An XML-based DEVS Modeling Tool to Enhance Simulation Interoperability

Yung-Hsin Wang
Department of Information Management
Tatung University
40 Chungshan N. Rd., 3rd Sec, Taipei 104, Taiwan
E-mail: ywang@mis.ttu.edu.tw

Yao-Chung Lu
Department of Computer Science and Engineering
Tatung University
40 Chungshan N. Rd., 3rd Sec, Taipei 104, Taiwan
E-mail: g8906023@mail.ttu.edu.tw

KEYWORDS
Modeling and Simulation, DEVS, XML, Interoperability.

ABSTRACT

There is a need to develop standards for a computer processable representation of DEVS (Discrete Event System Specification) models that supports common understanding, sharing and interoperability of distinct DEVS implementations. To achieve the goal, this paper focuses on the adoption of XML in model description and development to generate a standard intermediate format to entitle sharing between different DEVS simulation environments that will facilitate simulation interoperability standards and reuse.

INTRODUCTION

With advances of computer technology, simulation has been used extensively in all aspects of industry, government and academia as powerful methodology for analyzing and designing the performance and characteristics of complex systems. In general project of simulation, model builder creates an all-new simulation model from basic constructs, and quite often the simulation model had been built cannot share to others in different simulation environments. This is a kind of waste of model designer's efforts and sometimes causes inconvenient. Therefore, it is important to integrate different simulation tools to enhance the model reusability and exchangeability between simulation practitioners.

In this paper we focus on DEVS (Discrete Event System Specification) approach (Zeigler 1984; Zeigler 1990), which provides a formal basis for specifying discrete event models in a hierarchical, modular manner. Because of its theoretical foundation, the DEVS formalism and its associated techniques have proven to be a powerful means for model management and simulation of general systems. Currently, there are many different realizations of DEVS-based simulation environments in the world (Kim and Park 1992; Tan 1996; Zeigler 1986; Zeigler et al. 1996; Zeigler 1997). A common problem would be the model reusability and code sharing. For example, a model built in the DEVS-Scheme under DOS or Windows cannot run on the C++ or Java versions of DEVS under Unix or Windows. Project designers in the heterogeneous platforms may all use the same language to develop their simulation models. However, the problem then is that they need to develop every API of their own platform or possess the same simulation environment as others. In other words, users may have to give up their proficient and original simulation environments if required. This is infeasible due to a lot of expenditure to a project with limited budget.

The goal of this paper is to develop a user-friendly tool that allows users to construct DEVS simulation models via a graphical user interface as well as to facilitate model reuse and code sharing of DEVS simulation models. We use the XML technology to present the DEVS model for solving the interoperability problem. On the other hand, users can generate simulation codes easily using a specific model language translator. Abundant XML developing tools and parsers can help users rapidly build a new model language translator such as that of C++ or Java for various DEVS-based simulation environments.

In the rest of this paper, we will give a brief review of the DEVS modeling and simulation methodology, System Entity Structure concept and XML technologies, which are the foundation of our system. We then present the architecture and implementation of our XML-Based DEVS simulation modeling tool and describe the method that translates XML document to the model code. Finally a conclusion of this study is made.

DEVS SIMULATION AND XML TECHNOLOGIES

Overview of the DEVS formalism and SES

The structure of the model that generates the model behavior can be expressed in a mathematical language or formalism. The discrete-event system specification formalism provides a means of specifying a mathematical object called a system. Basically, systems contain many common attributes including a time base, inputs, states, and outputs, and functions for determining next states and outputs given current states and inputs. When using DEVS approach in simulation, one must specify (1) the atomic models, from which larger coupled models are built, and (2) how these models are connected in hierarchical fashion.

Formally, an atomic model M is defined as

$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$

where

- X : the set of external input event types
- S : the sequential state set
- Y : the set of external event types generated as output
- δ_{int} : $S \rightarrow S$, is the internal transition function
- δ_{ext} : $Q \times X \rightarrow S$, is the external transition function where $Q = \{ (s, e) \mid s \in S, 0 \le e \le ta(s) \}$ is the total state set
- λ : $S \rightarrow Y$, is the output function
- ta : $S \rightarrow R^{+}_{0,\infty}$ (non-negative real), is the time advance function

The second form of DEVS model, the coupled model, specifies how to connect several models to form a new model. The structure of a coupled model DN (stands for DEVS Network) is defined as

$DN = < D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, select >$

where

D is a set of component names,

for each i in D,

M_i is a component basic model

I_i is a set, the influences of i,

and for each j in Ii

$Z_{i,j}$ is a function, the i-to-j output translation, and

select is a function, the tie-breaking selector

Detail descriptions for the definitions of the atomic model, coupled model and DEVS formalism can be found in (Zeigler 1984; Zeigler 1990). The primary advantage of DEVS is that it provides a formal way that it characterized how discrete event languages specify their discrete event system parameters. With this abstraction, it is possible to design new simulation languages with sound semantics that are easier to understand. Indeed, the modeling and simulation environments that implement DEVS formalism inherit the abilities such as support of building models in a hierarchical, modular manner and object-oriented concepts. In addition, DEVS formalism appears to be the more powerful formalism, trading mathematical tractability for expressive power.

