A W3C XML Schema for DEVS Scenarios

J. L. Risco Martín[1], Saurabh Mittal[2], M. A. López-Peña[3] and J. M. de la Cruz[1]

[1]Departamento de Arquitectura de Computadores y Automática
Universidad Complutense de Madrid,
28040 Madrid, Spain
{jlrisco , jmcruz}@dacya.ucm.es

[2]Arizona Center of Integrative Modeling and Simulation
Electrical and Computer Engineering Department
The University of Arizona, Tucson, AZ 85721, USA
saurabh@ece.arizona.edu

[3]Sistemas Avanzados de Tecnología, S.A.
Av. Europa 34ª, 28023 Madrid, Spain
malopez@satec.es

Abstract
There are numerous DEVS-based simulators to configure DEVS systems and run them. Current programs or libraries are based in programming languages such as JAVA, C++, etc. The existence of these different implementations has resulted in the formation of various groups that are not able to share models and cannot capitalize on model reuse. To encourage system modeling and simulator compatibility we propose a standard representation for such models. Such a standard must be simple to manipulate and validate, and promote the integration of DEVS M&S software on different platforms. In this paper we present an XML Schema for representing the structure and the behavior of DEVS coupled scenarios. In addition, we developed a simulator (xDEVS) that can execute these XML-based scenarios over DEVS protocol. We also demonstrate how an Atomic DEVS behavior can be represented using XML by proposing the XML Schema of an atomic model as well. This paper is one of the first papers in the ongoing research on M&S using XML as the communication platform.

Keywords: DEVS (Discrete EVent System Specification), XML (eXtensible Markup Language), XML-Schema, Meta-Language, Meta-Model.

1 Introduction

The DEVS formalism (Discrete Event Systems Specification) [1] provides a way to specify discrete event systems as well as a base for distributed simulation environments. DEVS is a universal formalism for the discrete event dynamic systems and has been used to represent varied classes of dynamic systems. DEVS environments, such as DEVSJAVA, DEVS.C++, and others [2,4,5] are embedded in object-oriented implementations and they support the goal of representing executable model architectures in an object-oriented representational language. As a mathematical formalism, DEVS, is platform independent, and its implementations adhere to the DEVS protocol so that DEVS models easily translate from one form (e.g., C++) to another (e.g., Java) [3,4,5] With its recent advances like variable structure modeling, dynamic model reconfiguration, real-time simulation control and model continuity [6], DEVS is emerging today as the chosen way to employ M&S in design phases of any dynamic system.

There are varied libraries for expressing DEVS models across the globe, and all of them have efficient implementations for executing the DEVS protocol. However, this proliferation of libraries presents a difficulty for the modelers, in sharing models, who have to learn the programming language of the simulator and sometimes they remain tied to it. In the present work we aim to bridge this gap by employing XML as a means to share model information and a step towards model interoperability and reuse.

Section 2 describes an overview of our proposed solution. Section 3 describes the DEVS representation of scenarios. Section 4 explains why XML is an appropriate technology and a preferred choice for DEVS-XML representation among others, such as SESM/CM. Section 5 provides the DEVS-XML Schema definitions. Section 6 shows an example of a DEVS-XML atomic model generated from an XDEVS-Java atomic model. Finally, we present the conclusions and future work.

2 Proposed Solution

One way to increase modeler-simulator compatibility is to adopt a standard representation of a scenario. Here it is important to make a distinction between models and scenarios. A model is an abstract, symbolic representation of a DEVS system – also named behavior, that contains the finite state machine of a closed system. Specifically speaking, a model here is refered as an atomic model in DEVS terms. A scenario is an explicit description of this system that contains many of such atomic models and is called a coupled model in DEVS terms. A scenario does not contain the behavior of the modeled system but it contains information about the connectivity relationships of the various atomic models contained therein.

Using a standard representation of a scenario, using XML, the modeler may use this representation which does not require a lot of programming knowledge, making automatic transformations from the XML representation to a particular simulator representation. To facilitate such automated conversion, we developed a simulator called XDEVS [7], which executes XML coupled scenarios and translates our implementation of DEVS simulator. In order to accomplish this, we also developed a parser that allows transformation of the XDEVS-Java scenario into a DEVS-XML scenario.

Our proposed representation allows the behavior of the scenario to be specified in XML [8]. The behavior Schema for an atomic model is not yet completed, but in its current state we still can represent basic types of atomic models, for example, a generator. Consequently, we are looking for a 20-80 rule (20% of the effort, 80% of the models) which amounts to putting the effort in reusing the existing models Figure 1 shows the modeling process.

