TOWARDS DEVS META LANGUAGE

Vladimír Janoušek
Petr Polášek
Pavel Slavíček
Faculty of Information Technology,
Brno University of Technology
Božetěchova 2
612 66 Brno, Czech Republic
E-mail: {janousek|polasek|slavicek}@fit.vutbr.cz

KEYWORDS

DEVS formalism, simulation model, meta-model, model specification, XML, XSL transformation

ABSTRACT

The aim of this article is to present DEVSML (*DEVS Meta Language*) that is intended for implementation of simulation models based on the DEVS formalism. A model implemented by DEVSML is independent of the concrete simulation environment implementation. Furthermore, a model can be simply and automatically transformed and used by potentially whatever DEVS-based simulation environment, such as DEVS/C++, DEVSJava, etc. Since models are generally not portable between different simulation environments, it is necessary to implement models for every framework, which costs a lot of time and resources. DEVSML eliminates this annoyance and enables creation of models, which can be shared between different simulation environments. DEVSML defines model's structure as well as behavior by XML (eXtensible Markup Language). Transformation of models is based on the XSL transformation. For every simulation environment, an XSL template needs to be created and used for the transformation.

INTRODUCTION

The interest in modeling and simulation is still increasing, since modeling and simulation is used in many areas of human activity and cannot be substituted during design and analysis of complex systems. Actually, there is a pragmatical trend to group models and simulators into tightly linked packages. While creating a new simulation environment, basic models are reconstructed and reimplemented, since they cannot be shared among different frameworks. It seems to be desirable to specify a standard for implementations of models, which will put away this annoyance and allow migration of models between different environments, saving time and effort spent on implementation. Then the creation of libraries consisting of commonly used components and models will be possible.

The article is focused on the DEVS formalism (*Discrete EVent System specification*) (Zeigler et al. 2000) representing a formal basis for specification of discrete event systems. In theory, the DEVS models are independent of the chosen simulator as well as of the experimental frame. Actually, many implementations of this formalism exist, such as DEVS/C++, DEVSJava, etc. Their main disadvantage is impossibility of sharing models between them because the models are usually implemented in the same language as simulator itself.

The aim of this article is to present our approach trying to solve this problem. We are developing a meta-language for description of DEVS models that is based on XML. Models described by this language can be simply transformed to different simulation environments and frameworks without needs of their changes. Furthermore, to simplify the implementation, we are developing a prototype of a modeling tool, based on this language, which enables us to graphically specify a model with the definition of transition and transformation functions of atomic models

The paper is organized as follows. The first section shortly reviews basic components and terms of DEVS formalism. Next section delimitates the areas of our work with respect to the DEVS standardization group. It deals with existing tools and discusses their advantages and disadvantages. The main parts of the article are the last two sections, describing the DEVSML and shortly reviewing our modeling tool based on this meta-language. The advantages our approach gives and future plans are discussed in the conclusion.

THE DEVS FORMALISM

The DEVS formalism specifies discrete event systems hierarchically. We start from the atomic models, from which larger coupled models are built. An atomic model M is defined as:

$$M = \langle S, ta, \delta_{\text{int}}, X, \delta_{ext}, Y, \lambda \rangle$$

where

S is the sequential state set,

ta: $S \rightarrow R^{+}_{0,\infty}$ is the time advance function,

$\delta_{\text{int}} : S \rightarrow S$ is the internal transition function,

X is the set of external input event types,

$\delta_{ext} : Q \times X \rightarrow S$ is the external transition function

where $Q = \{(s,e) | s \in S, 0 \le e \le ta(s)\}$,

Y is the set of external event types generated as output,

$\lambda : S \to Y$ is the output function.

The coupled model can contain atomic models and also coupled models, connected together. This model can be inserted as a component in another coupled model, forming a hierarchical structure. A coupled model C is defined as:

$$C = \langle X_{self}, Y_{self}, D, EIC, EOC, IC, select \rangle$$

where

X_{self} is the set of external input events,

Y_{self} is the set of output events,

D is a set of DEVS component models,

EIC is the external input coupling relation,

EOC is the external output coupling relation,

IC is the internal coupling relation,

$select$ is a function, the tie-breaking selector.

More detailed descriptions for the definitions of DEVS models, their variants and abstract simulators can be found in (Zeigler et al. 2000).

EXISTING SYSTEMS AND TOOLS

There exist many implementations of DEVS formalism. Some of the most known are DEVSC++ (Zeigler et al. 1996), DEVSJava (Sarjoughhian and Zeigler 1998; Zeigler 1997) and we can also mention PythonDEVS (Bolduc and Vangheluwe 2001) among others. DEVSJava is considered as a reference implementation among them.

