A SysML Profile for Classical DEVS Simulators

Mara Nikolaidou Vassilis Dalakas Loreta Mitsi George-Dimitrios Kapos
Dimosthenis Anagnostopoulos

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

Abstract

Discrete event simulation specification (DEVS) is a formalism facilitating hierarchical and modular description of the models executed using DEVS simulators. Lack of standardized, easy-to-use interface enabling simulation practitioners to define their models is an important drawback, since in most cases DEVS models are defined as C++ or Java programs based on existing simulator-specific DEVS libraries. Standard MDA concepts can be applied for the construction of DEVS models executed in different programming environments. DEVS models can be defined using DEVSML, a platform-independent, XML-based format. SysML is proposed as a standardized, graphical representation language of DEVS models stored in DEVSML, consequently transformed into executable code for existing DEVS Simulators, as DEVSJava and DEVSim++. The first step toward this endeavor, is the formal definition of the DEVS SysML profile proposed in this paper.

1 Introduction

The DEVS (Discrete Event System Specification) formalism [9] provides a conceptual framework for specifying discrete event simulation models. Using DEVS, discrete event models can be described in a modular and hierarchical form. Nowadays, a variety of software tools and simulators is available [10, 3], such as *DEVJava* or *Parallel DEVS*, that execute DEVS models, improving the security of the simulations, reducing the testing time and increasing productivity, hence, offering at the simulation community the ability to control and validate the overall simulation process and simulation results. Though, simulationists must represent DEVS model as programming code using predefined libraries offered by each simulator. In [5], DEVSML, a platform-independent, XML-based format, for describing DEVS executable code is proposed independently of the underlying simulator. DEVSML is consequently transformed into executable code for existing DEVS Simulators, as DEVSJava and DEVSim++, using translators as the ones proposed in [5] for DEVSJava simulators.

There is an ongoing effort to combine UML with DEVS to build simulation models. A mapping between DEVS models and UML state charts has been introduced in [8]. A mathematical proof, mapping DEVS to UML, has been presented in [11]. In [2], the representation of atomic DEVS models using UML sequence diagrams is proposed. However, these papers focus on exploring the circumstances under which models defined using DEVS and UML can be equivalent. In [1], an attempt has been undertaken to develop DCharts as a graphics language for DEVS models. DCharts is a UML-like language that does not follow any UML standard.

The Systems Modeling Language (SysML) is a general-purpose graphical modeling language that customizes UML, for systems engineering applications. 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 was adopted by the OMG as a standard in July 2006 [7].

We argue that SysML is more suitable than UML for the graphical representation of DEVS models, since SysML language and especially block diagrams provide for the natural representation of DEVS model decomposition. SysML could serve as a standardized, easy-to-use, graphical method to define DEVSML models, that can be consequently executed in existing DEVS Simulators, as the popular DEVSJava.

Among the advantages of such an endeavor are: SysML is based on UML that is widely known and easy-use.

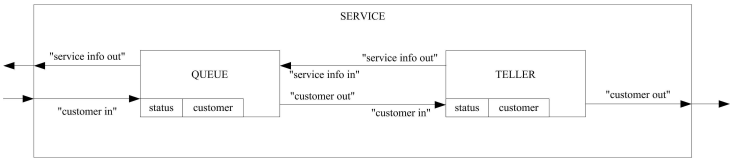

Figure 1. DEVS model paradigm

SysML offers both conceptual modeling and visual representation. SysML models may lead to automated code generation. There is a wide range of software tools supporting UML and SysML.

The formal method, proposed by OMG to extent or to restrict UML/SysML models, is the definition of a profile [6], emphasizing the use of UML/SysML to describe a specific "world", as the DEVS formalism. Since SysML profiles are based on formal UML extension mechanisms, they can be implemented in any standard UML modelling tool, such as MagicDraw [4], providing automated code generation for DEVS simulators, in DEVSML format.

In the following, we propose a SysML profile supporting the description of DEVS simulation models. Our aim is a) to offer a graphical, standardised environment for the definition of DEVS models using the proposed profile and b) if the profile is embedded in a UML modelling tool, to be able to generate code for DEVS simulators. The generated code should correspond to DEVSML model definitions forwarded to DEVS simulators.

The paper is structured as follows: A brief description of DEVS formalism and simulation tools is provided in section 2. The scope of DEVS SysML profile and the overall proposed framework is presented in section 3. In section 4, the definition of DEVS profile is analytically described, along with a simple example to demonstrated provided functionality. Conclusions and future work reside in section 5.