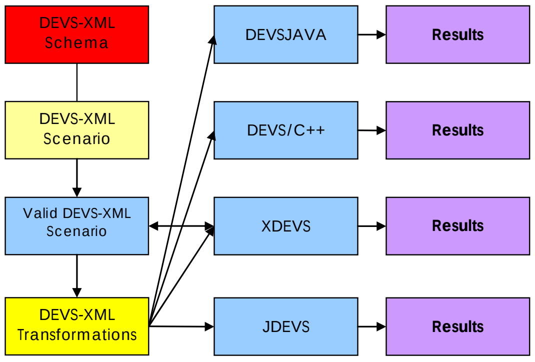

Figure 1: Modeling process of coupled scenarios

Earlier work by Vladimir [9] has also proposed XML representations for DEVS models, but they used JavaML [10] for the behavior specification, so the user cannot write a DEVS-XML scenario without the help of a graphical tool. In contrast to this, our proposal provides a way to:

1. Write DEVS-XML scenarios without any support from a programming language
2. Validate the DEVS-XML scenarios created
3. Transform a XDEVS-Java scenario into a DEVS-XML scenario.

3 The DEVS representation

The Discrete EVent System specification formalism (DEVS) was pioneered by Bernard Zeigler in the mid–seventies [1]. DEVS allows representation of all systems whose input/output behavior can be described by a sequence of events with the caveat that the state has a finite number of changes in any finite interval of time.

A DEVS model processes an input event trajectory along with its own initial conditions, provokes an output event trajectory.

Formally, a DEVS atomic model is defined by the following structure:

$$M = \left(X, S, Y, \delta_{\text{int}}, \delta_{ext}, \lambda, ta\right)$$

Where:

- X is the set of input event values, i.e., the set of all the values that an input event can take.
- S is the set of state values.
- Y is the set of output event values.
- δ_{int}, δ_{ext}, λ and ta are functions which define the system dynamics.

Atomic models may be coupled in the DEVS formalism to form a coupled model. A coupled model provides a way to couple (connect) several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model, thereby giving rise to hierarchical construction (see Figure 2).

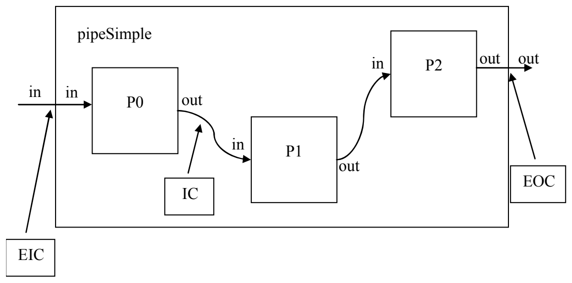

Figure 2: A Coupled model

The mathematical formulation that describes this process is as follows:

$$N = \left\{X, Y, D, \{M_d / d \in D\}, EIC, EOC, IC\ \right\}$$

where:

- X is the set of input event values.
- Y is the set of output event values.
- D is the set of the components.
- M_d is the model of the d-component.
- EIC is the set of the external input connections.
- EOC is the set of the external output connections.
- IC is the set of the internal connections.

A DEVS scenario can be defined using mathematical formulation. Consider the simple processor example showed in Figure 3 [9]:

$$DEVS = \left(X_M, Y_M, S, \delta_{ext}, \delta_{\text{int}}, \delta_{con}, \lambda, ta\right), \text{ where}$$

$$IPorts = \{"in"\}, \text{ where } Xin = J \text{ (a set of job identifies),}$$

$$X_M = \{(p,v) \mid p \in IPorts, v \in Xp\}$$

$$OPorts = \{"out"\}, \text{ where } Yout = J.$$

$$Y_M = \{(p,v) \mid p \in OPorts, v \in Yp\}$$

$$\text{is the set of output ports and values;}$$

$$S = \{"passive","busy"\} \times \Re_0^+ \times J$$

$$\delta_{ext}\left(phase, \sigma, j, e, \left(("in", j1), ("in", j2), \ldots, ("in", jn)\right)\right) =$$

$$= \begin{cases} ("busy", processing\_time, jn) & \text{if } phase = "active" \\ (phase, \sigma - e, j) & i.o.c. \end{cases}$$

$$\delta_{\text{int}}(phase, \sigma, j.q) = \begin{cases} ("busy", \infty, q) & \text{if } q = \Lambda \\ ("passive", processing\_time, j.q) & i.o.c. \end{cases}$$