As a hierarchical discrete-event model base is developed and expanded for a particular domain or family, the "manual" management of relative models becomes difficult. A structured knowledge representation scheme called System Entity Structure (SES) is applied to direct the synthesis of models from the model base. It incorporates decomposition, taxonomy, and coupling knowledge concerning a domain of real systems. A modeler may prune the SES according to the objectives of his/her study obtaining a reduced structure (i.e., pruned SES, or PES) that specifies a hierarchical discrete event model.

The SES is defined as a labeled tree with various types of attachment that satisfies several well-defined axioms (Zeigler 1984; Zeigler 1990). There are three types of nodes in the SES—*entity*, *aspect*, and *specialization*—which stand for three types of knowledge about the structure of the system. An *entity* represents a real world object, which can either be independently identified or postulated as a component of some decomposition of other real world object. In simulation perspective an entity is a model. An *aspect* represents a decomposition out of many possibilities of an entity. The descending entities of an aspect can be considered as components of the aspect's parent. A *specialization* is a mode of classifying entities and is used to represent the taxonomy of the system being modeled.

Consider the simple server architecture as an example. A simple server consists of a processing element and a queue with two types of queuing discipline, FIFO and LIFO. Figure 1 shows its SES representation. The root entity SERV_ARCH has an *aspect* node (the decomposition relation is represented as a vertical bar '|') that implies the descendants (PROCESSOR and QUEUE) can be coupled to form SERV_ARCH. A *specialization* node (represented as double bars '||') is attached to the entity QUEUE meaning that the two descending nodes, FIFO_Q and LIFO_Q are types of QUEUE. If a SES has no specialization and at most one aspect under each entity, it is said to be pure. Such a SES can be transformed into a hierarchical model and simulated. *Pruning* is required to create a pure SES. Figure 2 shows a Pruned Entity Structure (PES) of SERV_ARCH.

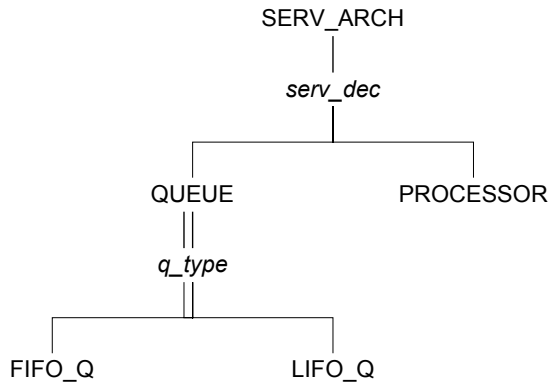

Figure 1: The SES of the Simple Server Architecture

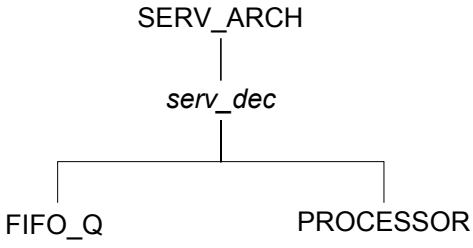

Figure 2: The Pruned Entity Structure of SERV_ARCH

Extensible Markup Language

Extensible Markup Language (XML), defined by the World Wide Web Consortium (W3C), is a subset of SGML specifically designed for the Web (Deitel et al. 2001). It is a universal language and is emerging as a new way to store, describe and exchange data on the web. Even though its primary application is as the future of the World Wide Web, it can be used in a variety of situations to structure digital data (Kim 2001).

XML, like HTML, is based on tags and represents documents as trees of element. It also has two sorts of element: empty and non-empty elements. Moreover, the XML specification defines precise rules that make document parsing simple. XML specification defines two types of XML document: valid documents and well-formed documents. To define and validate an XML document's structure one can use Document Type Definitions (DTDs) or Schemas. Note that we use the Microsoft XML Schemas in this study to describe the structure and element content of our XML documents for the DEVS models.

