From SysML models to DEVS executable code: The role of DEVS-XML

G.D. Kapos, V. Dalakas, M. Nikolaidou, D. Anagnostopoulos
Department of Informatics and Telematics
Harokopio University of Athens
70, El. Venizelou Str, 17671, Kallithea, Athens, Greece
{**gdkapos, vdalakas, mara, dimosthe**}**@hua.gr**

Keywords: DEVS, SysML models, Model transformation, Automated simulation code generation

Abstract

The authors have previously proposed a methodology for integrating simulation capabilities into SysML models, with the aid of the DEVS SysML profile that facilitates the construction DEVS simulation models based on SysML system models. In this paper, the proposed conceptual integration of SysML and DEVS models is materialized, by providing a concrete mapping of DEVS SysML profile entities to DEVS-XML elements, which is properly extended for this purpose. DEVS-XML is a declarative, platform independent language, which, however, may facilitate automatic simulation code generation and execution in various DEVS simulation execution environments. Thus, automatic simulation code generation and execution based the original system model described in SysML, applying MDA concepts and utilizing DEVS-XML, becomes feasible. Such an approach may facilitate a system engineer, familiar with SysML, to use a DEVS-based simulator to evaluate system models without the necessity of understanding DEVS theory and simulation-specific properties.

1. INTRODUCTION

In model-based system engineering, a central system model is used to perform all engineering activities in the specification, design, integration, validation, and operation of a system (definition by INCOSE [1]). Such activities are commonly served by independently/autonomously defined system models. System evaluation is an engineering activity which may performed in a model-based fashion using simulation.

In the simulation community, discrete simulation methodologies provide the means to define custom system models, which are consequently simulated using corresponding simulation environments. In this case, both system modeling and system simulation are treated as a unified activity.

This contribution presents an effort to bridge the community of system engineering with simulation community, combining a widely accepted graphical modeling language for complex systems with an established formalism for discrete event simulation. Such an effort would benefit both worlds as it could offer validity and reliability to the final results. SysML [13] was proposed as a general-purpose graphical modeling language of describing models of a broad range of systems and systems-of-systems. Specific activities may be accomplished either by the system engineer using a SysML modeling tool (for example system design) or by specific tools in an automated fashion (for example system validation) or even by a combination of both. DEVS (Discrete Event System Specification) formalism provides a conceptual framework for specifying discrete event simulation models executed on a variety of simulators [21], as DEVS-C++, DEVSJava [22] , cell-DEVS [19], DEVS/RMI [23] or even DEVS/SOA [9], which offers DEVS simulators as web services. In any case, executable models are defined either in C++ or Java.

Since SysML has become a standard by the Object Management Group (OMG), the need to integrate SysML modeling tools and simulation environments is evident. 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 [17]. In most cases, SysML models defined within a modeling tool are exported in XML format and, consequently, transformed into simulator specific models and forwarded to the simulation environment. 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 [15], a method for simulating the behavior of continuous systems using mathematical simulation is presented, utilizing SysML parametric diagrams, which allow the description of complex mathematical equations. System models are simulated using COBs. It should be noted that in any case SysML models should be defined in a way, which facilitates simulating them [18]. In [14], simulation is performed using Modelica. To ensure that a complete and accurate Modelica model is constructed using SysML, a corresponding profile is proposed to enrich SysML models with simulation-specific capabilities. All these approaches are better suited for system with continuous behavior.

Simulation of discrete event systems is utilized, based on system behavior described in SysML activity, sequence or state diagrams. In [5], system models defined in SysML are

translated to be simulated using Arena simulation software. SysML models are not enriched with simulation-specific properties, while emphasis is given to system structure rather than system behavior. Model Driven Architecture (MDA) concepts are applied to export SysML models from a SysML modeling tool and, consequently, transformed into Arena simulation models, which should be enriched with behavioral characteristics before becoming executable. In [20], the utilization of Colored Petri Nets is proposed to simulate SysML models. If the system behavior is described using activity and sequence diagrams in SysML, it may be consequently simulated using discrete event simulation via Petri Nets.