$$\delta_{con}(S, ta(s), x) = \delta_{ext}(\delta_{\text{int}}(s), O, x)$$

$$\lambda("bussy", \sigma, j.q) = j$$

$$ta(phase, \sigma, j) = \sigma$$

Figure 3: Simple processor expressed in DEVS

Figure 4 shows the deltext behavior expressed in Java within the DEVSJAVA simulator context.

```
public void deltext(
      double e,message x) {
 Continue(e);
 if (phaseIs("passive"))
  for (int i=0; i< x.getLength();i++)
   if (messageOnPort(x,"in",i)) {
    job = x.getValOnPort("in",i);
    holdIn("busy",processing_time);
   }
}
```

Figure 4: deltext() model (behaviour) in DEVSJAVA

An XML modeling vocabulary could take its place alongside the other modeling languages, but this is not the goal of the research we report here. Creating a standard for scenarios is fundamentally different from creating a standard for atomic models. The mathematical components are different, the efficiency and representational considerations are different, and the contexts in which the designs are to be applied are different. Modeling languages and atomic systems are intended to be used by people, and so have a variety of designs reflecting personal preferences and usages. for the communication of scenarios between modeling systems, which are largely hidden from people, are much more readily standardized. Thus a standard for scenarios has much better prospects for being widely adopted and reused.

4 The case for XML

There are other approaches to represent DEVS models, such as SESM/CM [17]. SESM/CM is suitable for developing component-based hierarchical models. It offers a basis for modeling behavioral aspect of atomic models by providing the structural specification and storage of the model. However, a XML representation offers several advantages and is increasingly being adopted as a standard for the interchange of information in diverse fields of science [15] and operations research [16].

An XML vocabulary is formally defined by an XML Schema [12, 13, 14] against which every file written in the vocabulary can be automatically validated. This arrangement gives an XML vocabulary several important advantages over PSMs (Platform Specific Models):

- Validation against a schema promotes stability of the standard.
- The schema can restrict data types.
- The schema can define key data to insure, for example, that one atomic name is not used more than once.
- The schema can be extended to include constraint types or simulator directives. Files that validated under the original schema continue to validate under the extended one (though of course, the reverse is not guaranteed).

This broader relevance has benefits for DEVS systems:

- When scenarios are stored in XML format, DEVS technology results are more readily integrated into broader information technology infrastructures.
- XML is the data interchange language of Web services.
- XML lends itself very well to compression.
- XML-based Extensible Stylesheet Language Transformations (XSLT) offers a convenient way to specify translations of XML documents. If a DEVS scenario (with perhaps corresponding results) is stored in XML, then XSLT is easily applied to the scenario to produce a Web browser document that displays the results data in reports that are suitable for people to read.
- Encryption standards such as XML Encryption are emerging for XML data. This encryption is important to commercial DEVS applications where the scenarios contain confidential data.

Libraries for these purposes - validating files, defining keys, compressing files, and the like, are provided by numerous XML tools designed for manipulating and parsing XML data. It suffices to define our XML vocabulary in the form of a schema that these tools can work with. This contrasts to ad hoc formats that require writing, debugging, and maintaining routines equivalent to these tools.

5 DEVS-XML Schema definitions

5.1 Atomic DEVS Meta-Language

The XML-Schema which defines atomic DEVS models is called *Atomic DEVS-XML Schema*. It is organized in a structure based on the same elements as DEVS formalism. These are the inputs, outputs, states set, transition functions, output function and time-advance function. Figure 5 shows the general structure of the *Atomic DEVS-XML Schema*.

The input and output specification of DEVS-XML is made by means of the ports definition (see Figure 6). These ports have an internal structure formed by a group of signals: the ports may be simple (only one signal or message) or composed (a set of signals).

The DEVS-XML Schema incorporates the definition of signal types. These types are divided in two categories: elementary (integer, real, positive integer, etc.) and enumerated. This set of types can be extended within the DEVS-XML Schema to cover specific types needed.

The states specification is made by means of two structures: one for the state of the system and other for the set of state variables (see Figure 7). The state of the system is defined as a set of possible states, whereas the state variables are defined one by one specifying their type. These data types are the same explained for the case of signals.

The transition functions are similar. In the case of the external transition function it is necessary to specify the port by which the events arrive. The specification of the transition functions consists of the description of the new state where the system will be as well as the new values for state variables. This work contributes some behavior structures, such as expressions, that allow us to define different types of transitions based on logic expressions. Figure 8 shows the schema of the transition functions.