SYSTEM ARCHITECTURE AND IMPLEMENTATION

Figure 3 presents our system architecture, which contains two main parts, the Model Builder and the Specific DEVS Model Language Translation Module. Different DEVS simulation environments need individual translation module. In order to unify the specific presentations of DEVS simulation environment into one standard, we use the XML format to present the DEVS model. In other words, the XML document is an intermediate meta-form of the DEVS model. It allows DEVS models to be shared by other Model Builders of different DEVS simulation environments.

The system works as follows. Users first construct the SES and input model specifications from the User Interface Module, then the XML Translation Module translates the SES into an intermediate meta-form in XML format. Once an XML formatted model is shared out, other users could get and convert it to specific DEVS simulation code by their Model Language Translation Module and execute under the users' own DEVS simulation environment.

The Model Builder

The Model Builder we designed provides an easy and friendly interface for users to build a platform neutral DEVS model in XML. The Model Builder contains two modules: the User Interface Module and the XML Translation Module, which will be illustrated in what follows. Figure 4 shows a screen capture of the user interface of the Model Builder.

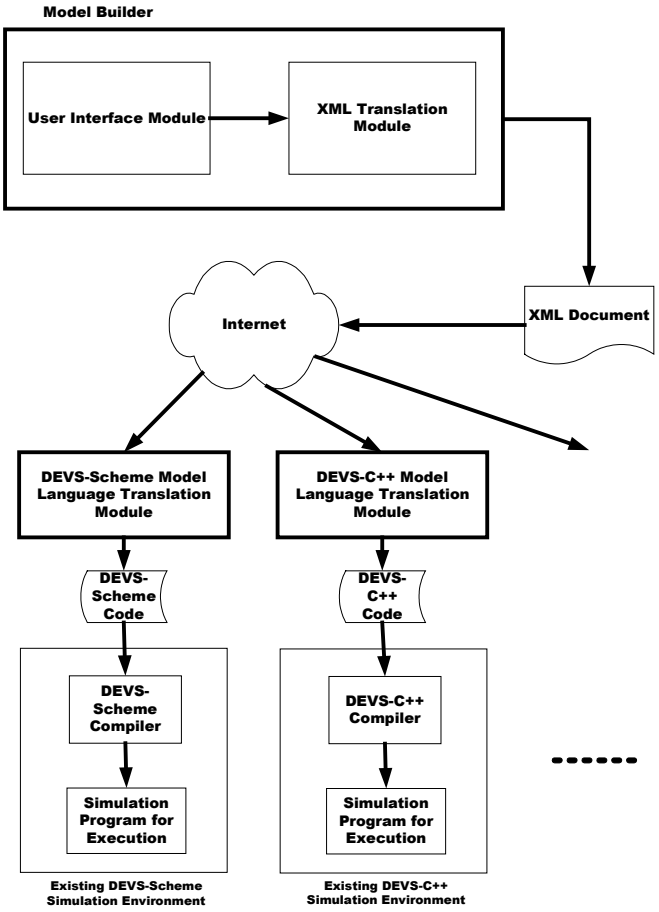

Figure 3: The Software Architecture of Our System

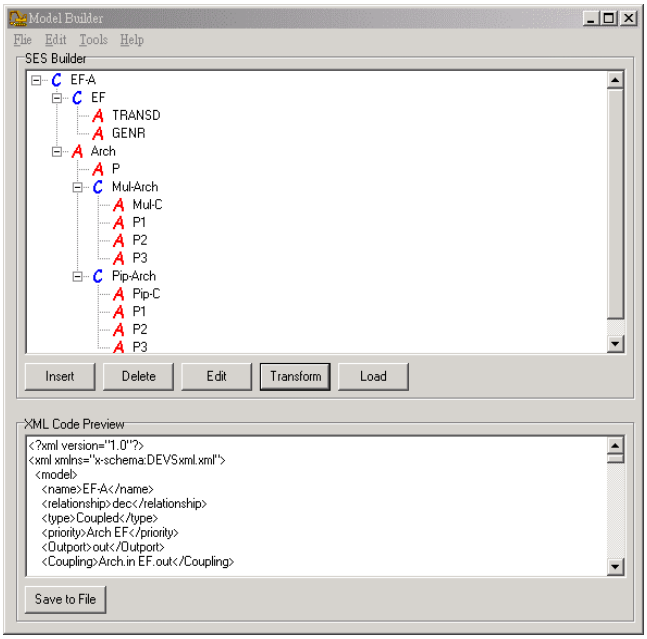

Figure 4: User Interface of the Model Builder

The primary function of the User Interface Module is to offer a GUI for building SES. For the sake of concise appearance, we represent the SES a tree structure consisting of entity nodes without the aspect (i.e., decomposition) and specialization (i.e., taxonomy) nodes. In the SES structure, each entity node is on behalf of an atomic model or a coupled model in DEVS formalism. The attributes of a node contain the information of each model. If a node is expressed as a coupled model, then there is some additional information in this node. The information must include the coupling relationships between parent entity and its child entities, or those among child entities. Those relationships also contain decomposition or specialization information between the parent entity and child entities.