There exists an effort to develop a standard of the DEVS formalism. One team that is concerned with the standardization is the DEVS standardization group. Their research (Wainer 2005) can be divided into three basic areas: interoperability of existing DEVS tools (1), specification of the minimal 'kernel' for a tool to be DEVS compliant (2), definition and implementation of a language for atomic and coupled model definition and broadening the models between the community of users (3).

There exist some articles (Fishwick 2002) concerning the area (2) and proposing the manners of simulation models description. The description or model's structure is based on XML, the translation functions are dismissed or are described with a pseudo code.

There also exists a general modeling tool ATOM3 (Lara and Vangheluwe 2002) focused on meta-modeling and model-transforming. Meta-modeling refers to description or modeling different kinds of formalisms used to model systems. Model-transforming refers to the automatic process of transforming a model in a given formalism to another one (in the same or different formalism). The tool is based on graph grammars and uses graph rewriting for model transformation.

Our focus is the area (3). At least two similar projects are connected with this area. The first is DEVSW (Yung-Hsin and Lung-Hsiung 1998). In this project, the model's structure description is based on XML and the description of behavior of atomic models is specified by a pseudo code. An elegant solution represents the second project (Schäfer 2003) that uses XML only. The description of the functions is done by rules with a finite set of states so only finite state automata can be specified by this approach.

DEVSML

The primary motivation to develop the DEVSML was to enable portability of a model between our experimental simulation tools and other DEVS-based simulation frameworks. The portability of a DEVS model allows us to use advantages of complementary features of the other frameworks. However, portability is a general problem and its solution can introduce interesting possibilities and facilitate the modeler's effort.

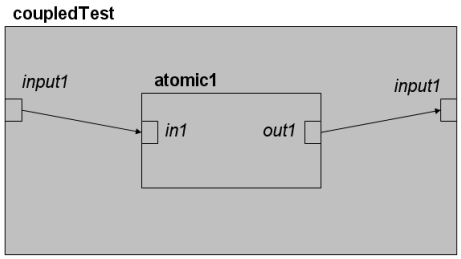
Structure of a Model in DEVSML

DEVSML (DEVS formalism Meta Language) is based on XML (extensible Markup Language). An XML document has a hierarchical structure so that the hierarchical structure of DEVS model can be mapped to XML easily. An XML document defining a coupled DEVS component comprises an input and output ports specification, a list of names of the inner components (i.e. atomic DEVS models) - with links to the documents containing their definitions - and a description of couplings of the components. Every DEVS component is defined in a separate (possibly detached) XML document, which can be stored in local files as well as anywhere on the internet. This way it is possible to create publicly available libraries of reusable components. An example of a coupled component defined by DEVSML is shown in Figure 1.

Figure 1 depicts also a definition of an atomic model. The definition of input and output ports of an atomic component is similar to the ones of a coupled component. A concept of a super model can be applied here to simplify the implementation by means of inheritance - the definition of ports, state variables, and functions can be inherited from a super-model.

As the definition of an atomic and a coupled DEVS model structure is quite simple, the definition of the atomic DEVS behavior (specified by the internal transition, the external transition, the output function and the time advance function) is a bit more problematic. We have got inspired by JavaML project (Badros 2000), which defines an XML representation of Java code. This way, the Java code is available for optimization, verification, and/or transformation. JavaML features perfectly fit to our needs here. That is why our notation for basic programming

constructs and expressions usable in DEVS functions specification came out directly from JavaML.




```
<coupled name="coupledTest" modelX="50" modelY="50">
  <ports>
    <input>
      <port name="input1"/>
    </input>
    <output>
      <port name="output1"/>
    </output>
  </ports>
  <D>
    <component name="atomic1" source="file://atomic1.xml"
       modelX="50" modelY="50"/>
  </D>
  <influences>
    <influence source="self" source-port="input1"
               target="atomic1" target-port="in1"/>
    <influence source="atomic1" source-port="out1"
               target="self" target-port="output1"/>
  </influences>
</coupled>

<atomic name="atomic1" modelX="50" modelY="50">
  <state-variables>
    <state-variable name="A" type="integer" initial-value=0/>
  </state-variables>
  <ports>
    <input>
      <port name="in1"/>
    </input>
    <output>
      <port name="out1"/>
    </output>
  </ports>
  <ta> ... <ta/>
  <internal-transition-function>
    ...
  </internal-transition-function>
  <external-transition-function>
    ...
   </external-transition-function>
   <output-function> ... </output-function>
</coupled>
```

Figure 1: A DEVSML Description of a Coupled and Atomic DEVS Model

In JavaML as well as in DEVSML there are defined basic elements of expressions such as integer values and unary and binary operators (such as greater than, equal, etc.). Basic syntax elements comprise getting and setting the value of state variable (getStateVar, setStateVar), the element which gets and sets value of local variable (getVar, setVar), conditional branch of program (if-then-else) and a cycle with condition on the beginning (while), at the end (until) and cycle with known number of passes (for).