We have defined a grammar based on regular expressions in order to specify the logic of the state transitions. This grammar facilitates a homogeneous definition of the model behavior. In addition, it facilitates automatic processing and conversion from DEVS-XML models to other PSMs and vice versa. Actually, the transformation from PSMs to DEVS-XML is developed only in the case of XDEVS scenarios.

The output function is specified by means of the description of "SEND" clauses (see Figure 9). These clauses provide the mechanism to define the path that the output signals must follow. As in the case of the transition functions, DEVS-XML Schema incorporates the definition of conditional clauses in order to define the logic of the output decisions. It offers the possibility of incorporating multiple signals by multiple ports.

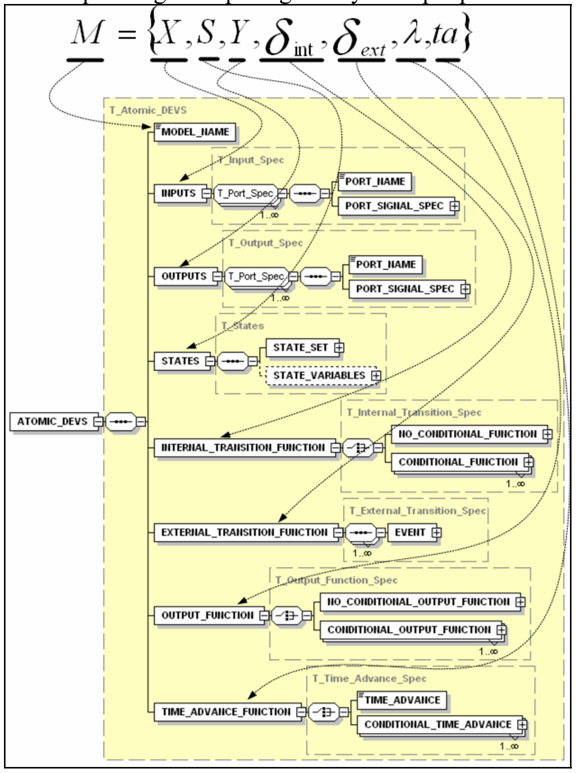

Figure 5. DEVS-XML Schema: Atomic model structure

DEVS-XML Schema also provides a structure to define the time advance function (Figure 10). As in previous cases the XML-Schema permits the possibility of including conditional logic.