2 Classical DEVS Review

Two types of models are defined in DEVS: *atomic models* (behavioral representation), from which larger ones are built and describe basic model functionality, and *coupled models* (structural representation) expressing how basic models are connected in a hierarchical form. The formal definition of DEVS formalism can be found in [9].

Coupled models consists of other DEVS models (coupled or atomic). Models communicate through input and output ports properly interconnected. Fig. 1 depicts a simple example of a DEVS coupled model. SERVICE model is defined as the coupling of two atomic models: QUEUE and TELLER. Coupling is described by the definition of the correspondence among input and output ports of the components and the coupled model. Atomic models are described

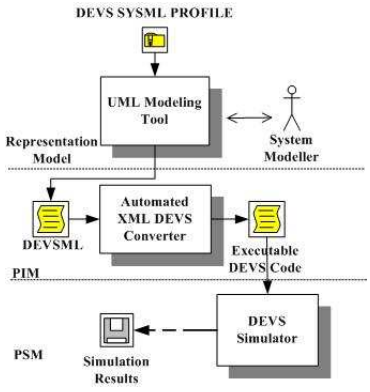

Figure 2. Proposed Simulation Environment

by corresponding model states and the transitions between them. Four functions are used to describe their behavior. Internal transition function specifies the next state to which the system will transit. External transition function 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 generates an external output just before an internal transition occurs. Time advance function controls the timing of internal transitions. In Fig. 1, TELLER atomic model waits for a customer in "idle" state. Upon receiving a customer in the input port "customer in", TELLER changes its status in "busy" state and increases the variable "customer". Service time is an exponentially distributed random variable. When the service is finished, it sends customer to the output port "customer out" and "ready" message to the QUEUE informing its state.

DEVS simulators facilitate system modellers to generate simulation code in one-to-one correspondence with DEVS formalism. The code imported in DEVS simulators consists of DEVS entity declarations (structural and behavioural). All DEVS model entities are defined as ancestors of predefined class hierarchy provided in DEVS libraries. Messages manage transmissions of events between models, defining the method for sending output events and receiving the incoming events. Coupled models support methods for defining ports and the coupling between them. Atomic models, besides methods for handling structural information, for example ports or state variables, also include methods for the description of system behavior. The implementation of methods for handling structural information is provided in DEVS libraries. The modeler has to specify the implementation of methods corresponding to internal transition, external transition, output and time advance functions, namely InTransFn, ExtTransFn, OutputFn and Ta, which are model-related and thus can not be predefined. In most cases, system modeler has to write himself object-oriented code in C++ or Java using DEVS libraries.

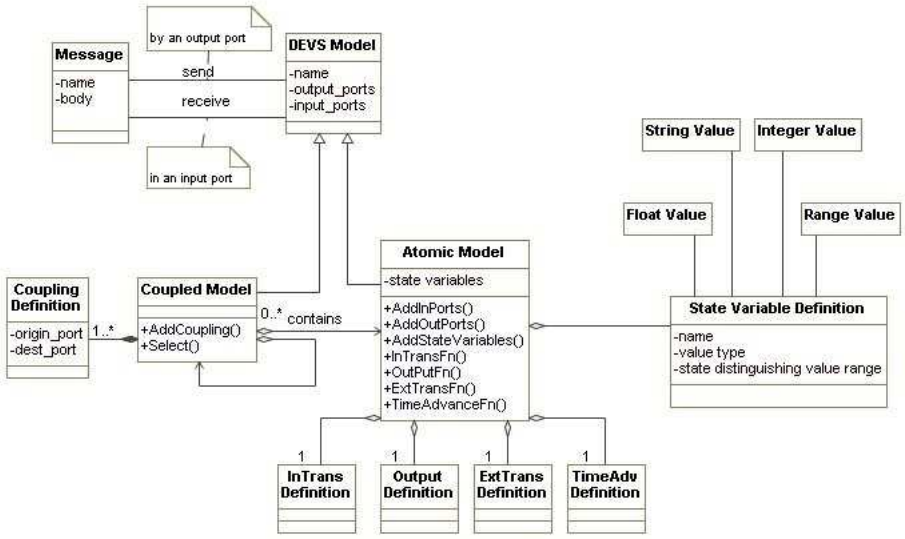

Figure 3. Basic DEVS meta-model

3 Scope of DEVS SysML Profile

An important drawback of DEVS formalism is the lack of a standardized, easy-to-use interface facilitating system modelers to define simulation models for object-oriented DEVS simulators, independently of their internal characteristics and implementation language. SysML may provide a standardized, easy-to-use, graphical environment for defining DEVS models, that can be consequently executed in existing DEVS Simulators. In such a case, the modeler will complete the following steps according to MDA concepts:

- The DEVS simulation model is easily defined in SYSML using a popular UML modeling tool and the corresponding profile.
- A platform independent model (PIM according to MDA) of DEVS simulation code is generated in DEVSML [5].
- DEVSML code is automatically translated into corresponding code in C++ or Java (platform specific model – PSM) and executed in the corresponding DEVS simulator.

The overall process is depicted in Fig. 2.

Within the profile, all discrete DEVS entities should be described in an object-oriented fashion, while common DEVS simulator class hierarchy should also be taken into account. A meta-model for classical DEVS description is presented in Fig. 3. DEVS SysML profile must provide for the description of all the entities included in the meta-model.

4 DEVS SysML Profile

In the following, we discuss the SysML profile defined for classical DEVS formalism. The main purpose of DEVS SysML profile is to facilitates simulationists to easily described both structural and behavioral properties of DEVS models by combining alternative SysML diagrams depicting in detail different aspects of atomic and coupled models. Special attention has been paid to the description of behavioral properties, e.g. atomic model functions, to minimize corresponding programming effort.

DEVS model entities are defined as stereotypes of SysML entities, while constraints are used to restrict SysML semantics to DEVS formalism.

4.1 DEVS Coupled Model

In the context of DEVS coupled models (DEVS CM), emphasis is given on constituting models (atomic or coupled), their interconnections through connection points, called ports, and compositional capability. SysML blocks are naturally selected for representing DEVS models, since both share the same basic properties, as ports.

SysML Block Definition Diagrams (BDDs) provide the means to define system composition, thus overall DEVS simulation 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 Block Definition Diagrams (BDD) may represent which are

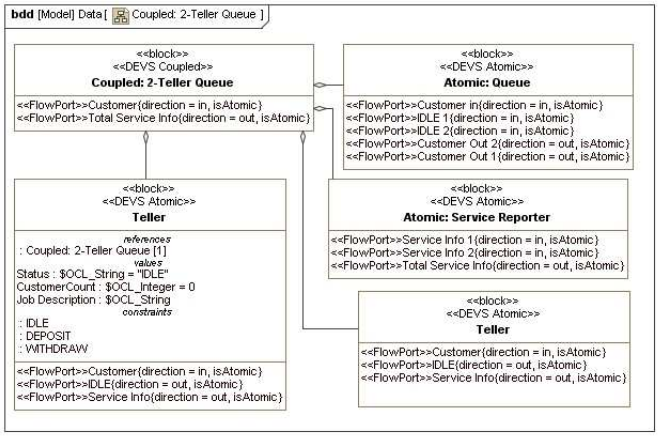

Figure 4. 2-Teller Queue Overall DEVS model

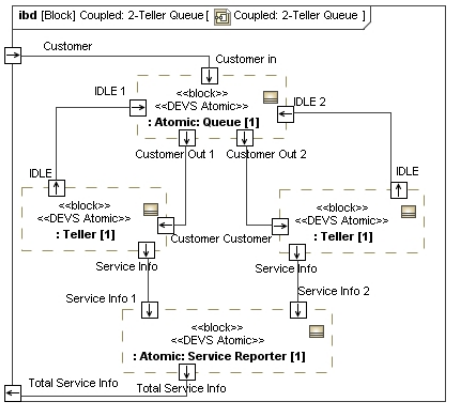

Figure 5. 2-Teller Queue DEVS Coupled Internal model

the components of each block, they do not depict how components are interconnected. This is thoroughly defined in Internal Block Diagrams (IBDs). Thus, one IBD must be defined for each DEVS CM, to describe coupling between constituting DEVS models.

Fig. 4 depicts the SysML representation of the coupled DEVS model described in Fig. 1. The specific model is rather simple, consisting of four atomic models. Fig. 5 depicts the corresponding IBD for coupled *2-Teller Queue* model. Both atomic and coupled DEVS models are represented as stereotypes of the SysML Block element, namely DEVS AM and DEVS CM. Constraints are defined to depict the relationships between DEVS entities (as depicted in Fig. 3) and restrict SysML BDD and IBD functionality to effectively correspond to DEVS formalism. DEVS CM is only described by input and output ports. No other SysML block properties are used. Stereotypes defined for coupled DEVS models reside in Table 1.