Source Languages for DEVSML

The implementation of DEVS models directly in DEVSML is possible but not very handy. A more user-friendly language has to be chosen for modeling.

The hierarchical structure of a model and the coupling of the components can be specified graphically. This is shown in Figure 1. Such a visual language can be transformed to DEVSML quite simply. The graphical specification of models and their transformation is one of the main features of our experimental modeling tool we are developing. This is discussed in the next section.

What is more difficult is the behavior specification of the atomic DEVS. Some simple language or a pseudo code can be used for specification of the functions. Currently, we are experimenting with a Lisp-like language, which is shown in Figure 2.

LISP-LIKE CODE:

```
(if (> A 10)
      (for i from 0 to 5 do
              (set B[i] 0)))
(set A 5)
```

DEVSML:

```
<Code>
  <If>
    <Test>
      <BinOp operator=">">
        <Left>
          <GetVar name="A"/>
        </Left>
        <Right>
          <Literal value="10"/>
        </Right>
      </BinOp>
    </Test>
    <Code>
      <Loop kind="for" variable="i" start="0" end="5" step="1">
        <SetStateVar name="B[i]">
          <Literal value="0"/>
        </SetStateVar>
      </Loop>
    </Code>
    <Else/>
  </If>
  <SetVar name="a">
    <Literal value="5"/>
  </SetVar>
  <SetStateVar name="A">
    <Literal value="5"/>
  </SetStateVar>
</Code>
```

Figure 2: Lisp-like Code and Corresponding Representation in DEVSML

Besides a translator from such a source language to the DEVSML code it is essential to have a possibility to translate the DEVSML code to the source code as well. Then we can use DEVSML code as the only code, which have to be stored and maintained. We also suppose that there can be even more languages used as the user-friendly views on DEVSML-specified model.

To be more consistent with the graphical specification of DEVS components structure, one of the possibilities we are going to investigate comprise visual languages for expressing the DEVS functions. We have slightly experimented with a visual language based on the UML (France et al. 1997), especially the diagram of activity. An example of this approach is showed in Figure 3 (code from Figure 2 is used). Nevertheless we consider this approach to be very experimental.

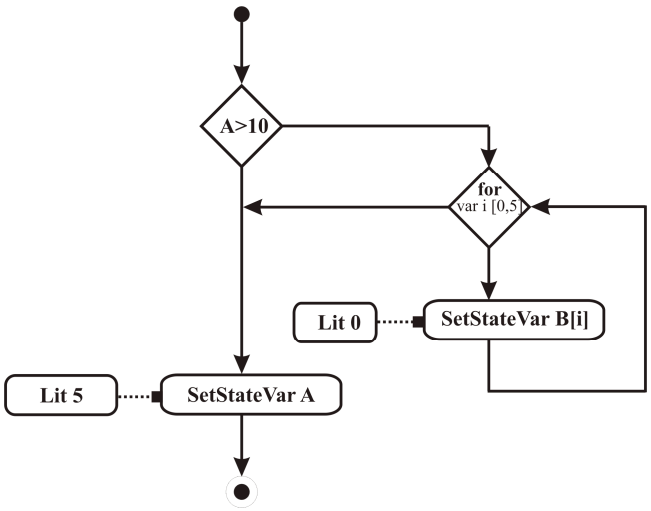


Figure 3: Graphical Language for Definition of DEVSML Functions

Transformation to Target Simulation Environment

DEVSML describes the structure of DEVS components and the semantics of functions describing the behavior of atomic components. It represents a basis for the transformation to the chosen simulation environment. The transformation is based on an XSL transformation with an appropriate XSL template that contains rules for the transformation. Such a transformation is illustrated in Figure 4. To add a support for a new environment, it is necessary to implement only a new XSL template, the models remain unchanged.

