SIMULATING SYSML MODELS: AUTOMATED CODE GENERATION FOR DEVS SIMULATORS

Vassilis Dalakas
vdalakas@hua.gr

George-Dimitrios Kapos
gdkapos@hua.gr

Mara Nikolaidou
mara@hua.gr

Dimosthenis Anagnostopoulos
dimosthe@hua.gr

Dept. of Informatics and Telematics
Harokopio University of Athens
70, El. Venizelou Str., 17671, Athens, GREECE

ABSTRACT

SysML is considered as an emerging standard for model-based system engineering where the system engineer should perform all engineering activities based on a common model. The evaluation of system models designed by the system engineer is one of the most common engineering activities, frequently performed using simulation. Thus, there are numerous efforts to simulate SysML models using a variety of simulation methodologies and tools. None of them support the automated generation of executable simulation code suitable of a specific tool or environment. DEVS formalism provides a conceptual framework for discrete event simulation and is supported by a wide variety of simulators built in numerous programming environments. Based on the similarities between SysML and DEVS model structure, this paper presents a methodology and a set of tools for constructing executable DEVS code for system models already defined in SysML. To achieve this, a DEVS SysML profile is proposed integrating simulation capabilities into SysML, while MDA concepts are adopted to realize SysML model transformation into DEVS executable code executed in an XML-based DEVS simulation environment. To facilitate model transformation using QVT language, a DEVS XMI metamodel is also introduced. The overall process is discussed with the aid of a working example.

Keywords

MDA, Simulation Methodology, DEVS, Simulation Framework, Automated Simulation Code Generation, SysML

1. INTRODUCTION

Model-based system engineering, as defined by International Council on Systems Engineering (INCOSE) [1], is utilized by a central system model, used to perform all engineering activities in the specification, design, integration, validation, and operation of a system. Systems Modeling Language (SysML) [2] was proposed by the Object Management Group (OMG) as a general-purpose graphical modeling language of describing such models of a broad range of systems and systems-of-systems. SysML system models should be defined independently of specific implementations or tools. 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. In the case where specific tools are used, SysML models should be transformed to tool specific models, serving the specific engineering activity.

Since SysML became a standard, the need to integrate SysML modeling tools and simulation environments was 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 [3, 4]. In most cases, SysML models defined within a modeling tool are exported in Extensible Markup Language (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 [5], 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 composable objects (COBs) [6]. It should be noted that in any case SysML models should be defined in a way, which facilitates simulating them [7]. In [8], 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. 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 [9], system models defined in SysML are

translated to be simulated using Arena simulation software. Model Driven Architecture (MDA) concepts are applied to export SysML models from a SysML modeling tool and, consequently, transformed into Arena simulation models, which must be enriched with behavioral characteristics before becoming executable. In [10], 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. In both cases, although SysML system models are extracted and used, the system engineer must write large parts of the simulation code, especially concerning system behavior.

In this paper we propose an integrated approach to simulate SysML model by generating executable simulation models based Discrete Event System Specification (DEVS) formalism. The system engineer may enrich existing SysML models with DEVS specific characteristics and, then, automatically generate simulation code for specific DEVS simulators. Concepts of MDA, as in [9] and [4], are adopted to form a methodology that transforms SysML models to executable DEVS simulation models. To enable automated simulation code generation, DEVS simulation capabilities are embedded within SysML models using a profile mechanism. There were two key reasons for selecting DEVS simulation formalism:

(i) DEVS formalism provides a conceptual framework for specifying discrete event simulation models executed on a variety of simulators [11], as DEVS-C++, DEVSJava [12], cell-DEVS [13], DEVS/RMI [14] or even DEVS/SOA [15], which offers DEVS simulators as web services, and

(ii) Both SysML and DEVS follow the same principles regarding model structure [16]. The main structural elements in SysML are blocks with properties and ports, which are very similar to DEVS components. Both support composite and simple models. Containment and interconnection of SysML blocks through their ports is similar to composition and coupling in DEVS coupled blocks. These similarities facilitate the transformation of SysML models to valid executable DEVS simulation models. Embedding DEVS formalism detailed description within SysML models provides the means to describe model behavior and enables the automated execution of these models on existing, popular and effective simulation environments.

The rest of the paper is structured as follows: The proposed methodology for automated DEVS simulation code generation and execution from SysML system models is discussed in Section 2. Related steps and the way to implement them using standard tools based on the MDA approach are also discussed. In Section 3 SysML to DEVS model transformation is explored. Standard MDA languages Query/View/Transformation (QVT) are utilized to describe such transformation between SysML and DEVS metamodels. Then, in Section 5 the way such models may be executed in an XML-based DEVS simulation environment, named XLCS DEVS, is briefly presented. Conclusions and future work are summarized in Section 6.