As can be seen in Figure 4, the model builder can insert a node and specify model attributes of a node by clicking the appropriate button from the main form. When clicking on the "insert" button, a dialog form will pop up where users can assign model attributes such as the model name, model type, relationships, input and output ports. After model attributes have been filled, if the appointed model still has further child nodes decomposed itself, users can set priority and build coupling relationship among these child nodes. If the sub-nodes need to set the priority, selecting their parent node and clicking on the "edit" button will pop up a dialog form for this purpose. This form is the same as the form that users used to assign model attributes. To set coupling relationships among models, users simply double-click the input or output port's name, then all connectible ports will show up for users to select and add new connections. In this way, models in the SES can be built easily and quickly.

When the SES has been constructed, it is ready to translate the whole structure into an intermediate meta-form in XML format. The XML Translation Module is responsible for this task. By clicking on the "Transform" button, the SES will be translated into an XML formatted document, which users can preview in the box shown in Figure 4. Users can then just click on the "Save to File" button to save this XML file. The system model represented in XML should be able to be executed among different systems. Users can freely exchange the XML files and the XML files obtained from other model builders can be easily modified to suit users' simulation needs. With the help of this interface, users can also load an XML file, and its corresponding SES tree will be presented. In a nutshell, the function of XML Translation Module is to do the transformation process between the SES tree and its XML equivalent.

The Intermediate Meta-form in XML Format

The Intermediate Meta-form is a well-formed XML document which includes the essential information of the DEVS model from the System Entity Structure. Since every DEVS model is based on the DEVS formalism, all DEVS-based simulation environments can be satisfied with the information that this intermediate meta-form present.

We establish a concise format to form a profile of the whole SES. The XML document is a hierarchical structure as is the SES structure. Therefore, we can directly map the whole SES tree to the XML tree. The attributes of each coupled model and atomic model are written between the "<model>" and "</model>" tags and the value of the attributes are written between the tags of each attribute. An example of a DEVS model intermediate meta-form in XML format is shown in Figure 5.

```
<?xml version="1.0" encoding="UTF-8"?>
<devs xmlns="x-schema:DEVSxml.xml">
  <model>
    <name>SERV_ARCH</name>
    <relationship>dec</relationship>
    <type>Coupled</type>
    <Inport>in</Inport>
    <Outport>out</Outport>
    <Coupling>PROCESSOR.in QUEUE.out</Coupling>
    <Coupling>PROCESSOR.out QUEUE.in</Coupling>
    <model>
      <name>PROCESSOR</name>
      <relationship>ent</relationship>
      <type>Atomic</type>
      <priority/>
      <Inport>in</Inport>
      <Outport>out</Outport>
    </model>
    <model>
      <name>QUEUE</name>
      <relationship>spec</relationship>
      <type>Atomic</type>
      <Inport>in</Inport>
      <Outport>out</Outport>
      <model>
        <name>FIFO_Q</name>
        <relationship>ent</relationship>
        <type>Atomic</type>
        <priority/>
        <Inport>in</Inport>
        <Outport>out</Outport>
      </model>
      <model>
        <name>LIFO_Q</name>
        <relationship>ent</relationship>
        <type>Atomic</type>
        <priority/>
        <Inport>in</Inport>
        <Outport>out</Outport>
      </model>
    </model>
  </model>
</devs>
```

Figure 5: Intermediate Meta-form in XML

Specific DEVS Model Language Translation Module

In order to solve interoperability problems in different simulation environments, we designed an intermediate meta-form in XML format for system modeling as has been presented. Any model builder can easily get the modeling

information from the XML document, and create a specific DEVS model language for its corresponding simulation environment. Our first attempt of the implementation of the language translation module is for DEVS-Scheme. Note that each DEVS simulation environment must have its own translation module just like each operation system has its own Java virtual machine. Nevertheless, users can easily build one by using abundant XML developing tools and XML parsers available. Afterward, all modelers can use their own simulation environment to execute other DEVS models or make models open and share to other model builders without any modification. A form of such language translation module is shown in Figure 6.