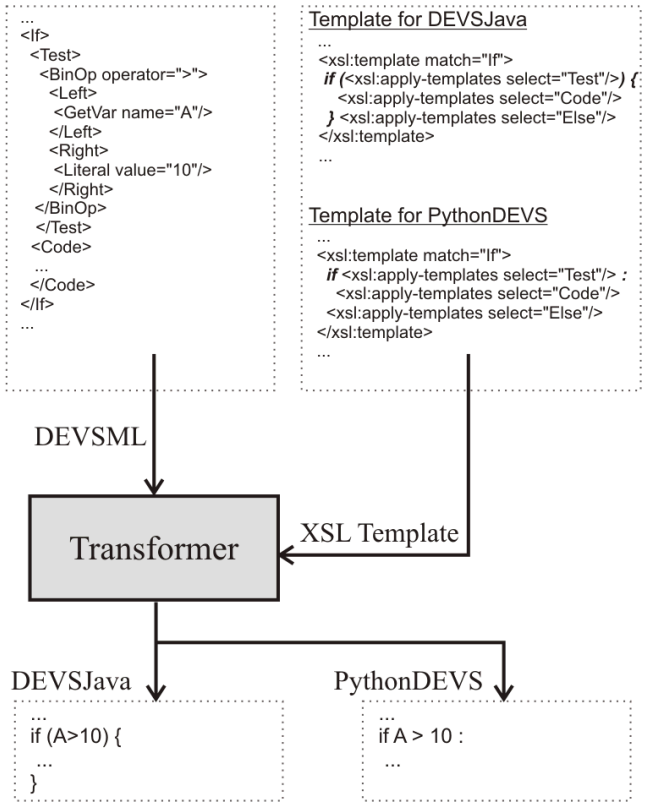


Figure 4: XSL Transformation

A MODELING TOOL

The prototype implementation of a DEVSML-based modeling tool is being implemented in the Java. The whole system has two main parts. The first part represents a graphical user interface used to create the structure of a model and a definition of its atomic models behavior. The second part of the system supports transformation of a model defined in DEVSML into an equivalent implementation for some simulation tool or environment.

The model structure editor allows for insertion of components whose definition can be stored in some local files, or somewhere on the internet. Furthermore it allows for the creation of new atomic components with the use of inherited properties of other atomic components. Among the atomic DEVS functions, the tool supports also user-defined functions that can be called from the atomic DEVS component's functions specifying its behavior. User defined functions are included as properties of atomic components.

The transformer is not just an implementation of XSL transformations. Its task is also to retrieve from internet or from files the necessary DEVSML definitions needed for the transformation and after that to link and to sort the results of the XSL transformation in a form acceptable by the chosen target simulation framework.

CONCLUSION

The article deals with sharing of simulation models among different DEVS-based simulation frameworks. We have presented a proposal of our solution - DEVSML. A model defined by DEVSML is independent of the particular implementation of the DEVS simulator. DEVSML specifies the hierarchical structure of model as well as the structure and the behavior of the atomic components by XML. The creation of models is simplified with the use of our modeling tool prototype. This experimental tool allows us to graphically specify the model structure and its behavior. As a future work, we are planning to extend it by support for more simulators and to implement new data types useful for implementation of atomic models behavior.

DEVSML puts away the dependence between the implementation of a model and a simulation environment. A model specified by DEVSML is portable to theoretically whatever DEVS-based simulation framework which is the main gain of this project. The benefits of portable models are very important. For example, the development of a model can be done in a sequential framework where testing is often easier. Afterwards, the model can be ported and simulated in a distributed or real-time simulation framework that could be more appropriate for some particular simulation studies. DEVSML can be used also for creating libraries of commonly used reusable models.

The next benefit of DEVSML is the possibility of automatical verification of new simulation environments, as suggested in (Wainer 2005). A verification can be done through a formal proof, which is obviously difficult. The

other method of verification is to simulate model in a reference simulation environment and than compare the results with simulation in the tested environment. DEVSML offers sharing of models between the tested and reference environment.

ACKNOWLEDGEMENT

This work has been supported by the Grant Agency of Czech Republic grant No. 102/04/0780 "Automated Methods and Tools Supporting Development of Reliable Concurrent and Distributed Systems".

REFERENCES

Badros, G. 2000. “JavaML: A Markup Language for Java Source Code.” *Proceedings of the 9th International World Wide Web Conference* (Amsterdam, Netherlands, May. 15-19), 159-77.

Bolduc, J. and H. Vangheluwe. 2001. “The Modelling and Simulation Package PythonDEVS for Classical Hierarchical DEVS.” MSDL Technical Report MSDL-TR-2001-01. Modelling, Simulation & Design Lab, McGill University. (Feb).