2. PROPOSED METHODOLOGY

Both SysML and DEVS are established and widely accepted in the areas of system modeling and simulation, respectively. Furthermore, several approaches have been proposed on the one hand for the simulation of SysML models and on the other hand for the visual modeling of DEVS models. Utilizing the de-facto acceptance of SysML and DEVS, a methodology is defined for simulating SysML models using DEVS simulators, provided that model are described in a way compatible to DEVS formalism. The system engineer specifies his/her system model using SysML via a modeling tool and receives valid simulation results from the execution of the corresponding DEVS model in an appropriate simulation environment. An overall perspective of the proposed approach is depicted in Fig. 1.

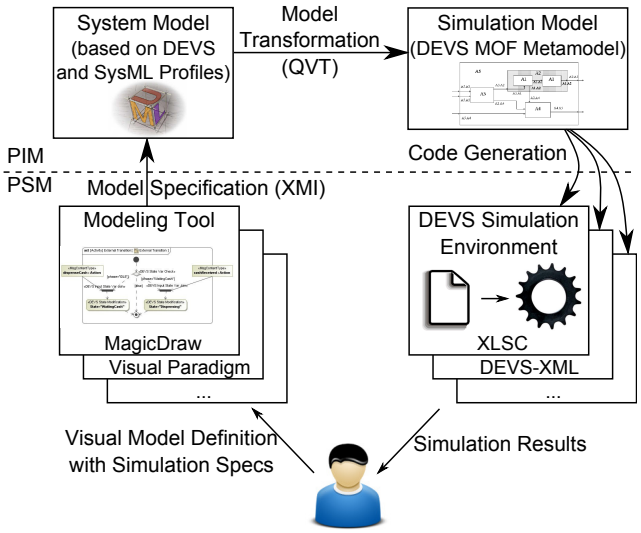

Figure 1: Simulating SysML Models with DEVS

In order to provide such capabilities, the system engineer should enrich system models with DEVS-compliant simulation information. This is performed during system modeling, within a SysML modeling tool. System model enrichment is enabled by a proposed DEVS SysML profile. The profile constraints and extends SysML models, so that they are simulation enabled. The DEVS SysML profile consists of a set of stereotypes and constraints. Stereotypes are used to characterize specific SysML model elements, while constraints specify how models should be created.

DEVS enriched SysML models can be automatically transformed to simulation code for specific DEVS simulators. Since DEVS formalism is supported by numerous implementations, the transformation includes an intermediate, yet autonomous and very important step, which is the generation of a pure DEVS representation of the system model, based on a DEVS MOF 2.0 metamodel. The Meta-Object Facility (MOF) is an OMG standard for model-driven engineering that allows the definition of models representing specific domains, like the DEVS simulation formalism using XML Metadata Interchange (XMI). The existence of a DEVS metamodel independent of specific simulators enhances the usability of the proposed approach and facilitates simpler transformations for diverse simulation environments, as DEVSJava, cellDEVS, etc.

Transformation of enriched SysML models to DEVS XMI representations is specified and implemented using QVT, a standard set of languages for model transformation defined by the OMG. Specifically, QVT defines three transformation languages: QVT-Operational, QVT-Relations and QVT-Core. In this case, QVT-Relations, a declarative lan-

guage for defining constraints on source and target model elements, has been used. QVT transformations can be applied on models that conform to MOF 2.0 metamodels (SysML and DEVS metamodel in our case). Object Constraint Language (OCL), another OMG standard language for defining constraints, is integrated and also extended in QVT with imperative features.

The transformation of DEVS MOF models to executable code for specific DEVS simulation environments is feasible, since all DEVS related information is clearly contained in the DEVS MOF models. However, the transformation depends on the target simulation environment. In our current implementation, the XML-based language for simulation components (XLSC) DEVS [17] was used as the simulation environment. The simulator accepts as input DEVS models described in XML and simulates them in a Java environment.

From a software engineering perspective, according to concepts of MDA [18], for each real-world domain, two kinds of 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 the PIM. In the DEVS simulation domain, the DEVS MOF metamodel is used to define a PIM, and consequently translated into code executed on a variety of DEVS simulators, as XLSC DEVSJava and DEVS/SOA, corresponding to PSMs. In the SysML modeling domain, DEVS SysML profile is used to define SysML models enriched with simulation capabilities. The SysML metamodel and the DEVS profile is used to define a PIM, while discrete Unified Modeling Language (UML)/SysML modeling tools correspond to PSMs.

