Integrating Simulation Capabilities in SysML using DEVS

Mara Nikolaidou, Vassilis Dalakas and Dimosthenis Anagnostopoulos

Department of Informatics & Telematics
Harokopio University of Athens
70 El. Venizelou Str, 17671 Athens, GREECE.
{mara, vdalakas, dimosthe}@hua.gr.

***Abstract*—SysML is considered as an emerging standard for system engineering. Using SysML, the system engineer may study alternative system configurations. However, in order to be able to argue for or against a certain configuration, performance evaluation should be performed and SysML models should become executable using a simulation environment. For the simulation community, DEVS formalism provides a conceptual framework for specifying discrete event simulation models in a modular and hierarchical form. In addition DEVS is supported by a wide variety of DEVS simulators built in numerous programming environments. This paper illustrates and exploits the similarities between SysML and DEVS conceptual model and proposes a DEVS profile for SysML to make SysML models executable using a DEVS simulator. Integrating simulation capabilities into SysML models, automatic code generation for DEVS simulators may be possible, facilitating simulation for models already defined in SysML.**

I. Introduction

The Systems Modeling Language (SysML) is a general-purpose graphical modeling language for systems engineering. It supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. These systems may include hardware, software, information, processes, personnel, and facilities. SysML v.1.0, adopted by the OMG as a standard [1], is supported by the OMG Systems Engineering Domain Special Interest Group and by INCOSE (International Council on Systems Engineering). It is expected that the language will become a de facto standard for Systems Engineering, as UML has become one for software engineering. UML tools provide for both software design and corresponding software generation, contributing to the overall software creation process. The same vision was embraced for SysML regarding complex system composition. Though SysML standard [1] and corresponding profiles, implemented in popular UML modeling tools [2], provide the means for defining a system structure and behavior, they do not directly facilitate the evaluation of the designed system models.

Consequently, since SysML became a standard by OMG in 2006, it was clear that there was an urgent need to integrate SysML and simulation environments, provided that SysML models are defined in a way, which facilitates simulating them [3]. Apparently SysML supports a variety of diagrams describing system structure and states, necessary to perform simulation, thus, there are a lot of efforts from both research and industrial communities to simulate SysML models [4], [5]. SysML may be used for modeling a wide variety of systems, exhibiting either continuous or discrete behavior. Depending on the nature and specific characteristics of systems under study, there is a diversity of approaches on simulating models defined in SysML, which utilize different SysML diagrams. In most cases, SysML models defined within a modeling tool are extracted in XML format and, consequently transformed in a simulator specific format and forwarded to the simulation environment. In [6], a method for simulating the behavior of systems using mathematical simulation is presented, utilizing SysML parametric diagrams, which allow the description of complex mathematical equations. System models are simulated using COBs. A similar approach is presented in [7], where simulation models could be executed using Modelica. To ensure that a complete and accurate Modelica model is constructed using SysML, a corresponding profile is proposed. These approaches are better suited for systems with continuous behavior.

Simulation of discrete event systems is utilized, based on system behavior described in SysML activity, sequence or state diagrams. In [8], system models defined in SysML are translated to be simulated using Arena simulation software. Though, emphasis is given to system structure rather than system behavior. In [9], Colored Petri Nets are utilized to simulate SysML models. System behavior is described using mainly activity and sequence diagrams and may be consequently simulated using discrete event simulation via Petri Nets. In the following, we discuss how to embed discrete event simulation capabilities within SysML models using DEVS formalism, and ensure the construction of executable simulation code on a variety of DEVS simulators.

DEVS (Discrete Event System Specification) formalism [10] provides a conceptual framework for specifying discrete event simulation models in a modular and hierarchical form. Nowadays, a variety of software tools and simulators is available as *DEVJava* [11],or *DEVS-C++*. Extensions of these implementations operating in a distributed environment are also available as DEVS/HLA [12], DEVS/CORBA [13], cell-DEVS [14], DEVS/RMI [15] or even DEVS/SOA [16], which

offers DEVS simulators as web services. Since DEVS is inherently based on object oriented methodology, C++ and Java are the chosen programming languages. In any case, models are coded either in C++ or Java. DEVS simulation community is well established and very active, following the latest technological standards and treads in developing distributed application code.