Filippi, J. B.; F. Bernardi; and M. Delhom. 2002. “The JDEVS Modelling and Simulation Environment.” *Proceedings of the 1st Biennial Meeting of the iEMSs* (Lugano, Switzerland, Jun. 24-27). International Environmental Modelling and Software Society, 283-288.

Fishwick, P. 2002. “XML Based Modeling and Simulation: Using XML For Simulation Modeling.” *Proceedings of the 2002 Winter Simulation Conference: Exploring New Frontiers 2002* (San Diego, California, Dec. 8-11), 616-622.

France, R.; A. Evans; K. Lano; and B. Rumpe. 1997. “The UML as a Formal Modeling Notation.” *Proceedings OOPSLA'97 Workshop on Object-oriented Behavioral Semantics* (Atlanta, Georgia, Oct. 6). Munich University of Technology, 75-81.

Lara, J. and H. Vangheluwe. 2002. “Using AToM^3 as a Meta-CASE Tool.” *The 4th International Conference on Enterprise Information Systems* (Ciudad Real, Spain, Apr.), 642-649.

Sarjoughhian, H. S. and B. P. Zeigler. 1998. “DEVSJAVA: Basis for a DEVS-Based Collaborative M&S Environment.” *Proceedings of the 1998 SCS International Conference on Web-Based Modeling and Simulation* (San Diego, CA, Jan. 11-14), 29-36.

Schäfer, A. 2003. “Visualisierung und XML-Darstellung von DEVS-Modellen.” M.S. Thesis. Fakultät für Informatik, Universität der Bundeswehr München.

Vangheluwe, H.; J. Bolduc; and E. Posse. 2001. “DEVS Standardization: Some Thoughts.“ DEVS Standards Group meeting, *Winter Simulation Conference 2001* (Washington, DC, Dec. 11).

Vangheluwe, H. and J. Lara. 2002. “Meta-Models are Models too.” *Proceedings of the 2002 Winter Simulation Conference*, 597-605.

Wainer, G. 2005. “DEVS Standardization Study Group.“ Interim Final Report (Seattle, WA, Apr. 26). The SISO Standards Activities Committee (SAC).

Yung-Hsin, W. and W. Lung-Hsiung. 1998. “A Modeling and Simulation Example Using DEVSW.” *The 31st Annual Simulation Symposium* (Boston, MA, Apr. 5-9). IEEE Computer Society, 210.

Yung-Hsin, W. and L. Yao-Chung. 2002. “An XML-based DEVS Modeling Tool to Enhance Simulation Interoperability.” *The 14th European Simulation Symposium and Exhibition* (Dresden, Germany, Oct. 23-26), SCS Europe, 406-410.

Zeigler, B. P. and V. Sankait. 1993. “DEVS Formalism and Methodology: Unity of Conception/Diversity of Application.” *Proceedings of the 1993 Winter simulation conference*. ACM Press, New York, NY, USA, 573-579.

Zeigler, B. P.; Y. Moon; D. Kim; J.G. Kim. 1996. “DEVS/C++ A High Performance Modelling and Simulation Environment.“ *29th Annual Hawaii International Conference on System Sciences* (Maui, Hawaii, Jan. 3-6). IEEE Computer Society, 350-359.

Zeigler, B.P. 1997. “DEVS-JAVA User’s Guide.” Technical Report. AI & Simulation Lab. Department of Electrical and Computer Engineering. University of Arizona, Tucson. (Feb).

Zeigler, B. P.; H. Praehofer; and T. Kim. 2000. “Theory of Modeling and Simulation.” 2nd edition. Academic Press. ISBN 0-12-778455-1.

BIOGRAPHY

VLADIMÍR JANOUŠEK received the Ph.D. degree from the Faculty of Information Technology**,** Brno University of Technology in 1999. He is an assistant professor in the Department of Intelligent Systems at the Faculty of Information Technology**,** Brno University of Technology. His research focuses on simulation-driven developlement, pure object orientation and reflective architectures.

PETR POLÁŠEK received the M.S. degree from Faculty of Information Technology**,** Brno University of Technology in 2005. He is a Ph.D. student in the Department of Intelligent Systems at the Faculty of Information Technology**,** Brno University of Technology. His research focuses on meta-modeling.

PAVEL SLAVÍČEK received the M.S. degree from Faculty of Information Technology**,** Brno University of Technology in 2003. He is a last year Ph.D. student in the Department of Intelligent Systems at the Faculty of Information Technology**,** Brno University of Technology. His research focuses on meta-modeling, distributed simulations and multiagent systems.