Flow ports of blocks are used for DEVS ports representation, both input (depicted as inbound flow ports) and output (depicted as outbound flow ports). No additional semantics were required. DEVS ports are depicted in DEVS BDD. They way they are related is defined within a DEVS Coupled Internal Model corresponding to a SysML IBD diagram, as indicated in the following (Fig. 5):

1. Input ports of the external DEVS coupled block (2-Teller Queue) connect to input ports of the contained DEVS blocks. Similarly, output ports of the external DEVS coupled block connect to output ports of the contained DEVS blocks.

2. All other connections (between internal DEVS models, either DEVS CM or DEVS AM) are made through opposite-directed flow ports.

4.2 DEVS Atomic Model

Defining a DEVS atomic model 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.

DEVS Atomic models are defined as stereotypes of blocks (DEVS AM entity of Table 1) in the DEVS Model BDD. For each DEVS AM input and output ports are defined. Atomic models behavior is defined as transitions between discrete model states [9]. 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.

To describe atomic model behavior, four sub-diagrams must be related to each DEVS AM included in DEVS Model BDD:

- 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.

All four are discussed in the following.

Table 1. DEVS Model Structural Stereotypes

DEVS Stereotype	SysML Entity	Constraints
DEVS Model	Block Definition Diagram	There is one BDD containing the overall model. Only DEVS CM and DEVS AM entities participate in this diagram.
DEVS Coupled Internal Diagram	Internal Block Diagram	A DEVS Coupled Internal Diagram must be associated to any DEVS CM. The diagram may only contain the DEVS models directly used by the corresponding DEVS CM. All flow ports must be connected in an appropriate manner (output to input flow port).
DEVS CM	Block	A DEVS CM may only have property parts and unidirectional (in or out) flow ports. Every DEVS CM is associated to a DEVS Coupled Internal Diagram.
DEVS AM	Block	A DEVS AM may only have unidirectional (in or out) flow ports, value properties and constraints on the value properties. Four sub-diagrams must be associated to each DEVS AM to describe its behavior: DEVS States Definition, DEVS States Association, DEVS Atomic Internal and DEVS Atomic External.

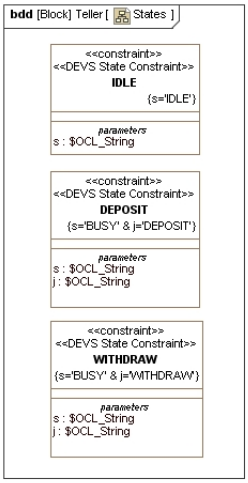

Figure 6. Teller DEVS States Definition model

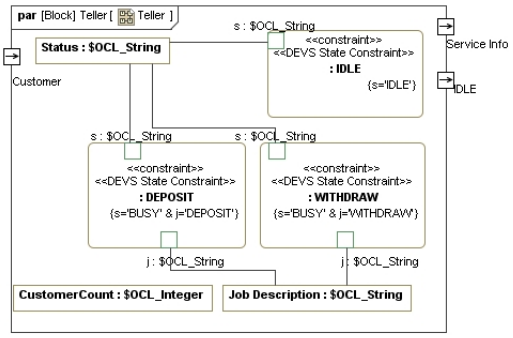

Figure 7. Teller DEVS States Association model

4.2.1 States Definition

DEVS atomic model set of states are defined as independent constraints based on corresponding state variables. Thus, SysML constraint BDD and corresponding SysML PDs are utilized for the definition of DEVS atomic model states.

As shown in Fig. 6, state variables are represented as constraint blocks in DEVS States Defitinion diagram. Each state variable is associated with one or more DEVS AM values. In the corresponding SysML PD, named DEVS State Association diagram. Corresponding stereotypes reside in Table 2.

Fig. 7 illustrates the DEVS State Association diagram of *Teller* atomic model. Using this diagram, discrete model states can be defined by combining discrete values of the state variables defined in associated DEVS AM. Consequently, discrete model states can be automatically inserted in DEVS Atomic Internal Diagram, corresponding to Internal Transition, Output and Time Advancement functions, discussed in the following paragraph.