The authors have previously proposed, an integrated approach to transform SysML models to executable discrete event simulation models, utilizing MDA concepts for model transformation, as in [5] and [17]. Ideally, the simulation models extracted from SysML models should be executable without the additional programming effort from the system engineer, while model transformation should be bidirectional (from and to SysML models). To enable the construction of executable simulation models, discrete event simulation capabilities should be embedded within SysML models utilizing profile mechanism. Furthermore, the simulation methodology adopted should be popular and facilitate the execution of simulation models on a variety of simulators, while supported system models should be similar with SysML models to ease model transformation. SysML and DEVS follow the same approach for system representation, since they both facilitate the description of systems as a hierarchy of interacting components. These similarities could be exploited in order to integrate them and support the transformation of SysML models to valid executable DEVS simulation models and vice-versa. Embedding DEVS formalism detail description within SysML models enhances their expressiveness in terms of system validation, since it enables the straightforward execution of these models on existing, popular and effective simulation environments. At the same time, one might consider that it restricts the modeler when defining system behavior. Thus DEVS-related constraints should be applied only when system validation activity is performed, using a *DEVS SysML profile* properly extending SysML meta-model for simulation purposes. SysML models defined using DEVS SysML profile could be consequently simulated in any DEVS simulator, if transformed in DEVS-XML [4]. The purpose of this paper is to provide a concrete mapping of DEVS SysML profile entities to DEVS-XML elements, which is properly extended for this purpose.

The rest of the paper is structured as follows: The proposed approach for simulating SysML models using DEVS formalism is discussed in Section 2. In Section 3 the DEVS SysML profile is described, emphasizing the corresponding meta-model and provided functionality. In Section 4 the transformation between SysML models defined using DEVS profile and executable simulation models defined in DEVS-XML is discussed. Conclusions and future work reside in Section 5.

2. SYSML MODEL SIMULATION USING DEVS

SysML system models should be defined independently of specific implementations or tools and support different levels of detail to accommodate all engineering activities. In such case, simulation is considered as a discrete activity conducted independently of system modeling and supported by autonomous tools, though, performed based on the system model defined using SysML. Note that, this SysML model should be enriched with simulation-specific capabilities to serve system validation activity.

It is evident that 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 [10], as summarized in Table 1. Both coupled and atomic DEVS models correspond to SysML blocks, while DEVS coupled model description is similar with SysML *Internal Block Diagram (IBD)* diagram corresponding to each block further decomposed to other ones. DEVS state variables correspond to SysML block value properties, while SysML constraints may be used to depict the way state variables are interrelated to indicate system states.

Based on the similarity of SysML and DEVS system model definition, it is proposed to use DEVS in order to simulate system models defined in SysML. Though, in order for a SysML system model to be simulated using DEVS, DEVS atomic model behavior (e.g., DEVS functions) should be somehow included within the corresponding SysML model. The formal method proposed by OMG of extending or restricting UML/SysML to effectively model a specific domain, like DEVS formalism, is the definition of stereotypes grouped by means of a profile [11]. Using DEVS specific stereotypes of SysML behavior diagrams defined in DEVS SysML profile, their functionality can be restricted to conform to DEVS formalism (e.g., the description of DEVS atomic model functions).

Using the proposed DEVS SysML profile, one may define a SysML system model enclosing simulation capabilities using DEVS, utilizing the directed mapping between DEVS formalism and SysML meta-model (Fig. 1). As depicted in the figure, a platform-independent representation of SysML models (PIM) can be extracted by any SysML modeling tool using XML Metadata Interchange (XMI) [12], which defines an XML representation of UML and SysML meta-model. This SysML model contains all DEVS specific entities described using DEVS SysML profile, since they are formally defined as stereotypes of SysML entities. It can also be considered as a representation of DEVS formalism meta-model. The pro-

Table 1. Mapping between DEVS Formalism and SysML Entities

DEVS Formalism	SysML Entity
Atomic model	Block
Input port for events	Flow Port with Item Flow
Output port for events	Flow Port with Item Flow
State variables	Value properties & Constraints
Parameters	Value properties
DEVS atomic model functions (*deltint, deltext, lambda, ta*)	Behavior diagrams (either Activity, Sequence, State Machine)
Coupled model	Internal Block Diagram
Components	Blocks/Parts
Internal coupling	Connectors between Flow Ports of IBD's Parts
External coupling	Connectors between Flow Ports of the IBD's Enclosing Block and its Parts