In [17], DEVSXML, a platform-independent, XML-based format, for describing DEVS executable code was proposed independently of the underlying simulator. DEVSXML is consequently transformed into executable code for existing DEVS Simulators, using translators as the ones proposed in [17] for DEVSJava simulators. DEVSXML was proposed to establish DEVS model mobility and promote interoperability between discrete DEVS simulators independently of the programming language they are implemented in (either C++ or Java) and the way they operate (either in a distributed or centralized fashion). To achieve interoperability DEVSXML is used to describe platform-independent DEVS models according to Model Driven Architecture (MDA) concepts. Following MDA, for each real-world domain, two discrete models should be defined: a Platform-Independent Model (PIM), ensuring proper domain representation and Platform-Specific Models (PSMs), corresponding to one or more executable versions of PIM. In the DEVS domain, DEVSXML is used to define PIM, consequently translated into code executed on a variety of DEVS simulators, as DEVSJAVA and DEVS/SOA, corresponding to PSMs.

SysML and DEVS followed the same approach for system representation, since they both facilitate the description of systems as a hierarchy of interacting components. In practice, both DEVS formalism and SysML metamodel adopt the same concepts to describe system structure. These similarities could be exploited in order to bridge the gap between them and gain in functionality by offering a clear track from SysML models to valid system evaluation results. Embedding DEVS formalism and rules within SysML models may restrict the modeler when defining system behavior, but at the same time it also enables the straightforward execution of system models on existing, popular and effective simulation environments. The key advantage behind such an effort would be to enhance system engineering capabilities and promote the credibility of system analysis and design process. One way to accomplish DEVS and SysML interoperability, is to create an alternative platform independent model for DEVS using SysML concepts (as a DEVS SysML profile) and provide bidirectional transformation rules between the proposed model and DEVSXML.

The paper discusses SysML and DEVS similarities in Section II and the reasoning behind such an effort in Section III. Section IV gives some guidelines of how to build a simulation model using the proposed DEVS profile and finally Section V deals with issues relative to automated code generation.

II. Similarities of DEVS and SysML

The DEVS formalism is a conceptual framework consisting of mathematical sets to describe the structure and behavior of a model. Simulation models are specified in a modular and hierarchical form. Two types of models are defined: atomic models (behavioral representation), which describe basic system functionality and coupled models (structural representation) expressing how basic models are connected in a hierarchical form to built larger ones [10]. An atomic model consists of inputs, outputs, state variables and functions. Each model is described as:

- set of input ports for receiving external events
- set of output ports for sending external events
- set of state variables and parameters
- internal transition function (*deltint*), which specifies the next state to which the system will transit
- external transition function (*deltext*), which specifies the next system state when an input is received (the next state is computed on the basis of the present state, the elapsed time, and the content of the external input event)
- output function (*lambda*), which generates an external output just before an internal transition occurs
- time advance function (*ta*), which controls the timing of internal transitions

A coupled DEVS model contains the following information:

- set of components
- set of input and output ports
- external coupling, which connects the input/output ports of the coupled model to one or more input/output ports of the components
- internal coupling, which connects output ports of the components to input ports of other components - when an output is generated by a component it may be sent to the input ports of designated components (in addition to being sent to an output port of the coupled model).

In a DEVS model, system behavior is defined through the functions of atomic models, which are usually coded by the system designer using C++ or Java. Though, there are tools supporting DEVS structure definition in a graphical fashion, similar features are not supported for DEVS model behavior definition.

SysML aims at modeling systems in a graphical fashion, using discrete diagrams based on UML concepts. It supports the description of a)the structure of the system, b)its behavior and c)requirements imposed on its operation. System structure is described using two types of block diagrams, a) Block Definition Diagram (BDD) focusing on a hierarchical representation of system structure, where system components are defined as blocks, and b) Internal Block Diagram (IBD) focusing on composite block detail description, where component block relations are explored. Blocks are described using properties and constraints, while ports are defined to describe block communication. Furthermore, the Parametric Diagram (PD) is introduced to explore the relations between constraints. System behavior is described using Activity, Sequence, State Machine and Use Case diagrams, as defined in UML. The Requirement Diagram (RD) is introduced to depict system requirements.