From an implementation perspective, the DEVS SysML profile and a corresponding application programming interface (API) is implemented for MagicDraw [19], which is a widely used UML modeling tool supporting SysML with a user-friendly programming interface. SysML models constructed using the DEVS SysML profile and exported in XMI format are transformed into DEVS models (conforming to the DEVS MOF metamodel) via a QVT model transformation, that has been defined for that purpose. As a proof of concept, the last part of the transformation (code generation) has been implemented for XLSC DEVS execution environment [17]. This has been implemented in terms of EXtensible Stylesheet Language Transformations (XSLT) [20].

Within the proposed methodology, our main contribution relates to the definition of the DEVS-SysML profile and the corresponding implementation for MagicDraw modeling tool, the definition of the DEVS MOF 2.0 metamodel and the definition and implementation of the QVT transformation of DEVS enriched SysML PIMs to DEVS PIMs. An XSLT transformation of DEVS PIMs to XLSC DEVS PSMs has also been implemented as a proof of concept. Specification and implementation of the above-mentioned elements are described in the following sections.

To explore all the steps of the proposed methodology, a simple example often used in DEVS literature is employed and discussed in the following sections. It consists of a simple processor model and its experimental frame (EF), indicating the conditions under which the processor operates. The overall system is called EFP and is described in Fig. 2. The system consists of a processor, which is an atomic DEVS model and the EF, coupled DEVS model, consisting of a generator generating requests directed to the processor and a transducer collecting statistics.

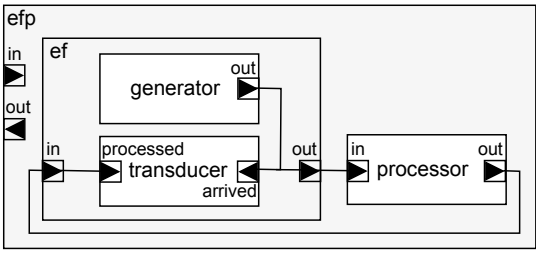

Figure 2: EFP DEVS model

3. DEVS SYSML PROFILE

As indicated in Fig. 1, system modelers initiate the process of simulating SysML models through enriching with DEVS simulation properties. Using DEVS specific stereotypes, SysML block functionality can be restricted to conform to DEVS formalism, while SysML behavior diagrams (e.g. state machine and activity) may be restricted to describe DEVS model behavior (e.g. the description of DEVS atomic model functions) [16]. The DEVS SysML profile has been implemented in MagicDraw modeling tool. Proposed stereotypes are defined using standard tool interface, while constraints are implemented using OCL [21], model customization, and the provided API.

Any SysML system model described using a 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 Internal Block Diagram (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 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. The IBD of the EF composite block described in Fig. 2 is depicted in Fig. 3, as an example.

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. To describe atomic model behavior, system states and the four related functions, namely *deltint, deltext, lambda* and *ta* should be defined. *DEVS Atomic* model behavior is defined as transitions between discrete model states [11]. 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. For this pur-

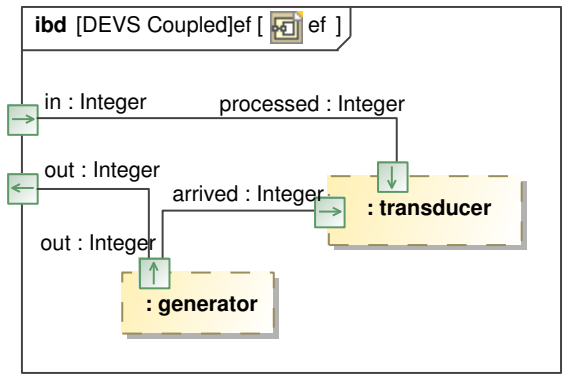

Figure 3: EF Internal Block Diagram

pose, four SysML diagrams must be related to each Atomic DEVS Block. Two of them facilitate state definition and the other two function definition.

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

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 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. Table 1 includes the stereotypes defined for *DEVS Atomic Internal Model*. It contains SysML stereotypes, SysML/UML entities on which they are applied, and constraints, regarding internal model specification.

In DEVS, an internal transition function specifies the next state to which the system will transit. DEVS states are computed based on DEVS State Definition Model and automatically inserted in DEVS Atomic Internal Model. The system modeler specifies DEVS Internal Transitions by inserting DEVS State Transitions between DEVS States. Output function generates an external output just before an internal transition occurs and Time Advance function controls the timing of internal transitions. The initial state is determined by the initial values of each state variable. The DEVS Atomic Internal Model for *processor* block, marked as a DEVS Atomic block, is depicted in Fig. 4, as an example. *Processor* system changes its state from *busy* to *idle* (as defined by DEVS Internal Transition represented as a state transition) after 3.5 seconds (as defined by Time Advance represented as Timing Condition of corresponding transition). It also produces output, as indicated by transition effect representing DEVS Output. Specifically, a job number is assigned to *out* output port. Fig. 4 also depicts a snapshot of MagicDraw modeling tool, supporting DEVS SysML profile, where corresponding stereotypes can be used from the palette.