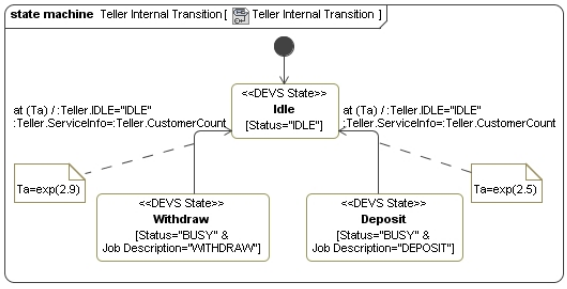

Figure 8. Teller DEVS Atomic Internal model

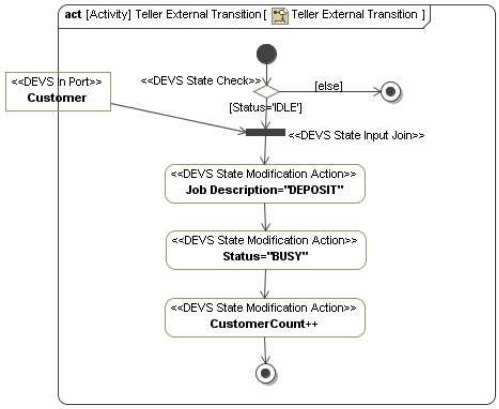

Figure 9. Teller DEVS Atomic External model

Table 2. DEVS States Definition Stereotypes

DEVS Stereotype	SysML Entity	Constraints
DEVS States Definition Diagram	Block Definition Diagram	It must be associated to a DEVS Atomic Block. The diagram may only contain DEVS State Constraints.
DEVS States Association Diagram	Parametric Diagram	It must be associated to a DEVS Atomic Block. The diagram contains the block's value properties, the DEVS State Constraints (defined in DEVS States Definition Diagram) and their interconnection. Each constraint parameter must be connected to a value property.
DEVS State Constraints	Constraint	It may have as many parameters as the number of state variables. The type of each parameter must be compatible to a subset of the state variable's type. The value of each parameter must be constrained.

4.2.2 DEVS Atomic Internal Diagram

State diagrams are used for the definition of the internal transition function. DEVS states are computed based on State Definition diagram and 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 transition occurs. The Time Advancement function is represent by a corresponding note associated to each DEVS state transition. 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 corresponding diagram for Teller atomic model is depicted in Fig. 8.

4.2.3 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 arrival time of the input event. Thus, initially there is a decision node checking current state variable values and creating different control flows.

There are many inputs and state conditions that need to be combined in order to define the distinct (input-state) conditions that determine the effect of the external transition function, thus a DEVS State Fork is used for every state condition that needs to be combined with many inputs to form different effects.

Combination of a state condition and one or more inputs is made with a DEVS State Input Join. Finally, each join is followed by a series of DEVS State Modification Actions, that perform value assignments to the state variables of the atomic DEVS model. In Fig. 9, Teller atomic model has only one input, so there is no need for a DEVS State Fork.

DEVS SYSML profile has been implemented using Magic Draw [4] 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. Code generation in DEVSML is currently under development.

5 Conclusions – Future Work

An important drawback of DEVS simulators is the lack of a standardized, easy-to-use interface facilitating system modelers to define simulation models independently of the specific DEVS programming environment. To this end, we proposed to adopt MDA concepts in DEVS model development. Code generation in DEVSML, an XML based format which may serve a platform independent DEVS specification language is currently under development. The first step towards this endeavor, is the formal definition of the DEVS SYSML profile presented in this paper.

References

[1] H. Feng. Dcharts, a formalism for modeling and simulation based design of reactive software systems. *Master Thesis, McGill University*, February 2004.

[2] S.-Y. Hong and T. G. Kim. Embedding UML subset into object-oriented DEVS modeling process. In *Proceedings of SCSC 2004*, pages 161–166, San Jose, CA, July 2004.

[3] T. G. Kim. *DEVSim++ © User's Manual. C++ Based Simulation with Hierarchical Modular DEVS Models*, 1998.

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

[5] 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.

[6] 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.

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

[8] S. Schulz, T. C. Ewing, and J. W. Rozenblit. Discrete event system specification (DEVS) and statemate statecharts equivalence for embedded systems modeling. In *Proceedings of 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 308–316, April 2000.

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

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

[11] D. Zinoviev. Mapping DEVS models onto UML models. In *Proceedings of the 2005 DEVS Integrative M&S Symposium, SpringSim05*, pages 101–106, San Diego, CA, April 2005.