SysML BDDs provide the means to define system composition, thus the overall system model is described by a corresponding BDD. Blocks, apart from value properties and constraints, may contain (part property) or use (reference property) other blocks, while they have ports used as the endpoints of inter-block connections. Ports facilitate sending or receiving events (standard ports) or data items (flow ports). Although BDDs may represent which are the components of each block, the way components are interconnected is defined in IBDs.

In a similar fashion using DEVS the overall system model is described using a coupled model, further decomposed to simpler models (either coupled or atomic) in a hierarchical fashion. Atomic models are used to depict system functionality, which can not be further analyzed. They are responsible to depict the behavior of the system. All models, both coupled and atomic are communicating via input and output ports defined for each model. DEVS ports are based on the same concept as SysML BDD ports. Thus, the structure of a system in both SysML and DEVS is defined in a similar fashion, allowing the bidirectional mapping between SysML and DEVS models [18].

To enable system simulation, the behavior of DEVS atomic models must be described through SysML. According to DEVS formalism, the atomic model behavior may be described using state charts, already supported in SysML to describe system behavior. Thus, SysML system blocks not further decomposed, should be characterized as atomic DEVS models and associate to DEVS atomic model behavior description, described using SysML behavior diagrams by restricting their functionality to conform to DEVS formalism. Based on these remarks, a DEVS profile for SysML was proposed, to provide DEVS behavior diagrams by restricting corresponding existing SysML diagrams and to facilitate the creation of DEVS simulation models for systems already modeled in SysML.

III. Scope of DEVS SysML Profile

The SysML Profile should provide the means to integrate on SysML models the definition of DEVS Simulation models, emphasizing DEVS atomic model behavior. The profile must facilitate the following:

- Ensure that the structure of system models defined is BDDs and IBDS contain all necessary information to simulate them using DEVS. This may be accomplished by specific constraints checking the ports and properties defined for system blocks participating in BDDs and IBDs to ensure that all DEVS related information is provided by system designer.
- Facilitate the means to characterize specific blocks as DEVS atomic models and provide DEVS Simulation SysML Diagrams to describe DEVS functions used to define the behavior of atomic simulation models.
- Define the experimentation environment.
- Create DEVS simulation code based on SysML diagrams.

The proposed profile is implemented in a standard UML modelling tool that supports a SysML profile, named Magic Draw [2]. The modeler could define his/her system model using SysML, add simulation properties using the DEVS profile and generate code for DEVS models. Code generation is completed in two phases:

1) A transformation between two different platform independent models (PIMs) according to MDA is performed to translate DEVS SysML entities, defined according to DEVS SysML profile) into of DEVS models defined in DEVSXML [17] and
2) A transformation of DEVS PIM defined in DEVSXML into executable DEVS platform specific models (PSMs) is performed. DEVSXML code is automatically translated into corresponding code in C++ or java (platform specific model – PSM) and executed in corresponding DEVS simulator. Such a translator for DEVSJava can be built, as described in [19].

The overall process is depicted in Fig. 1.

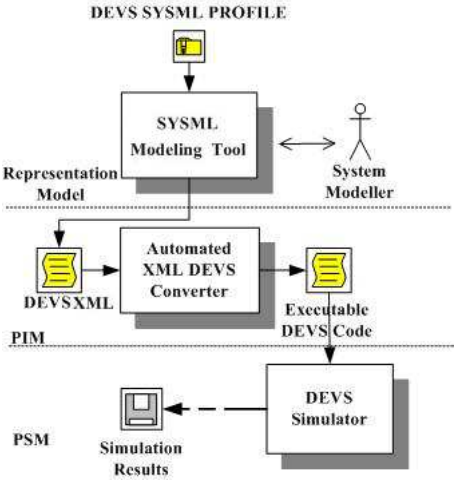

Fig. 1. Proposed Simulation Environment

IV. Defining simulation models with DEVS SysML profile

The formal method proposed by OMG of extending or restricting UML/SysML to effectively model a specific domain, as DEVS formalism, is the definition of stereotypes grouped by means of a profile [20].

In the following, we focus on SysML stereotypes used for the definition of DEVS Atomic model behavior. It consists of two stages:

- Model description, e.g. the definition of static characteristics, such as states and input and output ports.
- Behavior definition in response to input messages or time advancement.