Having defined the DEVS SysML profile, construction of SysML system models with DEVS simulation characteristics is feasible, and in an intuitive manner. As described in Fig. 1, the next step toward DEVS simulation code generation is the creation of the corresponding DEVS model representation in a standard common format as XML, that can be further transformed into DEVS executable code for specific simulators. This XML representation of DEVS model should be generated as the result of a transformation of the SysML system model.

4. MODEL TRANSFORMATION

DEVS SysML models comply to the UML2 MOF metamodel, which is a quite general metamodel that can be used for modeling a variety of artifacts, systems, processes, etc. This means that although we focus on DEVS related information with the DEVS SysML profile, the entities of the models contain large amounts of information that is irrelevant to DEVS-oriented simulation. Therefore, XMI representations of SysML models, exported from MagicDraw tool, are very large and cumbersome to be used for DEVS simulation code generation. Additionally, in the sense of standardizing the way DEVS models should be specified in an XMI representation, independently of how they were constructed (e.g., DEVS SysML models, DEVS visual tools), there is a need to define the DEVS metamodel in terms of a standard metamodeling facility. Such a facility is MOF, which is provided as a standard by the OMG.

4.1 The DEVS MOF Metamodel

After the evaluation of several proposed DEVS-XML representations, we have defined the DEVS MOF metamodel, based on the version of DEVS-XML proposed in [22]. DEVS MOF metamodel is schematically presented in the UML Class Diagram of Fig. 5(right part), but has also been defined in terms of MOF elements. Thus, it can be used within model manipulation tools (i.e., for model transformation), as MediniQVT. DEVS MOF metamodel defines elements defining structural and behavioural aspects of DEVS atomic (ports, states, internal transition, output, time advance, and external transition functions) and coupled components (ports and coupling).

Compared to DEVS-XML [22], the DEVS MOF metamodel incorporates the relation between state variable values and states. This is feature enabling the execution of corresponding XMI DEVS model using a wider variety of DEVS simulators, either implemented in C++ or Java. It also handles complex expression values and provides simpler structure and conceptual coherence. Therefore, DEVS MOF metamodel establishes a standard, solid foundation for defining DEVS models.

Regarding Fig. 5, it should be noted that the definition of CONDITION and VALUE classes is recursive. For ex-

Table 1: DEVS Atomic Internal Stereotypes

DEVS Stereotype	SysML Entity	Constraints
DEVS Atomic Internal Model	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 Internal Transition	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 Internal Transition	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 Internal Transition state transition The note contains the mathematical function describing advancement of time describing Ta.

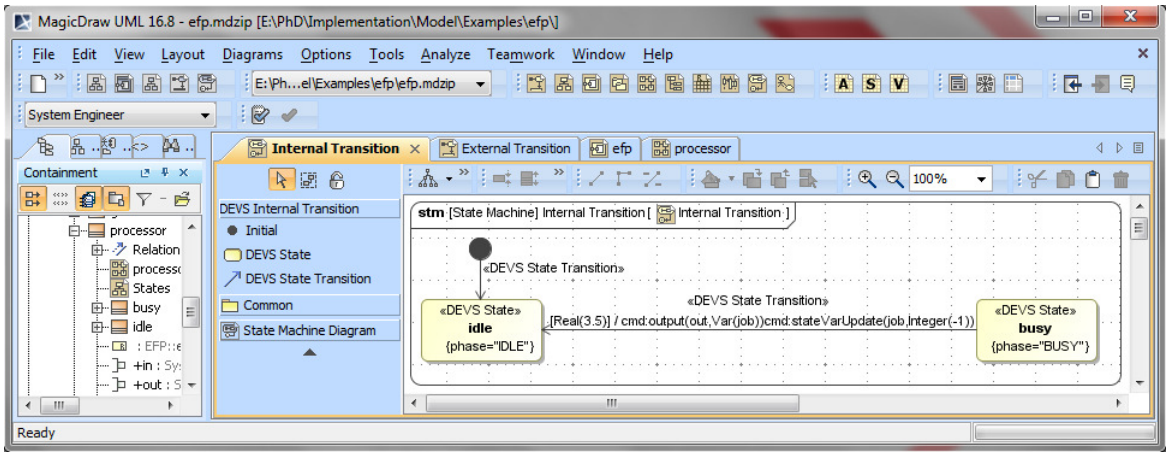

Figure 4: Processor DEVS Atomic Internal model

ample, a VALUE entity may be composed of one or more other VALUE entities. This is useful and allowed only when an operation is specified, so that a complex expression may be declared (e.g., V1+V2). In the case of CONDITION, *AND* or *OR* sub-conditions are allowed, to declare complex conditions (e.g., C1 AND C2).