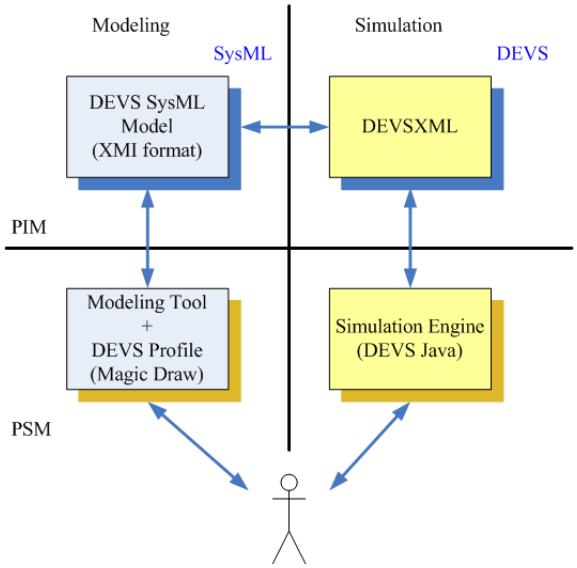

Figure 1. Proposed SysML Model Simulation Approach using DEVS

posed DEVS SysML profile and a corresponding API is implemented in Magic Draw [7], which is a standard UML modeling tool supporting SysML and a user-friendly programming interface.

In the DEVS simulation domain, an XML representation of DEVS can used to define a PIM, and consequently translated into code executed on a variety of DEVS simulators, as for example DEVSJAVA. There exist several efforts under consideration ([2], [4], [6], [8], [16]) for XML-based DEVS modeling and interpretation in different programming languages. An XML data encapsulation is accomplished in [2] within the DEVS environment as a unifying communication method among the entities in any SoS architecture. In [8] the problem of model interoperability is addressed, with a novel approach of developing DEVSML as the transformation medium towards composability and dynamic scenario construction. The composed coupled models are then validated using atomic and coupled DTDs. In this case, model behavior is not emphasized. In [6] an XML Schema is introduced for XLSC, a language for modeling atomic and coupled DEVS models. It was shown that a) XLSC can express a model's behavior as well as its structure and b) was shown how an XLSC model can be simulated. An interpreter was prototypically implemented in Java and employed to directly execute the model's functions and update the model's state. In this case, atomic model behavior can be described in XML using a series of actions depicting specific instructions included in the simulation code. In [4], DEVSXML was proposed, as a platform-independent, XML-based format for describing DEVS executable models. DEVSXML is consequently transformed into executable code for existing DEVS Simulators, using translators as the ones proposed in [4] for DEVSJava simulator. 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). The tools presented in literature are most of the times either under development or not available in public for evaluation. Among these tools, the one presented in [16] is the most advanced, and is currently used to transform DEVSJava code into XML and vice versa for a subset of the DEVS formalism, FD-DEVS [3]. The latter DEVS-XML version is referred as XFD-DEVS and offers XSD definitions among with a tool for the transformation. After reviewing XML DEVS versions available in the literature, we have decided to adopt DEVSXML ([16]), since it is generic, while atomic model behavior is described in an abstract, non-implementation specific fashion, better suited for DEVS SysML profile.

The system engineer may specify his/her system model in SysML using a SysML modeling tool, add simulation properties according to the DEVS profile, extract the SysML model in XMI and transform it into a DEVS executable model using DEVSXML. This process is fully automated and conforms to MDA guidelines.

In this paper, emphasis is given on system model transformations between the two different platform independent models (PIMs) to translate DEVS SysML entities, defined