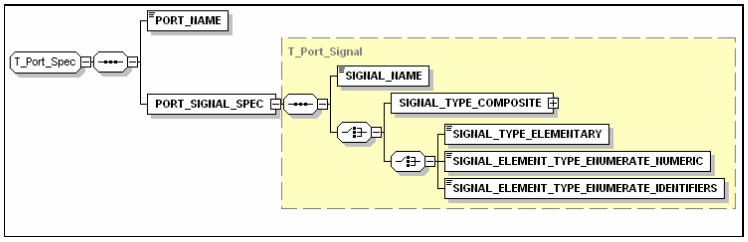

Figure 6: DEVS-XML Schema: Port specification

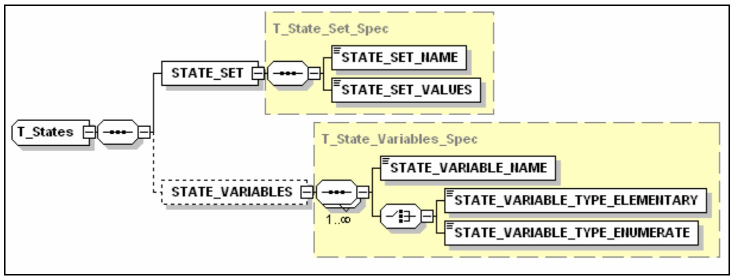

Figure 7: DEVS-XML Schema: State specification

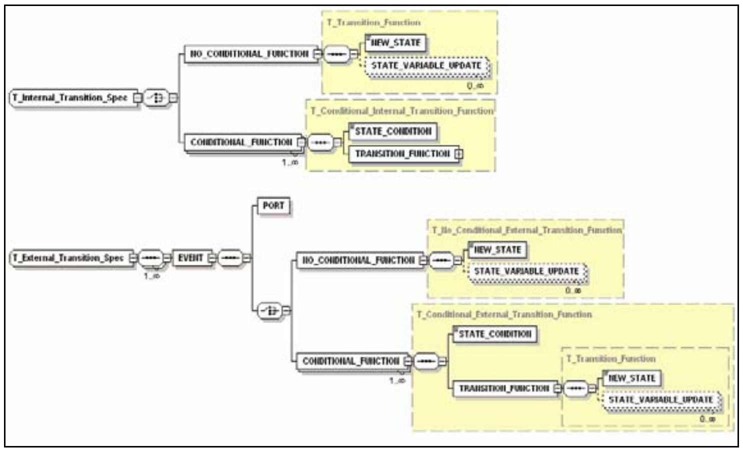

Figure 8: DEVS-XML Schema: Transition functions specification

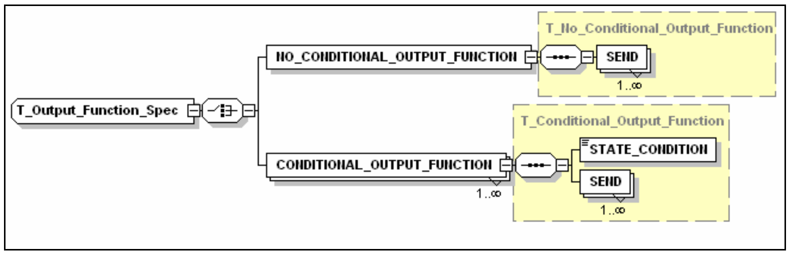
T_Output_Function_Spec
NO_CONDITIONAL_OUTPUT_FUNCTION
T_No_Conditional_Output_Function
SEND
1..∞
CONDITIONAL_OUTPUT_FUNCTION
1..∞
T_Conditional_Output_Function
STATE_CONDITION
SEND
1..∞

Figure 9: DEVS-XML Schema: Output function specification

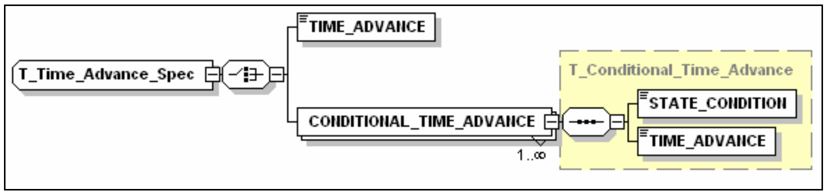

Figure 10: DEVS-XML Schema: Time advance function specification

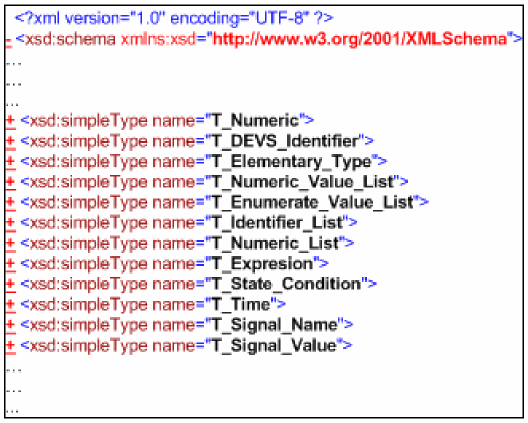
```
  <?xml version="1.0" encoding="UTF-8" ?>
- <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
…
…
…
+ <xsd:simpleType name="T_Numeric">
+ <xsd:simpleType name="T_DEVS_Identifier">
+ <xsd:simpleType name="T_Elementary_Type">
+ <xsd:simpleType name="T_Numeric_Value_List">
+ <xsd:simpleType name="T_Enumerate_Value_List">
+ <xsd:simpleType name="T_Identifier_List">
+ <xsd:simpleType name="T_Numeric_List">
+ <xsd:simpleType name="T_Expresion">
+ <xsd:simpleType name="T_State_Condition">
+ <xsd:simpleType name="T_Time">
+ <xsd:simpleType name="T_Signal_Name">
+ <xsd:simpleType name="T_Signal_Value">
…
…
…
```

Figure 11: DEVS-XML Schema: Type specification

DEVS-XML Schema incorporates the definition of a set of types giving support to main and internal structure of elements. The main types defined in DEVS-XML Schema are listed in Figure 11. The most important characteristics for such XML type specification are the following ones:

- Definition of simple ports with a single signal or compound ports (formed by a set of different signals).
- Definition of different type of signals: integer, real, enumerated type, natural, etc.
- Definition of state variables.
- Definition of model behavior by means of expressions related to the state of the system and the set of state variables.
- Multiple output, in the sense of that an atomic model can emit a signal or a set of signals through several ports at once.