A. DEVS Atomic Model Definition

DEVS Atomic models (DEVS AM) are defined as stereotypes of blocks in BDD diagrams. For each DEVS AM input

and output ports are defined. Atomic models behavior is defined as transitions between discrete model states [10]. In practice, states are described by corresponding state variables. For each DEVS AM, state variables are described as SysML block values, while constraints may also be defined to restrict the value range of a specific state variable. When the DEVS AM stereotype is applied on a system block in a BDD diagram, DEVS structural constraints are applied to the specific block to ensure the definition of DEVS ports and state variables [18].

B. DEVS Atomic Model Behavior

To describe atomic model behavior, system states and the four related functions, namely *delint, delext, lambda* and *ta* (see Section II) should be defined. For this purpose, four sub-diagrams must be related to each DEVS AM. Two of them facilitate state definition and the other two function definition.

- DEVS States Definition Diagram: A SysML constraint BDD defining constraints, each denoting a possible system state.
- DEVS States Association Diagram: A Parametric Diagram (PD) facilitating state definition based on the constraints of the previous BDD. The states (constraints) are formed from their association with the states variables (value properties).
- A DEVS Atomic Internal Diagram: A state machine diagram (SMD) facilitating the definition of internal transition function, output function and time advance function.
- A DEVS Atomic External Diagram: An activity diagram (AD) facilitating the definition of external transition function.

These diagrams are briefly discussed in the following.

1) States Definition: DEVS atomic model behavior is described as transitions between valid states. State are defined based on state variables define in SysML as block values. The valid set of states for each atomic model is defined as independent constraints based on corresponding state variables. Thus, SysML constraint BDD and corresponding SysML parametric diagram are utilized for the definition of DEVS atomic model states.

A simple example on state definition is presented in Fig. 2, where the states of an atomic model representing a Teller in a bank are defined. The Teller is either idle or busy performing two typical operations, a withdrawal and a deposit. These three states are defined as constraints based on the values of two Teller block properties, *s* indicating Teller *Status* (either busy or idle) and *j* indicating *Job Description* containing the specific jobs a Teller may perform.

As shown in Fig. 2, state variables are represented as constraint blocks in DEVS States Definition diagram. Each state variable is associated with one or more parameters (*s* and *j*).

The corresponding SysML PD, indicating the exact manner these parameters are associated with corresponding block values is named DEVS State Association diagram. Fig. 3 illustrates the DEVS State Association diagram of *Teller*

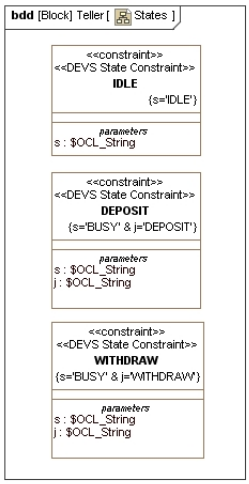

Fig. 2. Teller DEVS States Definition model

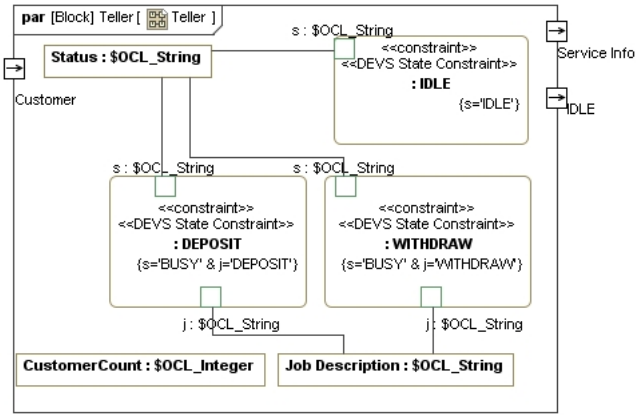

Fig. 3. Teller DEVS States Association model

atomic model. As shown in the figure, the *s* parameter of constraint *Deposit* is associated to the *Status* block value of *Teller* block.

Using those two diagram, discrete model states can be defined by combining discrete values of the state variables defined in associated DEVS AM blocks. Consequently, discrete model states can be automatically inserted in DEVS Atomic Internal and DEVS Atomic External diagrams used to define DEVS atomic model functions.