4.2 Model Transformation with QVT

Models defined using the DEVS SysML profile may be represented in XMI format, according to the UML2 MOF metamodel. Since a compatible MOF metamodel has been defined for DEVS, an appropriate transformation from UML2 to DEVS MOF is required. A set of QVT relations between the stereotypes of DEVS SysML profile included in UML2 MOF and DEVS MOF metamodels, that implement a transformation of models from the first to the second, have been defined. An Eclipse-based QVT tool (MediniQVT) has been successfully used for the execution of the QVT transformation for various DEVS SysML models.

The transformation is shortly described in Fig. 5, illustrating both UML2 DEVS SysML (left side) and DEVS (right side) MOF entities. It also provides a schematical representation of the transformation, by indicating the correspondence between parts of the two metamodels. The numbered arrows, denoting several parts of the transformation, are described in the following:

1. The topmost part of the transformation: The *DEVS Model Block Definition Diagram* (UML2 entity) contains all *DEVS Atomic* and *DEVS Coupled* blocks to be transformed into DEVS entities *DEVS_ATOMIC* and *DEVS_COUPLED* . It is implemented by a QVT relation, which finds all *DEVS Model Block Definition Diagram* entities in the UML2 XMI (normally there will be only one) and creates a DEVS MODEL for each one.
2. DEVS common elements transformation: This part transforms elements that are common in atomic and coupled DEVS models, i.e., model name and input / output ports.
3. DEVS atomic state definition transformation: Transforms the *DEVS State Constraint* blocks that define the state set.
4. DEVS atomic state variable definition transformation: Transforms the *DEVS State Variable value properties* to *DEVS State Variables*.
5. DEVS atomic state association transformation: Transforms the *DEVS State Association Model* to conditions attached to state set values.
6. DEVS atomic internal model transformation: Transforms the *DEVS Atomic Internal Model* (State Ma-