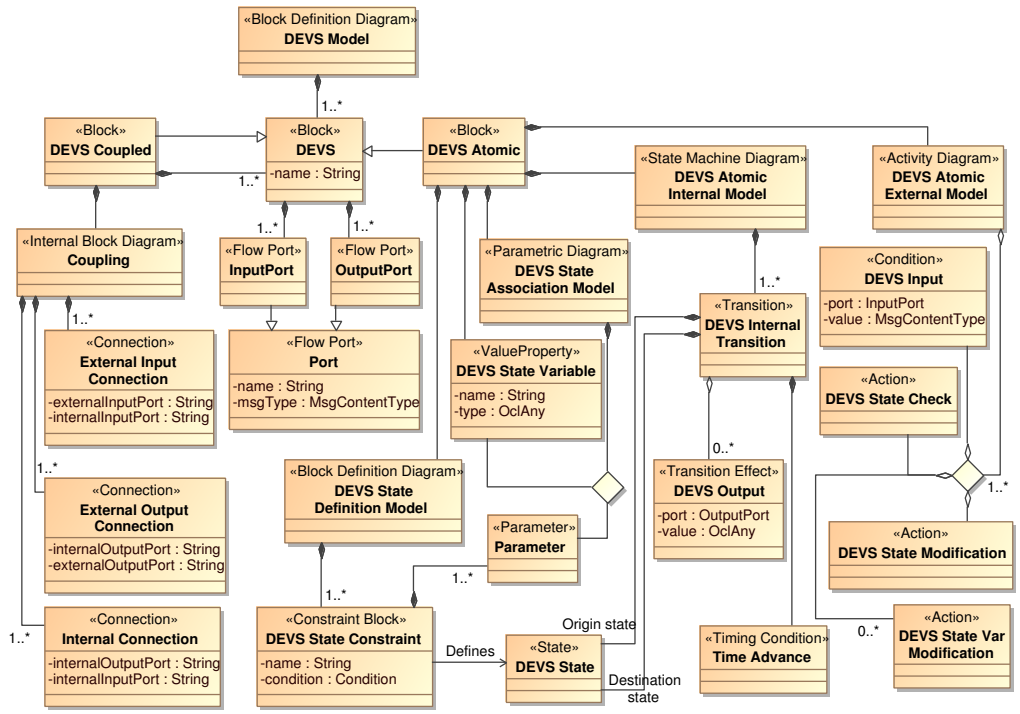

Figure 2. DEVS Profile Meta-model

according to DEVS SysML profile, into DEVS models defined in DEVSXML [4]. Such a transformation facilitates automated DEVS code generation based for SysML models conforming to DEVS SysML profile. The proposed profile is briefly described in the following section, while proposed stereotypes are discussed in [10].

3. DEVS SYSML PROFILE

DEVS SysML profile entities are summarized in Fig. 2. They are defined as stereotypes of SysML entities, since, according to OMG, the stereotype mechanism is used to extend/restrict SysML functionality. Associations included in the meta-model are aggregations, compositions, generalizations and generic associations, where textual description is given, while multiplicity constraints are also used.

Any SysML system model described using a Block Definition Diagram (BDD) is considered as a DEVS model. System blocks (with unidirectional ports) are identified as *DEVS* blocks and categorized as either *DEVS Coupled* or *DEVS Atomic* blocks. A *DEVS Coupled* block consists of a set of other blocks (atomic or coupled) and a *Coupling* element, expressed as an IBD. The coupling of a *DEVS Coupled* block defines the interconnections between (a) the ports of part blocks (internal connections) and (b) the ports of the container *DEVS Coupled* block and its parts (external connections). All this information is included in any SysML Internal Block Diagram (IBD) corresponding to a complex SysML block. When the SysML block is characterized as a *DEVS Coupled* block, by using the corresponding stereotype, related DEVS structural constraints are applied to ensure that all couplings between container and part block port are properly defined. So far, no specific DEVS-related entities are defined.

On the other hand, DEVS-related entities should be defined for any SysML block characterized as *DEVS Atomic* block, by applying the corresponding stereotype, in order to describe simulation model behavior. *DEVS Atomic* model behavior is defined as transitions between discrete model states [21]. Thus, a set of states and simulation model behavior must be described for any *DEVS Atomic* block, using a series of diagrams and the corresponding DEVS stereotypes. When the *DEVS Atomic* stereotype is applied on a system block in a BDD diagram, DEVS structural constraints are also applied to the specific block to ensure the definition of DEVS ports and state variables.

System states are defined in the *DEVS State Definition Model* as *DEVS State* constraints and are further explained as combinations of state variable values in the *DEVS State Association Model*. The latter, a stereotype of the PD, is used to show the constraint each *DEVS State* enforces on *State Variable* values. *State Variables* are defined as value properties of *DEVS Atomic* blocks, e.g., value properties of corresponding *DEVS Atomic* blocks included in the diagram.