Finally, an important feature of XML is that it supports encryption standards such as XML Encryption. This standard has the flexibility to specify encryption of specific elements, for example, a user could encrypt the data in an atomic component by adding child elements to the sate variable initialization.

5.2 Coupled DEVS Meta-Language

The definition of coupled models is simple once the DEVS-XML Schema for atomic models has been defined. This schema structure, as is defined in DEVS formalism, is formed by the following elements:

- A set of inputs and outputs of the coupled system -with the same structure of signals and ports defined in the schema of the atomic model.
- A list of atomic models that forms the coupled model -a list of identifiers corresponding to the names of the internal atomic models.
- The atomic models participating into the coupled model.
- The set of external input connections: like a set of pairs (port/signal, atomic/port/signal). This set of pairs defines the connections between the input to the coupled model and the input of the elements within the coupled model.
- The set of external output connections, like a set of pairs (atomic/port/signal, port/signal). This set defines the connections between the output of the atomic models inside of the coupled model and the output of the coupled model.
- The set of internal connections between atomic models, like a set of pairs (atomic/port/signal, atomic/port/signal). This set defines the connections among elements inside of the coupled model.

Figure 12 shows the XML Schema for coupled DEVS models with the elements described above.

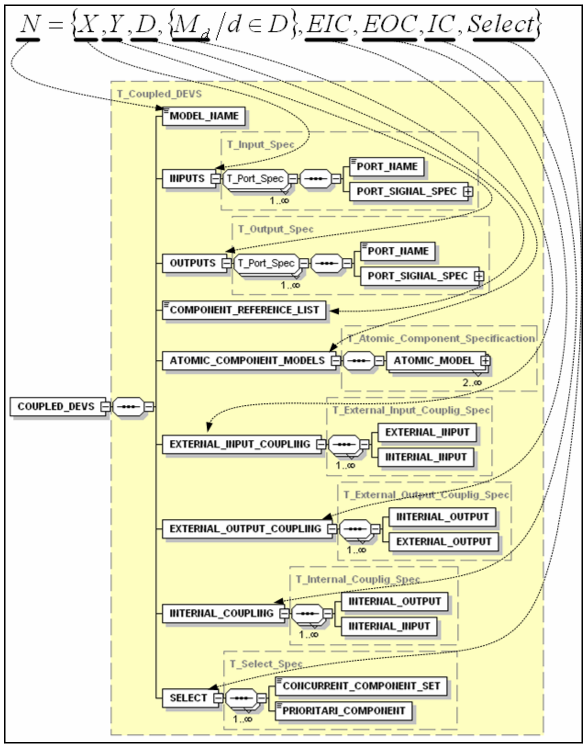

Figure 12: DEVS-XML Schema for coupled models

5.3 Transformations

DEVS-XML scenarios can be considered as PIMs (Platform Independent Models). Likewise, DEVS simulator implementations like DEVSJAVA, DEVS/C++, etc, can be considered PSMs (Platform Specific Models). The development of DEVS-XML scenarios provides of abstraction to the implementation details, allowing the user to be centered only on the problem solving.

DEVS-XML Schema facilitates the development of DEVS-XML scenarios because there are several tools that, in addition to XML editing, allow the user to validate and verify the XML scenario respect to the Schema. These kinds of tools provide a good support to the development of PIM models.

The next problem is the automatic conversion between DEVS-XML scenarios and the Simulator implementations on different platforms i.e. transformations from PIM scenarios to PSM scenarios and vice versa.

One proposed methodology for automated conversion of DEVS-XML scenarios to Simulator implementation is to use JAXP (Java API for XML Processing) and XSL (eXtensible Stylesheet Language). JAXP is an abstraction layer to work with SAX interface (Simple API for XML), trees and DOM classes (Document Object Model). XSL is a family of languages which allows one to describe how files encoded in the XML standard are to be formatted or transformed. With these tools the conversion becomes the two part process. First, we have to analyze the DEVS-XML structure in order to verify that it is well-formed and is valid. The second stage is the transformation.

DEVS-XML Schema includes a well defined structure for mathematical expressions in the form of regular expressions. This definition provides additional support that facilitates the analysis of these expressions (used in the transition functions, output function and time-advance function) and the code generation into any simulation platform. The conversion from a language specific model implementation to DEVS-XML Scenarios is similar to the previous one. In this case, the language dependency of the original models (Java, C++, etc) requires starting with a good specification of the grammar of the underlying language, i.e., a direct relation between the programming structures and XML. There are several tools for analysis (for example Flex, Bison, etc.). In our case, we use the XML representation of JAVA, know as JAVAML[7].