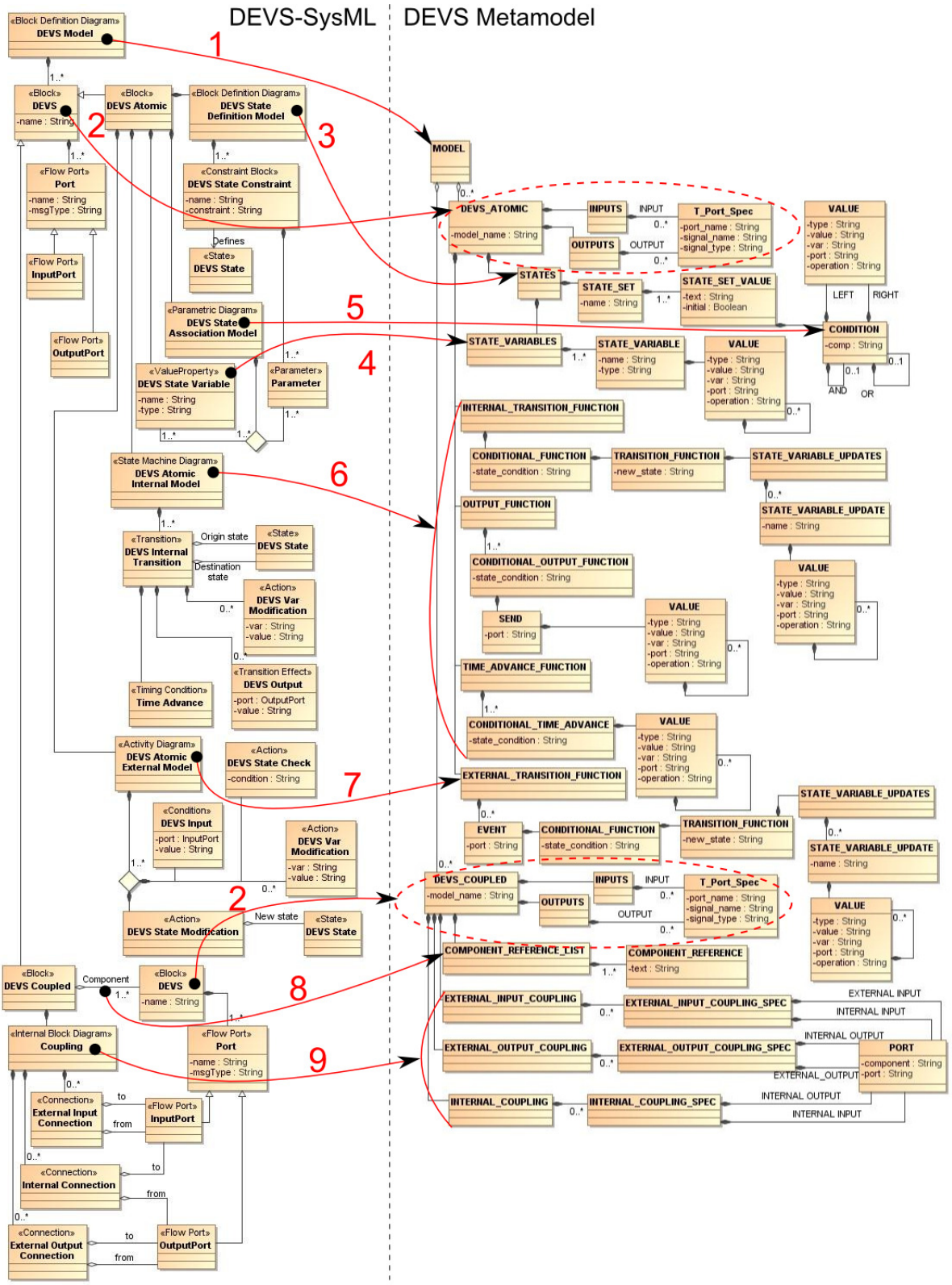

Figure 5: DEVS-SysML to DEVS Transformation (QVT)

chine Diagram) to *Internal Transition Function, Output Function* and *Time Advance Function.*

7. DEVS atomic external model transformation: Transforms the *DEVS Atomic External Model* (Activity Di-

agram) to *External Transition Function*.

8. DEVS coupled components transformation: Transforms the Component composition associations to the component reference list.
9. DEVS coupled coupling transformation: Transforms the port connections of the Internal Block Diagram to External Input Coupling, External Output Coupling and Internal Coupling.

The XMI code generated by the transformation of the Processor DEVS Atomic Internal model presented in Fig. 4 based on the DEVS MOF metamodel is presented in the following. According to Fig. 5 (arrow 6), the *DEVS Atomic Internal Model* defined as a stereotype of a State Machine diagram included in UML2 MOF is transformed to entities *INTERNAL_TRANSITION_FUNCTION*, *OUTPUT_ FUNCTION* and *TIME_ADVANCE_FUNCTION* of the DEVS MOF, representing related DEVS functions. Each of them is constructed using information included in the corresponding State Machine diagram. *Transitions* are used to define all functions. *CONDITIONAL_FUNCTION* elements are described by the *origin state*, while *TRANSITION_ FUNCTIONS* corresponding to a *CONDITIONAL_FUNCTION* are described by the *destination state*.

In the case of *INTERNAL_TRANSITION_FUNCTION* element, a STATE_VARIABLE_UPDATES element is also added to indicate state variable modification, as indicated by DEVS Var Modification Action associated to the Transition. In the case of Processor system (Fig. 4) the state variable Job, of type integer, is decreased by 1.

```
<INTERNAL_TRANSITION_FUNCTION>
  <CONDITIONAL_FUNCTION>
    <STATE_CONDITION text="busy"/>
    <TRANSITION_FUNCTION>
      <NEW_STATE text="idle"/>
      <STATE_VARIABLE_UPDATES>
        <STATE_VARIABLE_UPDATE name="job">
          <VALUE value="Integer(-1)"/>
        </STATE_VARIABLE_UPDATE>
      </STATE_VARIABLE_UPDATES>
    </TRANSITION_FUNCTION>
  </CONDITIONAL_FUNCTION>
</INTERNAL_TRANSITION_FUNCTION>
<OUTPUT_FUNCTION>
  <CONDITIONAL_OUTPUT_FUNCTION>
    <STATE_CONDITION text="busy">
      <SEND port="out">
        <STATE_VARIABLE_VALUE name="job"/>
      </SEND>
    </STATE_CONDITION>
  </CONDITIONAL_OUTPUT_FUNCTION>
</OUTPUT_FUNCTION>
<TIME_ADVANCE_FUNCTION>
  <CONDITIONAL_TIME_ADVANCE>
    <STATE_CONDITION text="busy"/>
    <TIME_ADVANCE>
      <VALUE type="Real" value="3.5"/>
    </TIME_ADVANCE>
  </CONDITIONAL_TIME_ADVANCE>
  <CONDITIONAL_TIME_ADVANCE>
    <STATE_CONDITION text="idle"/>
    <TIME_ADVANCE>
      <VALUE type="Real" value="Infinity"/>
    </TIME_ADVANCE>
  </CONDITIONAL_TIME_ADVANCE>
</TIME_ADVANCE_FUNCTION>
```

Fig. 6 illustrates a screenshot of the MediniQVT transformation tool, used for the execution of the QVT transformation. The first transformation part is illustrated (arrow 1).

