Case Study

We applied the proposed approach to an entity named JTAC, which is taken from the complete example of Joint Close Air Support (JCAS). The Joint Close Air Support Model is expressed in plain English. It is a small example involving components

exchanging messages towards a common objective. The components of JCAS model are JTAC, UAV, CAOC, USMC Aircraft and AWACS. The complete JCAS example is provided in [17], where these requirement descriptions are presented using multiple formats, such as BPMN/BPEL notations, DEVS finite state machines and Message-based with restricted Natural Language Processing (NLP) format. The complete JCAS model is executed through each of the formats above leading to an executable DEVS coupled model.

In this case study, we will show that another format, namely UML, also provides a mechanism to develop state machines that can be transformed to DEVS component based M&S. We will take the example of component JTAC listed above and will walk through the UML-SCXML-DEVS process. The behavior of JTAC component needs to be extracted from the scenario descriptions, which is a manual process.

5.1 The UML SM for JTAC

The UML SM for JTAC is modeled using IBM RSA. Figure 2 shows the resulting diagram. There are simple transitions, for example "goWaitForAssignment", external transitions triggered by events (goProvideTAC, which is triggered by the "RequestTAC" event) and time events. A timed event happens in, for example, the "goProvideTAC" transition. In such transition there will be an output called "InitialAttack" after 10 units of time starting when the system enters in the "ProvideTAC" state.

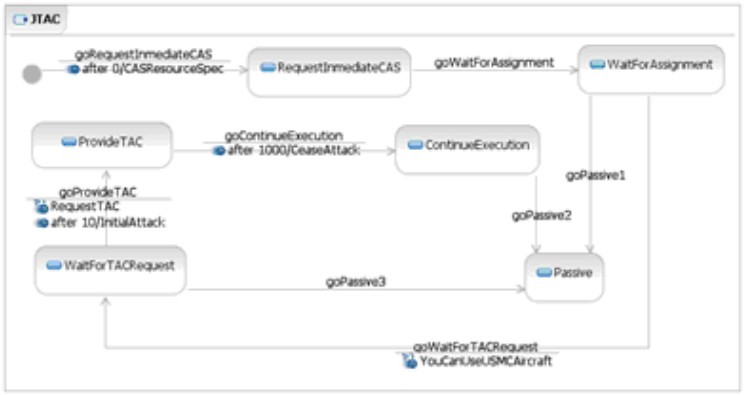

Fig. 2. UML SM diagram for JTAC

5.2 The SCXML SM for JTAC

The SCXML SM is automatedly generated from the SCXML plugin for IBM RSA. For more details on SCXML schema refer [24]. Figure 3 shows a little portion of the SCXML resulting model of component JTAC.

5.3 The DEVS SM model for JTAC

The DEVS SM model is obtained using our XSLT transformation on the SCXML SM file. Figure 4 shows a snippet of the resulting DEVS SM for component JTAC.

```
<?xml version=''1.0''?>
<scxml initialstate=''JTAC''>
 <state id=''JTAC''>
  ...
  <state id=''RequestInmediateCAS''>
   <transition target=''WaitForAssignment''/>
  </state>
  ...
  <state id=''ContinueExecution''>
   <onentry>
    <send event=''CeaseAttack'' delay=''1000ms''/>
   </onentry>
   <transition target=''Passive''/>
  </state>
  ...
 </state>
</scxml>
```

Fig. 3. SCXML SM

```
<?xml version=''1.0'' encoding=''UTF-8''?>
<statemachine name=''JTAC'' host=''localhost''>
 <deltint>
  ...
 </deltint>
 <deltext>
  <transitionsExt>
   <transitionExt>
    <incomingMsg>RequestTAC</incomingMsg>
    <transition>
     <startState>WaitForTACRequest</startState>
     <nextState>ProvideTAC</nextState>
     <timeout>10</timeout>
     <outMsg>InitialAttack</outMsg>
    </transition>
   </transitionExt>
   ...
  </transitionsExt>
 </deltext>
</statemachine>
```

Fig. 4. DEVS SM model

5.4 DEVSJAVA code for JTAC

Finally, the DEVSJAVA code is obtained from transformations using a DOM parser on valid DEVS SM model (Figure 5). Providing complete execution details of code generation are beyond the scope of this paper. More details can be found in [17], [20], [19].

6 Conclusions and Future Work

We have addressed transformation between two modeling paradigms: UML State Charts and DEVS formalism. UML is graphical while DEVS is system theoretic based mathematically formulatable with its extensions to differential equations and continuous simulation. We have used SCXML as the transformation medium that takes a UML model and generate a finite deterministic (FD) DEVS model, a subset of DEVS model specification. The paper provides a proof of concept towards integration of UML and DEVS behavioral specifications.