We developed a translator, which gets Java XML from XDEVS-Java and transforms it into DEVS-XML Scenarios. The simulator also implements the DEVS simulation protocol (in Java) so that the generated files can be readily tested for their verification.

6 DEVS-XML Generation Example

In this section, we present an example of an atomic model developed in Java to a DEVS-XML model. The atomic model shown here is a simple processor proposed by Zeigler [1]. Figure 13 shows the DEVSML description of this atomic model. The generation of a coupled model is easier since it only contains components and coupling information, not behavior.

```
<?xml version="1.0" encoding="UTF-8"?>
<ATOMIC_DEVS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="C:\MLP\Doctorado\Linea
de Tiempo Real\Especificacion_DEVSML\Atomic_DEVSML.xsd">
    <ATOMIC_MODEL_NAME>processor</ATOMIC_MODEL_NAME>
    <INPUTS>
        <PORT_NAME>in</PORT_NAME>
        <PORT_SIGNAL_SPEC>
            <SIGNAL_NAME>job_id</SIGNAL_NAME>
            <SIGNAL_TYPE_ELEMENTARY>Integer</SIGNAL_TYPE_ELEMENTARY>
        </PORT_SIGNAL_SPEC>
    </INPUTS>
    <OUTPUTS>
        <PORT_NAME>out</PORT_NAME>
        <PORT_SIGNAL_SPEC>
            <SIGNAL_NAME>job_id</SIGNAL_NAME>
            <SIGNAL_TYPE_ELEMENTARY>Integer</SIGNAL_TYPE_ELEMENTARY>
        </PORT_SIGNAL_SPEC>
    </OUTPUTS>
    <STATES>
        <STATE_SET>
            <STATE_SET_NAME>phase</STATE_SET_NAME>
            <STATE_SET_VALUES>busy passive</STATE_SET_VALUES>
        </STATE_SET>
        <STATE_VARIABLES>
            <STATE_VARIABLE_NAME>processing_time</STATE_VARIABLE_NAME>
            <STATE_VARIABLE_TYPE_ELEMENTARY>Real</STATE_VARIABLE_TYPE_ELEMENTARY>
            <STATE_VARIABLE_NAME>job</STATE_VARIABLE_NAME>
            <STATE_VARIABLE_TYPE_ELEMENTARY>Integer</STATE_VARIABLE_TYPE_ELEMENTARY>
        </STATE_VARIABLES>
    </STATES>
    <INTERNAL_TRANSITION_FUNCTION>
        <NO_CONDITIONAL_FUNCTION>
            <NEW_STATE>passive</NEW_STATE>
            <STATE_VARIABLE_UPDATE State_Variable_Name="processing_time"
       State_Variable_Value="5"/>
        </NO_CONDITIONAL_FUNCTION>
    </INTERNAL_TRANSITION_FUNCTION>
    <EXTERNAL_TRANSITION_FUNCTION>
        <EVENT>
            <PORT Port_Name="in"/>
            <CONDITIONAL_FUNCTION>
                <STATE_CONDITION>phase==passive</STATE_CONDITION>
                <TRANSITION_FUNCTION>
                    <NEW_STATE>busy</NEW_STATE>
                    <STATE_VARIABLE_UPDATE State_Variable_Name="processing_time"
        State_Variable_Value="5"/>
                </TRANSITION_FUNCTION>
            </CONDITIONAL_FUNCTION>
            <CONDITIONAL_FUNCTION>
                <STATE_CONDITION>phase==busy</STATE_CONDITION>
                <TRANSITION_FUNCTION>
```

```
                                        <NEW_STATE>busy</NEW_STATE>
                                        <STATE_VARIABLE_UPDATE State_Variable_Name="processing_time"
                State_Variable_Value="sigma-_e"/>
                                        <STATE_VARIABLE_UPDATE State_Variable_Name="job"
                State_Variable_Value="_xSignalValue("in","job_id"/>
                                </TRANSITION_FUNCTION>
                        </CONDITIONAL_FUNCTION>
                </EVENT>
        </EXTERNAL_TRANSITION_FUNCTION>
        <OUTPUT_FUNCTION>
                <NO_CONDITIONAL_OUTPUT_FUNCTION>
                        <SEND Port_Name="out" Signal_Name="job_id" Signal_Value="job"/>
                </NO_CONDITIONAL_OUTPUT_FUNCTION>
        </OUTPUT_FUNCTION>
        <TIME_ADVANCE_FUNCTION>
                <TIME_ADVANCE>Sigma</TIME_ADVANCE>
        </TIME_ADVANCE_FUNCTION>
</ATOMIC_DEVS>
```