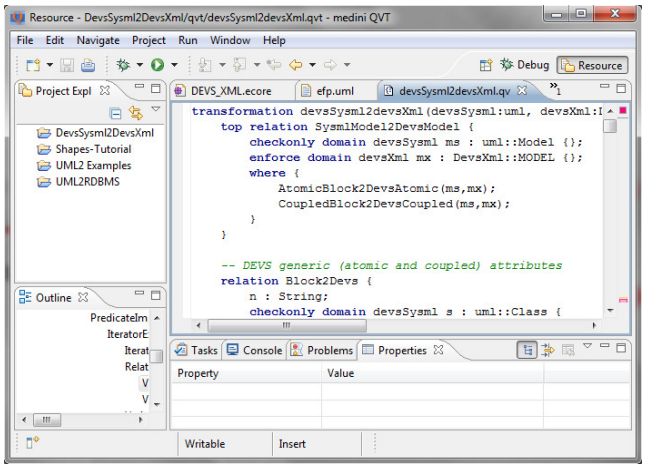

Figure 6: UML2 to DEVS MOF transformation with MediniQVT

5. EXECUTING GENERATED CODE WITH XSLC SIMULATION ENVIRONMENT

Following the steps presented so far, DEVS models are automatically generated from SysML system models. Based on the proposed methodology (Fig. 1), DEVS XMI models should be transformed to be executed in specific DEVS simulation environment in the appropriate target format (XML, Java code, etc.).

Many DEVS simulation environments aim at XML-based DEVS modeling and interpretation in different programming languages ([23], [22], [17], [24], [25]). An XML data encapsulation is accomplished in [23], within the DEVS environment, as a unifying communication method among the entities in any Systems-of-Systems (SoS) architecture. In [24] 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 Document Type Definitions (DTDs). In this case, model behavior is not emphasized. In [17] 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.

Hence, XLSC DEVS ([17]) was selected as the DEVS Simulator and DEVS XMI to XLSC XML transformation, as a

proof of concept. XLSC simulation environment requires an XML document of a specific structure for each DEVS component of the model. Therefore, the DEVS model (XMI format) is syntactically transformed to XLSC XML by XSLT. Since the two formats have small syntactical differences, the XLSC XML file is constructed with the help of a set of XSLT templates that match DEVS XMI elements, from which the needed values are taken and placed in the XLSC elements.

In general, DEVS XMI elements are transformed to similar XLSC elements. However, one should note the following issues:

- In XLSC there is no definition of the state set. Only state variables are defined.
- In XLSC all DEVS atomic functions (internal transition, output, time advance, external transition) are defined in a low level, procedural manner. In the corresponding XSLT we basically build the procedural XLSC elements that implement the behavior, declared in the respective DEVS model elements.
- Coupling is represented in a more simple and flat way, in XLSC. Couplings are not distinguished in internal or external (input and output). For each coupling, a source (component, port) and a target (component, port) are specified. When the source or target component is a DEVS coupled model, then the component attribute is set to "this". Otherwise, the name of the (component) DEVS model is used.

EditX is the XML tool that has been used for transformation of DEVS XMI models to XLSC code. It provides XSLT execution capabilities and is available in a free edition for non-commercial use. The defined XSLT transformations are rather simple. The generated XLSC code is passed to the XLSC interpretation environment, which dynamically creates DEVSJava classes for the atomic and coupled DEVS models. The simulation model can be executed in DEVSJava simulation environments. Fig. 7 shows simulation execution of the EFP system, defined using SysML in the MagicDraw tool, in the SimView DEVSJava component of the XLSC environment.

6. CONCLUSIONS

This paper presented a methodology that transforms system models already defined in SysML to DEVS executable models in a fully automated fashion. The properties of SysML system models have been enriched with simulation specific capabilities in order to generate executable discrete event simulation models based on DEVS formalism. The proposed methodology was tested to provide automated generation of DEVS executable code for the XLSC DEVS simulator. Each required step was defined, implemented and tested as well. The presented methodology is based on MDA concepts, implemented in an open, standard based and extensible framework.

Future work involves testing the profile and transformation code in real-world case studies involving complex system models, integrate additional DEVS simulators and provide additional capabilities within DEVS profile to enable the system engineer to integrate simulation results within the SysML system model.