2) DEVS Atomic Internal Diagram: This diagram, corresponds to Internal Transition, Output and Time Advancement functions. In DEVS an internal transition function specifies the next state to which the system will transit. Output function generates an external output just before an internal transition occurs and Time Advance function controls the timing of internal transitions. State diagrams are used for the definition of the internal transition function. DEVS states are computed based on State Definition diagram and can be automatically inserted in the diagram. The modeler specifies internal transitions by inserting simple transitions between states. According to DEVS formalism, time is advanced every time a state transi-

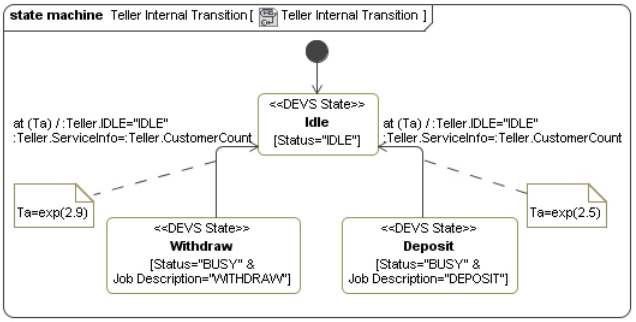

Fig. 4. Teller DEVS Atomic Internal model

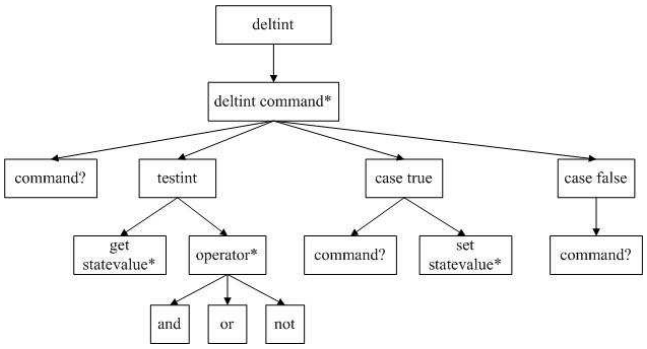

Fig. 5. Proposed XML Atomic Internal Function Representation

tion occurs. The Time Advancement function is represented by a corresponding note associated to each DEVS state transition. The initial state is determined by the initial values of each state variable. It was decided to include Output Function within the diagram rather than define a discrete one for each output port, since output generation is strictly related to internal state transition.

The diagram for Teller DEVS AM is depicted in Fig. 4, as an example. Teller atomic model changes its status from *Withdraw* state to *Idle* or from *Deposit* to *Idle*. Service time is an exponentially distributed random variable that is depicted with a note. When the service is finished, value assignment is taking place. Values are assigned to the output flow ports depicted in Fig. 3 as *ServiceInfo* and *IDLE*. This is depicted in Fig. 4 with the action body of a transition.

To provide an example of the way DEVS SysML stereotypes were defined, Atomic Internal model stereotypes are briefly presented in Table I. DEVS SYSML profile has been implemented using Magic Draw [2] modeling tool, which fully supports SysML and provides a rather friendly API. Proposed stereotypes are defined using standard interface, while constraints are implemented using the provided API or OCL.

3) DEVS Atomic External Diagram: DEVS Atomic External Diagrams are used to define DEVS external transition function. This function is executed whenever an input event arrives at the atomic DEVS model. Therefore, there is a parameter (of stereotype DEVS In Port) for every input flow port of the atomic DEVS model. The effect of this function (state modification) is also determined by the state of the atomic DEVS model at the time of the arrival of the input event. Thus, initially there is a decision node checking current state variable values and creating different control flows.

V. Transforming DEVS models to DEVSXML

As the main goal of integrating DEVS modeling capabilities into SysML is automated simulation code generation, transforming models defined with DEVS SysML profile into DEVSXML is essential. Modeling tools, as Magic Draw used for implementing DEVS SysML profile, provide the capability of exporting SysML models in XMI [1]. Both XMI and DEVSXML are used to define models represented in XML format. Thus, a conversion program should implement the automated transformation of DEVS SysML profile entities into DEVSXML entities and vise versa.