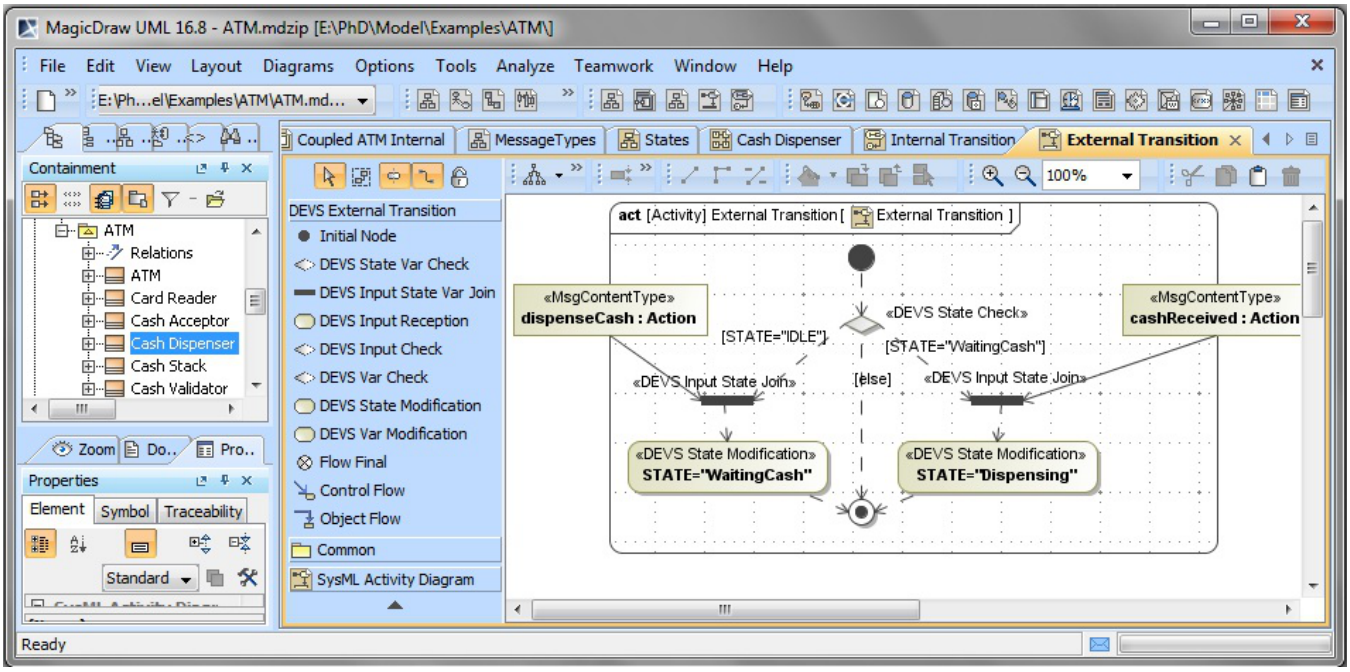

Figure 3. Profile screenshot - Cash Dispenser DEVS Atomic External Model

The *DEVS Atomic Internal Model* of a *DEVS Atomic* block specifies the behavior of the atomic DEVS model in case of internal state transition. Therefore, it is declared as a stereotype of State Machine Diagram (SMD), where *DEVS Internal Transitions* between the *DEVS States*, already defined in *DEVS State Definition Model*, occur at predefined *Time Advances* and may produce *DEVS Output* to some of the output ports of the atomic block.

The *DEVS Atomic External Model* of a *DEVS Atomic* block specifies the behavior of the atomic DEVS model in case of external state transition caused by the arrival of a specific input. It is declared as a stereotype of Activity Diagram (AD), where possible *DEVS States* and *DEVS Input* are combined. Each combination results in a set of actions, where current state and state variable values of the atomic block may be modified.

A screenshot of the previously mentioned DEVS Atomic External Model of an ATM component (Cash Despencer) is shown in Fig. 3. It is defined using the DEVS SYSML profile implemented in Magic Draw modeling tool. The proposed stereotypes are defined using standard tool interface, constraints, model customization, and the provided API.

4. TRANSFORMING DEVS SYSML MODEL TO DEVS-XML

DEVS SysML Profile enables system modelers to integrate the required simulation-specific information in the system model itself. Therefore, an equivalent DEVS model may always be constructed for every valid SysML model constructed using the profile. Although, the theoretical equivalent DEVS model may be of some interest, the executable DEVS model may be much more valuable from a practical perspective. To this end, expressing a DEVS SysML model in DEVS-XML is crucial, as the latter may automatically be transformed to code for a series of specific DEVS simulation environments ([4]).

Here we present how SysML models created using DEVS SysML profile may be transformed to DEVS-XML documents. A noticeable characteristic of this transformation is that the first part (DEVS SysML model) is a complex, conceptual model, created by a systems modeler using a GUI modeling tool, while the second part (DEVS-XML) is a plain XML document of a specific schema that can be compiled to executable code for DEVS simulation environments. This fact reveals that the two parts are not on the same level of abstraction within the proposed MDA process and implies a not so straightforward transformation definition.