7. ACKNOWLEDGEMENTS

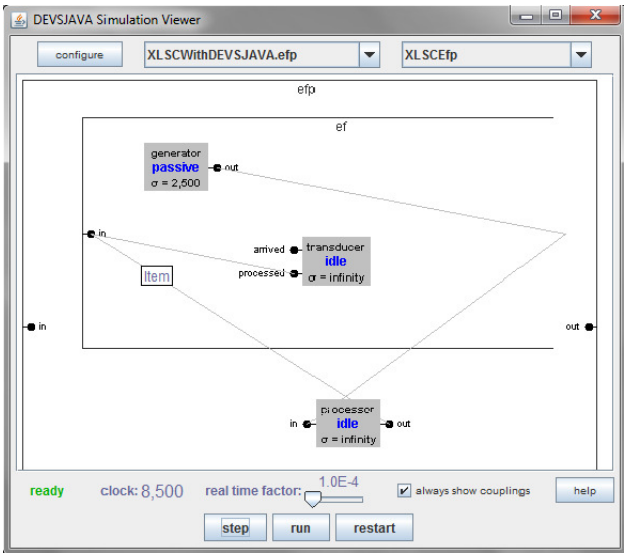

Figure 7: EFP simulation execution in XLSC DEVSJava

The authors would like to thank Nicolas Meseth, Patrick Kirchhof, and Thomas Witte for their valuable help. Not only they provided us with their XLSC prototype interpreter, but also they eagerly answered every question we posed.

8. REFERENCES

[1] Loyd Baker, Paul Clemente, Bob Cohen, Larry Permenter, Byron Purves, and Pete Salmon. *Foundational Concepts for Model Driven System Design*. INCOSE Model Driven System Design Interest Group, International Council on Systems Engineering, July 2000.

[2] OMG. Systems Modeling Language (SYSML) Specification. Version 1.0. September 2007.

[3] Edward Huang, Randeep Ramamurthy, and Leon F. McGinnis. System and simulation modeling using sysml. In *WSC '07: Proceedings of the 39th conference on Winter simulation*, pages 796–803, Piscataway, NJ, USA, 2007. IEEE Press.

[4] Oliver Schonherr and Oliver 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.

[5] R.S. Peak, R.M. Burkhart, S.A. Friedenthal, M.W. Wilson, M. Bajaj, and I. Kim. Simulation-based design using sysml part 1: A parametrics primer. In *INCOSE Intl. Symposium*, pages 1–20, San Diego, CA, USA, 2007.

[6] Russell Peak, Christiaan J.J. Paredis, and Diego R. Tamburini. The composable object (cob) knowledge representation: Enabling advanced collaborative engineering environments (cees), cob requirements & objectives (v1.0). Technical report, Georgia Institute of Technology, Atlanta, GA, Oct. 2005.

[7] Diego R. Tamburini. Defining Executable Design &

Simulation Models using SysML. *Available online via http://www.pslm.gatech.edu/topics/sysml/*, March 2006.

[8] Christiaan J. J. Paredis and Thomas 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.

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

[10] Renzhong Wang and C.H. 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.

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

[12] Bernard P. Zeigler and Hessam S. Sarjoughian. *Introduction to DEVS Modeling and Simulation with JAVA. DEVSJAVA Manual*, 2003.

[13] Gabriel A. Wainer and Norbert Giambiasi. *Timed Cell-DEVS: modelling and simulation of cell spaces*, chapter 10. H. Sarjoughian, F. Cellier Eds., Springer-Verlag, 2001.

[14] Ming Zhang, Bernard P. Zeigler, and Phillip Hammonds. Devs/rmi-an auto-adaptive and reconfigurable distributed simulation environment for engineering studies. *International Test and Evaluation Association Journal*, 27(1):49–60, 2005.

[15] Saurabh Mittal, José L. Risco-Martín, and Bernard P. Zeigler. Devs/soa: A cross-platform framework for net-centric modeling and simulation in devs unified process. *Simulation*, 85(7):419–450, 2009.

[16] Mara Nikolaidou, Vassilis Dalakas, Loreta Mitsi, Georgios-Dimitrios Kapos, and Dimosthenis 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.

[17] Nicolas Meseth, Patrick Kirchhof, and Thomas 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.

[18] OMG. Model Driven Architecture. Version 1.0.1. *Available online via http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf*, June 2003.

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

[20] World Wide Web Consortium (W3C). Extensible stylesheet language transformations (xslt). *Available online via http://www.w3.org/TR/xslt20*, 2007.

[21] OMG. UML 2.0 OCL Specification. *Available online via http://www.omg.org/docs/ptc/03-10-14.pdf*, October 2003.

[22] José Luis Risco-Martín, Saurabh Mittal, M. A. López-Peña, and Jesús Manuel 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.

[23] Matthew Hosking and Ferat 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.

[24] Saurabh Mittal, José L. Risco-Martín, and Bernard P. Zeigler. Devsml: Automating devs execution over soa towards transparent simulators. In *DEVS Symposium, Spring Simulation Multiconference*, pages 287–295. ACIMS Publications, March 2007.

[25] José L. Risco-Martín, Jesús M. De La Cruz, Saurabh Mittal, and Bernard P. Zeigler. eudevs: Executable uml with devs theory of modeling and simulation. *Simulation*, 85(11-12):750–777, 2009.