DEVSXML is currently used to transform DEVSJava code into XML and vice versa [19], while there are several versions under consideration, each focusing on different DEVS versions [21] [19]. All of them provide similar representation schemes for DEVS models structural part. Though, there are different approaches on how to represent DEVS atomic model behavior functions. In the version presented in [21], called DEVSML, behavior functions are simply represented in Java. The authors dealt with this and propose a format to represent behavior function code in DEVXML having in mind Java or C++ commands used to write corresponding function code. As an example, DEVS atomic internal function (delint) XML representation is presented in Fig. 5. The diagram describes the code representation corresponding to delint function, as a set of tests, which test the state value of a parameter and in case that the test result is true set the state value for another parameter.

Based on DEVSXML version presented in [17], as utilized in [19], both structural and behavioral DEVS characteristics are supported. In this case, atomic model functions are represented as state machine transitions. The transformation of DEVS SysML profile entities into DEVSXML entities is briefly presented in Table II, emphasizing DEVS Atomic model behavior functions. The transformation is straightforward and bi-directional.

The transformation of DEVS SysML profile entities emphasizing DEVS Atomic model behavior functions into the XML format presented in Fig. 5, is also straight-forward and bi-directional. The authors feel that the choice of which DEVSXML version to use should be based on the actual code generation capabilities provided. Thus, existing tools and corresponding code generation process performance should be explored. In any case, the concept of adopting DEVSXML as a discrete stage in constructing DEVS simulation code from SysML models may contribute to the efficient implementation of the overall process.

TABLE I
DEVS ATOMIC INTERNAL STEREOTYPES

DEVS Stereotype	SysML Entity	Constraints
DEVS Atomic Internal Diagram	State Machine Diagram	The diagram must be associated to a DEVS Atomic Block. The diagram contains an initial node, the DEVS States as derived from DEVS State Definition Diagram, transitions from state to state and notes indicating Time Advance.
DEVS State	State	Each state must be defined a state constraint in the DEVS State Definition Diagram of the same DEVS AM.
DEVS InTrans	State transition	Only one transition may start from any single state node. The transitions from state to state occurs at time *Ta* (time trigger event) that is identified with a note.
DEVS OutFn	DEVS InTrans	The action body of the transition has value assignments to the output flow ports of the DEVS Atomic Block.
DEVS Ta	Note	The DEVS Ta note is associated to a DEVS InTrans state transition The note contains the mathematical function describing advancement of time describing Ta.

TABLE II
DEVSXML – SYSML (BEHAVIORAL PART)

SysML	DEVSXML
	<!ELEMENT Internal_transition_function
<DEVS Atomic Internal Diagram>.<DEVS InTrans>.<where DEVS State is source>	<Conditional_function>
<DEVS Atomic Internal Diagram>.<DEVS InTrans>.<where DEVS State is target>	<State_variable_update>
	<!ELEMENT Output_function
<DEVS Atomic Internal Diagram>.<DEVS InTrans>.<where DEVS State is source>	<Conditional_output_function>
<DEVS Atomic Internal Diagram>.<DEVS InTrans>.<Action body>	<Send>
	<!ELEMENT Time_advance_function
<DEVS Atomic Internal Diagram>.<DEVS InTrans>.<where DEVS State is source>	<Conditional_time_advance>
<DEVS Atomic Internal Diagram>.<DEVS InTrans>.<Note>	<Time_advance>

VI. CONCLUSIONS AND FUTURE WORK

In this paper, the similarities between SysML and DEVS conceptual model were exploited and a systematic method to create DEVS simulation code based on existing SysML models were explored. The proposed DEVS SysML profile focusing on the definition of DEVS simulation models contributes toward this direction. Using SysML models defined using DEVS profile, DEVS executable code may be automatically constructed based on MDA concepts. DEVSXML format, consequently transformed in specific DEVS Simulator code, may serve as platform independent model describing DEVS executable models.

Currently, we are exploring DEVSXML versions and corresponding translation tools. We also focus on the implementation of the transformation between DEVS SysML model in XMI and DEVSXML.

REFERENCES

[1] OMG, "OMG System Modeling Language," *Available online via http://www.omg.org/docs/formal/07-09-01.pdf [accessed June 1, 2008]*, 2008.

[2] MG, *SysML Plugin for Magic Draw*, 2007.

[3] D. R. Tamburini, "Defining executable design & simulation models using sysml," March 2006.