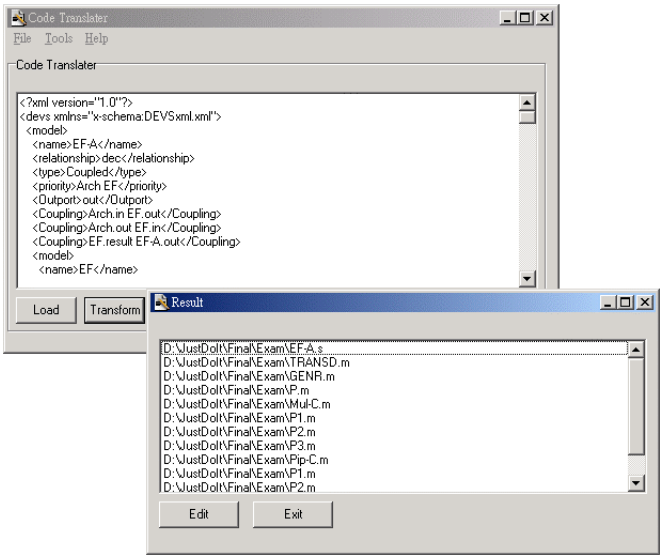

Figure 6: The Model Language Translation Module

The Implementation

In this study, we use the Borland Delphi 6 to implement the system. First, the User Interface Module of the Model Builder provides users to create a SES. In this module, we use the "TTreeView" component to achieve our purpose. We can create and view the SES by using the "TTreeView" component in a hierarchical structure. Object "TTreeView" displays a tree diagram of the visual components and their logical relationships, such as sibling, parent-child and property relationships, in a hierarchical fashion. Since a SES has tree-like structure, store it into a "TreeView" object is quite a natural fit. The attribute of each model is stored in the "TTreeView" component's "pointer" attribute. And then we recursively trace the whole "TTreeView" and sequentially write down the XML document.

Second, the XML Translation Module and DEVS-Scheme Language Translation Module use the DOM (Document Object Model) technologies to maintain the XML document. Borland Delphi 6 embeds the DOM implementations into a wrapper component, called "XMLDocument," that makes the XML document become an object with methods and properties. We can easily access the information in the XML object in an object-oriented way. All the key information is in the DOM of the XML document. So we recursively trace the whole XML object and sequentially write down the source code in specific modeling language. In this way, we translate the DEVS models written in XML to any simulation code easily.

CONCLUSION

Simulation technology has been widely used in various areas for saving the time and money of practically establishing the real system. However, system models being built often cannot be executed on other simulation environments. Modelers cannot exchange their simulation models when they do not use the same simulation tool. Most times model designers have to build a new model from the beginning. In this paper, we presented a method to portray DEVS models using the XML technology. We also provide a GUI tool to help users build the System Entity Structure of the model being simulated and save it into an XML document, which can run on any DEVS-based simulation environment after a simple translation. We hope this new realization will provide a solution to the simulation interoperability issue in DEVS community.

REFERENCES

Deitel, H.M.; P.J. Deitel; T.R. Nieto; T.M. Lin; and P. Sadhu. 2001. *XML How to Program*. Prentice-Hall, Englewood Cliffs, NJ.

Kim, H-D. 2001. "An XML-based modeling language for the open interchange of decision models," *Decision Support Systems* 31, Issue: 4 (Oct), 429-441.

Kim, T.G. and S.B. Park. 1992. "The DEVS Formalism: Hierarchical Modular Systems Specification in C++." In *Proceedings of European Simulation Multiconference* (York, UK, Jun 1-3). SCS Europe, 152-156.

Tan, C-T. 1996. "Design and Implementation of a DEVS Simulation Workbench." Master Thesis. Dept. of Computer Science and Engineering, Tatung University, Taipei, Taiwan.

Zeigler, B.P. 1984. *Multifacetted Modelling and Discrete Event Simulation*. Academic Press, London, UK and Orlando, FL.

Zeigler, B.P. 1986. "DEVS-Scheme: a Lisp-Based Environment for Hierarchical, Modular Discrete Event Models." Technical Report. AIS-2, CERL Lab., Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ.

Zeigler, B.P. 1990. *Object-Oriented Simulation with Hierarchical Modular Models: Intelligent Agents and Endomorphic Systems*. Academic Press, Boston, MA.

Zeigler, B.P.; Y.K. Moon; D.W. Kim; and J.G. Kim. 1996. "DEVS-C++: A High Performance Modelling and Simulation Environment." In *Proceedings of 29th Hawaii International Conference on System Science* (Maui, Hawaii, Jan 3-6). IEEE, Piscataway, NJ, 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, AZ.