In order to present the transformation in a more comprehensive, yet standardized way, we have decided to express the transformation using UML class diagrams. In UML class diagrams one may define complex model structures (using classes, their attributes and relations), as well as (directed) associations between elements of these models. These are the core concepts used for presenting the transformation. As far as the DEVS SysML model(left) part, we use a rearranged version of the DEVS SysML metamodel, in order to match DEVS-XML's structure and simplify the mapping. On the

DEVS-XML (right) part we use a class representation of the DEVS-XML structure defined in [4].

A set of conventions have been used in the diagrams, in order to make them more clear and readable. Each of the two different representations (DEVS SysML Profile and DEVS-XML) is placed in a dashed-line rectangle. Furthermore, DEVS SysML Profile classes are not filled with color, while DEVS-XML elements are light-grey. The multiplicity of each end of composition associations is implied to be 1, unless differently specified. Finally, mappings from DEVS SysML Profile elements to DEVS-XML elements are defined using thick directed associations.

For presentation reasons, the transformation of DEVS SysML models to DEVS-XML documents is divided in three class diagrams concerning (a) Coupled DEVS models, (b) structural part of Atomic DEVS models, and (c) behavioral part of Atomic DEVS models, respectively.

4.1. Mapping DEVS Coupled Models

Fig. 4 illustrates the mapping of DEVS SysML coupled models to respective DEVS-XML elements. The *DEVS Coupled* entity maps to *COUPLED_DEVS* element. Each of them consists of a series of properly mapped sub-entities and sub-elements, respectively. It is worth mentioning that, due to the more complex nature of the DEVS SysML metamodel, the notion of inheritance is used and depicted as a directional association with empty, triangular arrow head. For example, the *DEVS Coupled* entity inherits the *name* attribute, as well as *InputPorts* and *OutputPorts* from its superclass: *DEVS*. Similarly, both *InputPort* and *OutputPort* have the same structure due to their common origin: class *Port*.

The *DEVS Coupled* entity also has a set of internal DEVS models and coupling information. Due to its simple, structural nature the mapping to respective DEVS-XML elements is rather straightforward.

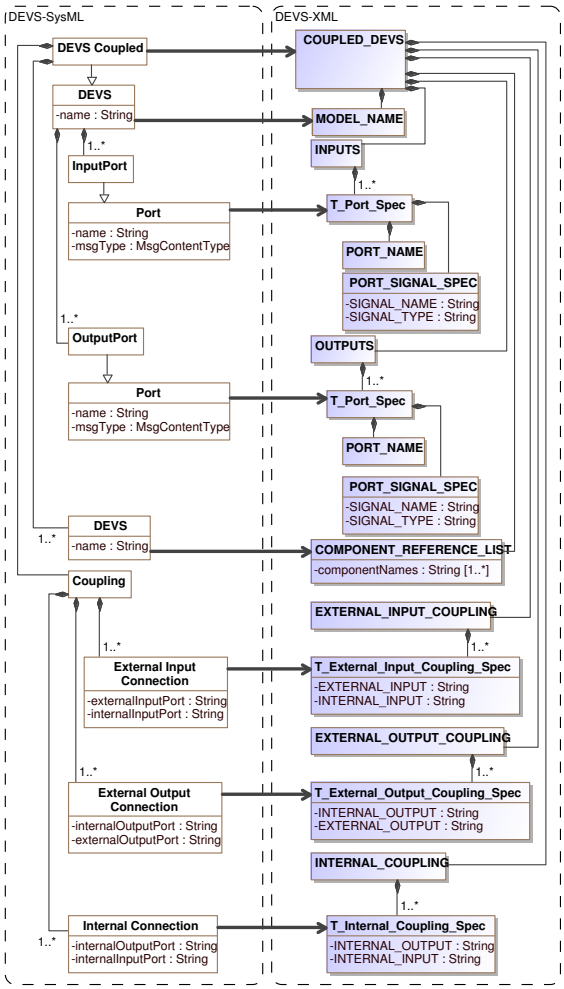

Figure 4. Mapping DEVS-SysML to DEVS-XML: Coupled DEVS models

4.2. Mapping DEVS Atomic Models: Structure

In a similar fashion, DEVS SysML Atomic Structural entities are mapped to respective DEVS-XML elements, in Fig. 5. However, there is a limitation in DEVS-XML: it does not correlate state set values with state variable values, a feature found very useful, when defining DEVS SysML models. This means that DEVS-XML does not define any state variable-dependent constraint that must hold in order to consider a state as current state. Thus, we have extended DEVS-XML, so that it becomes expressive enough to contain all DEVS-SysML simulation-related information. The two introduced elements (*STATE_SET_VALUE* and *CONDITION*) are easily identified in Fig. 5 due to their darker, grey fill color. With these additional elements it is possible to define a *STATE_SET_VALUE* with its standing *CONDITION*. Attributes *expr1* and *expr2* are mathematical expressions that may contain state variable names as variables, while *comp* attribute is a comparison operator. The recursive part of the definition of *CONDITION* element (*AND*, *OR*) allows the creation of rather complex conditions.