[4] E. Huang, R. Ramamurthy, and L. F. McGinnis, "System and simulation modeling using sysml," in *WSC '07: Proceedings of the 39th conference on Winter simulation*. Piscataway, NJ, USA: IEEE Press, 2007, pp. 796–803.

[5] O. Schonherr and O. Rose, "First steps towards a general SysML model for discrete processes in production systems," in *Proceedings of the 2009 Winter Simulation Conference*, Austin, TE, USA, December 2009, pp. 1711–1718.

[6] R. Peak, R. Burkhart, S. Friedenthal, M. Wilson, M. Bajaj, and I. Kim, "Simulation-based design using sysmlpart 1: A parametrics primer," in *INCOSE Intl. Symposium*, San Diego, CA, USA, 2007.

[7] M. Association, "Modeling of complex physical systems," *Available online via http://www.modelica.org/ [accessed February 1, 2010]*.

[8] L. McGinnis and V. Ustun, "A simple example of SysML-driven simulation," in *Proceedings of the 2009 Winter Simulation Conference*, Austin, TE, USA, December 2009, pp. 1703–1710.

[9] C. Renzhong Wang, Dagli, "An executable system architecture approach to discrete events system modeling using SysML in conjunction with colored petri nets," in *IEEE Systems Conference 2008*. Montreal: IEEE Computer Press, April 2008, pp. 1–8.

[10] B. P. Zeigler, H. Praehofer, and T. Kim, *Theory of Modeling and Simulation*, 2nd ed. Academic Press, 2000.

[11] B. P. Zeigler and H. S. Sarjoughian, *Introduction to DEVS Modeling and Simulation with JAVA. DEVSJAVA Manual*, 2003. [Online]. Available: www.acims.arizona.edu/PUBLICATIONS/publications.shtml

[12] "Devs and hla: complementary paradigms for modeling and simulation?" *Trans. Soc. Comput. Simul. Int.*, vol. 17, no. 4, pp. 187–197, 2000.

[13] B. Z. H. S. Cho, Y., "Design and implementation of distributed real-time DEVS/CORBA," in *IEEE International Conference on Systems, Man, and Cybernetics*, Tucson, AZ, USA, October 2001, pp. 3081–3086.

[14] G. Wainer, , G. Wainer, and N. Giambiasi, "Timed cell-devs: modelling and simulation of cell spaces," 2001.

[15] M. Zhang, B. P. Zeigler, and P. Hammonds, "Devs/rmi-an auto-adaptive and reconfigurable distributed simulation environment for engineering studies," *ITEA Journal*, 2005.

[16] S. Mittal, J. L. Risco-Martín, and B. P. Zeigler, "Devs/soa: A cross-platform framework for net-centric modeling and simulation in devs unified process," *Simulation*, vol. 85, no. 7, pp. 419–450, 2009.

[17] J. L. R. Martín, S. Mittal, M. A. López-Pe na, and J. M. de la Cruz, "A w3c xml schema for devs scenarios," in *SpringSim '07: Proceedings of the 2007 spring simulation multiconference*. San Diego, CA, USA: Society for Computer Simulation International, 2007, pp. 279–286.

[18] M. Nikolaidou, V. Dalakas, L. Mitsi, G.-D. Kapos, and D. Anagnostopoulos, "A sysml profile for classical devs simulators," in *Proceedings of the Third International Conference on Software Engineering Advances (ICSEA 2008)*. Malta: IEEE Computer Society, October 2008, pp. 445–450.

[19] J. L. Risco-Martín, J. M. De La Cruz, S. Mittal, and B. P. Zeigler, "eudevs: Executable uml with devs theory of modeling and simulation," *Simulation*, vol. 85, no. 11-12, pp. 750–777, 2009.

[20] OMG, "OMG Unified Modeling Language: Superstructure, version 2," *Available online via http://www.omg.org/docs/formal/05-07-04.pdf [accessed June 1, 2006]*, August 2004.

[21] S. Mittal, J. L. Risco-Martín, and B. P. Zeigler, "Devsml: Automating devs execution over soa towards transparent simulators," in *DEVS Symposium, Spring Simulation Multiconference*. ACIMS Publications, March 2007, pp. 287–295.