We have developed the needed transformations to generate a DEVSJAVA-based model, which is a Platform Specific Model (PSM). Nevertheless, by specific XSLT transformations, we could generate the model for any DEVS-capable simulation

```
public class JTAC extends atomic {

 // ...

 public void deltext(double e, message x) {
  Continue(e);
  for(int i=0; i<x.getLength(); i++) {
   if(this.messageOnPort(x, ''inYouCanUseUSMCAircraft'', i)) {
    if(phaseIs(''WaitForAssignment'')) {
     processWaitForTACRequest();;
     holdIn(''WaitForTACRequest'',9999.0);
    }
   }
   // ...
  }
 }

 // ...

}
```

Fig. 5. DEVSJAVA code for JTAC

engine. Consequently, a Platform Independent Model is obtained. Infact, the FD-DEVS XML model obtained is a PIM. This DEVS-Based PIM is further used in net-centric DEVS-based collaboration as evident in our earlier work on DEVSML and DEVS-SOA that allows heterogenous cross-platform simulation possibilities. Section 4 provide some ideas to accomplish the same.

Furthermore, the approximation to other proprietary's tools is feasible using PIMs. For example, defining transformations from MathWorks StateFlow ®models [15] into SCXML models which give the possibility to generate FD DEVS models from StateFlow models.

References

[1] ACIMS Software Site, URL: http://www.acims.arizona.edu/SOFTWARE/software.shtml.

[2] Bendre, S. and H.S. Sarjoughian, "Discrete-Event Behavioral Modeling in SESM: Software Design and Implementation", Advanced Simulation Technology Conf., San Diego, CA., 2005, 23–28.

[3] Borland, S. and H. Vangheluwe, *Transforming Statecharts to DEVS*, Society of Modeling and Simulation International, 2003.

[4] Call Control eXtensible Markup Language (CCXML), URL: http://www.w3.org/TR/ccxml.

[5] Choi, KeungSik, SungChul Jung, HyunJung Kim, Doo-Hwan Bae and DongHun Lee, "UML-based Modeling and Simulation Method for Mission-Critical Real-Time Embedded System Development", IASTED Conf. on Software Engineering 2006, 2006, 160–165.

[6] Department of Defense, "DoD Acquisition GuideBook V2006", URL: https://akss.dau.mil/dag.

[7] Giambiasi, N., "From Timed Automata to DEVS Models", Proceedings of the 2003 Winter Simulation Conference, 2003.

[8] Harel, D. and M. Politi, "Modeling Reactive Systems with Statecharts: The STATEMATE Approach", McGraw-Hill, 1998.

[9] Hong, S. and T. Kim, "Embedding UML subset into Object-Oriented DEVS Modeling Process", Proceeding on Summer Computer Simulation Conference, San Jose, CA, July 2004, 161–166.

[10] Hopcroft, John, and Jeffrey Ullman, "Introduction to Automata Theory, Languages, and Computation", Addison-Wesley Publishing Company, 2002.

[11] Hwang, M.H. and B.P. Zeigler, *Reachability Graph of Finite & Deterministic DEVS*, submitted to IEEE Transactions on Automation Science and Engineering.

[12] IBM RSA (SCXML plugin), URL: http://www.alphaworks.ibm.com/tech/scxml.

[13] Janousek, Vladimir, Petr Polásek and Pavel Slavícek, "Towards DEVS Meta Language", ISC 2006 Proceedings, Zwijnaarde, BE, 2006, 69–73.

[14] JavaML, URL: http://www.badros.com/greg/JavaML.

[15] The MathWorks - StateFlow, URL: http://www.mathworks.com/products/stateflow/.

[16] Mellor, S.J. and Marc J. Balcer, "Executable UML: A Foundation for Model Driven Architecture", Addison-Wesley, 2002.

[17] Mittal, S. "DEVS Unified Process for Integrated Development and Testing of Service Oriented Architectures", Ph.D. thesis, University of Arizona, May 2007.

[18] Mittal, S., J.L. Risco-Martín and B.P. Zeigler, "DEVSML: Automating DEVS Execution Over SOA Towards Transparent Simulators", DEVS Symposium, Spring Simulation Multiconference, Norfork, Virginia, 2007, 287–295.

[19] Mittal, S., "W3C Schema for Finite Deterministic FD-DEVS Models", URL: http://www.u.arizona.edu/ saurabh/fddevs/FD-DEVS.html.

[20] Mittal, S., M.H.Hwang and B.P.Zeigler, "Finite & Deterministic DEVS for Template based State Machine Development", in progress.

[21] Mittal, S., *Extending DoDAF to Allow DEVS-Based Modeling and Simulation*, Special issue on DoDAF, Journal of Defense Modeling and Simulation JDMS, 3(**2**), 2006.

[22] Object Management Group, URL: http://www.omg.org.

[23] Risco-Martín, J.L., S. Mittal, M.A. López-Peña and J.M. de la Cruz, "A W3C XML Schema for DEVS Scenarios", DEVS Symposium, Spring Simulation Multiconference, Norfork, Virginia, 2007, 279–286.

[24] State Chart XML (SCXML): State Machine Notation for Control Abstraction, URL: http://www.w3.org/TR/scxml.

[25] Tolk, Andreas and James A. Muguira, "M&S within the Model Driven Architecture", Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC), , December 2004, Paper 1477.

[26] Vangheluwe, H.L. and Ghislain C. Vansteenkiste. *SiE: Simulation for the Future*, Simulation, 66(**5**), 1996, 331–335.

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

[28] Zinoviev, D., "Mapping DEVS Models onto UML Models", DEVS Symposium, Spring Simulation Multiconference, San Diego, CA, April 2005, 101–106.