4.3. Mapping DEVS Atomic Models: Behavior

The mapping of DEVS atomic models behavioral aspects (depicted in Fig. 6) is particular in that DEVS-XML follows the classic DEVS form (internal transition, output, time advance, and external transition functions), while DEVS SysML profile introduces a much more compact form, to facilitate system engineer to describe DEVS model behavior in a simplified fashion. The main reason is the integration of internal transition, output, and time advance functions in *DEVS Atomic Internal Model*, a variation of SysML state machine

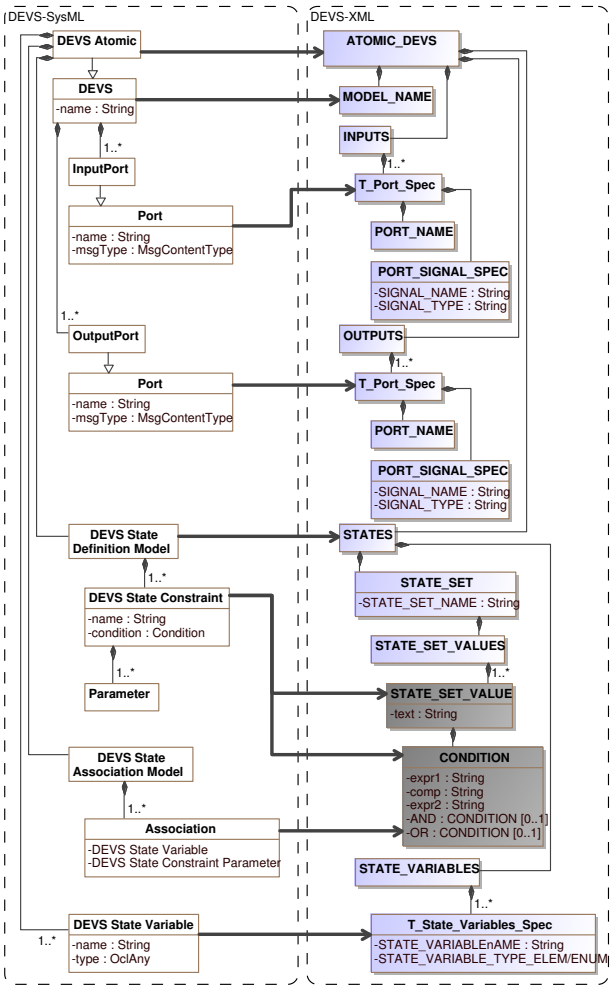

Figure 5. Mapping DEVS-SysML to DEVS-XML: Atomic DEVS models structure

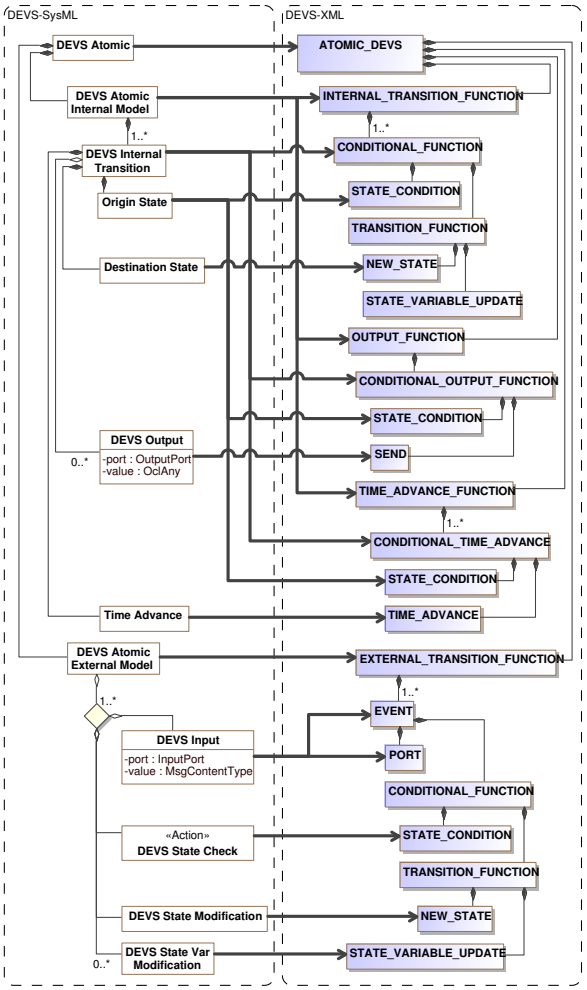

Figure 6. Mapping DEVS-SysML to DEVS-XML: Atomic DEVS models behavior

diagrams. However, this issue does not affect model transformation, other than making the mapping between DEVS SysML profile entities and DEVS-XML tags more complex. For example, each DEVS SysML profile *Origin State* entity included in *DEVS Atomic Internal Model* is mapped to three different DEVS-XML *STATE_CONDITION* elements, one for discrete function, e.g. internal transition, output and time advance.

5. CONCLUSIONS-FUTURE WORK

In this paper, the transformation of SysML models, including simulation capabilities through DEVS SysML profile, to DEVS-XML documents was explored. DEVSXML format describing DEVS executable models, consequently transformed in specific DEVS Simulator code, may serve as platform independent model for system simulation, while XMI exported from SysML modeling tools supporting DEVS SysML profile plays the same role for system modeling. Currently, we are focusing on testing the conversion tool to transform DEVS SysML models in XMI to DEVSXML and vice-versa and exploring the transformation of DEVS-XML models to DEVSJava code.

Applying the proposed concepts, DEVS SysML models can be transformed to executable DEVS models in DEVS-Java. Though, the conditions under which they should be validated, called Experimentation Framework in DEVS formalism, must be coded in DEVS simulator by the system engineer. Our future plans focus on the definition of the Experimentation Framework in SysML by properly extending DEVS SysML profile. This should help the system engineer to define in SysML a) the system model, b) its simulation-specific properties and c) the conditions under which it should be validated and, consequently, automatically produce executable simulation code for DEVS simulators.

REFERENCES

[1] L. Baker, P. Clemente, B. Cohen, L. Permenter, B. Purves, and P. Salmon. *Foundational Concepts for Model Driven System Design*. INCOSE Model Driven System Design Interest Group, International Council on Systems Engineering, July 2000.