Figure 13: DEVS-XML description of the Atomic model

7 Conclusions and Future work

The DEVS-XML Schema development can be considered a suitable base for the generation of platform-independent DEVS models. A W3C XML Schema has been introduced, first for DEVS atomic models and then for DEVS coupled scenarios. The use of XML with an XML Schema has been the preferred mechanism in order to generate transformations from DEVS-XML scenarios to PSM scenarios and vice versa. Henceforth, a transformer from platform-specific model to DEVS-XML has been developed demonstrating the proof of concept. We are in the process of developing transformations from DEVS-XML scenarios to platform-specific scenarios for ex, in Java and C++.

Presently, DEVS-XML Schema is applicable to a very small subset of DEVS models as the behavior grammar is not complete and suffers from absence of programming logic into XML for example, loops and if-else constructs. Representing scenarios of such kind is very important and is required to exploit the true power of DEVS modeling. An existing XML vocabulary, JavaML [7], can be used to represent Java code, thereby, building on the power of Java. One direction of research we are now following is to develop a schema that uses the JavaML namespace in order to describe the behavior along with the rules and XML-Schema developed herein to describe the structure.

References

[1] Zeigler, B. P.; Praehofer, H. and Kim, T. 2000. "Theory of modeling and simulation", 2nd Edition, Academic press 2000, ISBN 0-12-778455-1.
[2] http://www.acims.arizona.edu/SOFTWARE/software.shtml, last accessed Jan 12, 2005
[3] H. Sarjoughian, B. Zeigler, and S. Hall, "A Layered Modeling and Simulation Architecture for Agent-Based System Development", Proceedings of the IEEE 89 (2); 201-213, 2001
[4] Zeigler, B.P. 1997. "DEVS-JAVA User's Guide." TechnicalReport. AI & Simulation Lab., Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ.
[5] Zeigler, B. P.; Moon, Y.; Kim, D. and Kim, J.G. 1996. "DEVS/C++ A High Performance Modeling and simulation environment", Hawaii international conference on system sciences - HICSS, pages 350 – 359.
[6] Mittal, S., Mak E., Nutaro, J.J., "DEVS-Based Dynamic Model Reconfiguration and Simulation Control in the Enhanced DoDAF Design Process", submitted to special issue on DoDAF, Journal of Defense Modeling and Simulation (JDMS).
[7] XDEVS web page:
http://itis.cesfelipesegundo.com/~jlrisco/xdevs.html
[8] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau. 2004. "Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation". World Wide Web Consortium (W3C).
[9] Vladimír Janousek, Petr Polásek and Pavel Slavícek. "Towards DEVS Meta Language". In: ISC 2006 Proceedings, Zwijnaarde, BE, 2006, p. 69-73, ISBN 90-77381-26-0
[10] Badros, G., "JavaML: a markup language for Java source code". In *Proceedings of the 9th International World Wide Web conference on Computer networks: the international journal of computer and telecommunications networking*, pages 159-177.
[11] Zeigler B, Sarjoughian H., (2005) "Introduction to DEVS Modeling and Simulation with Java: Developing Component-Based Simulation Models". Arizona Center for Integrative Modeling and Simulation.
[12] David C. Fallside, Priscilla Walmsley .2004. "XML Schema Part [0]: Primer Second Edition". World Wide Web Consortium (W3C).
[13] Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn. 2004. "XML Schema Part [1]: Structures Second Edition". World Wide Web Consortium (W3C).
[14] Paul V. Biron, Kaiser Permanente, Ashok Malhotra. 2004. "XML Schema Part [2]: Datatypes Second Edition". World Wide Web Consortium (W3C).
[15] O'Reilly, Science XML vocabularies.
http://www.xml.com/pub/rg/Science, 1999
[16] Bradley, G., "Introduction to extensible markup language (XML) with operations research examples". Newsletter of the INFORMS Computing Society, 24:1-20, 2003.
[17] Bendre, S., Sarjoughian, H.S., "Discrete-Event Behavioral Modeling in SESM: Software Design and Implementation", Advanced Simulation Technology Conf., pp. 23-28, Apr., San Diego, CA., 2005.