[2] M. Hosking and F. Sahin. An xml based system of systems discrete event simulation communications framework. In *SpringSim '09: Proceedings of the 2009 Spring Simulation Multiconference*, pages 1–9, San Diego, CA, USA, 2009. Society for Computer Simulation International.

[3] M. H. Hwang and B. Zeigler. Reachability graph of finite and deterministic devs networks. *Automation Science and Engineering, IEEE Transactions on*, 6(3):468–478, July 2009.

[4] 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*, pages 279–286, San Diego, CA, USA, 2007. Society for Computer Simulation International.

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

[6] N. Meseth, P. Kirchhof, and T. Witte. Xml-based devs modeling and interpretation. In *SpringSim '09: Proceedings of the 2009 Spring Simulation Multiconference*, pages 1–9, San Diego, CA, USA, 2009. Society for Computer Simulation International.

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

[8] 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*, pages 287–295. ACIMS Publications, March 2007.

[9] 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*, 85(7):419–450, 2009.

[10] 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)*, pages 445–450, Malta, October 2008. IEEE Computer Society.

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

[12] OMG. MOF 2.0 XMI Mapping Specification. Version 2.1.1. O. M. G. Inc, December 2007.

[13] OMG. Systems Modeling Language (SYSML) Specification. Version 1.0. O. M. G. Inc, September 2007.

[14] C. J. J. Paredis and T. Johnson. Using omg's sysml to support simulation. In *WSC '08: Proceedings of the 40th Conference on Winter Simulation*, pages 2350–2352. Winter Simulation Conference, 2008.

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

[16] 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*, 85(11-12):750–777, 2009.

[17] 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*, pages 1711–1718, Austin, TE, USA, December 2009.

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

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

[20] R. Wang and C. Dagli. An executable system architecture approach to discrete events system modeling using SysML in conjunction with colored petri nets. In *IEEE Systems Conference 2008*, pages 1–8, Montreal, April 2008. IEEE Computer Press.

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

[22] B. P. Zeigler and H. S. Sarjoughian. *Introduction to DEVS Modeling and Simulation with JAVA. DEVSJAVA Manual*, 2003.

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