웹상에서의 시뮬레이션 모델 공유를 위한 XML 기반 DEVS 마크업 언어

An XML-based DEVS Markup Language for Sharing Simulation Models on the Web

김형도*, 김종우**
Kim, HyoungDo, Kim, JongWoo

Abstract

Driven by the explosive expansion and acceptance of the Internet and its multimedia front-end, the Web, a new generation of the modeling and simulation tools have come up with the name of Web-Based Simulation (WBS). Most of WBS libraries inherit its powerful advantages from Java. However, there are cases where explicit specification of models or interface objects is more desirable than the black-box programs. This paper presents an XML-based DEVS (Discrete Event System Specification) markup language for sharing simulation models on the Web. DEVS provides a system-theoretic formalism for the language while XML supports platform-independent data access. This paper focuses on the design of such a language.

Keyword : XML, Simulation Markup Language, DEVS, Model Sharing

* EC-Internet Development Team, Dacom Corp.
** Department of Statistics, ChungNam National University

1. Introduction

Driven by the explosive expansion and acceptance of the Internet and its multimedia front-end, the Web, a new generation of the modeling and simulation tools have come up with the name of Web-Based Simulation (WBS). Those programs generally fall into two categories [25]. The one comprises simulation programs that can be accessed remotely through the Web browsers and forms-based CGI scripts. Typically, these simulations allow the user to tailor (via the forms interface) model execution parameters such as mean service times and arrival rates, number of model replications, and so forth. A single copy of the simulation runs on a server and passes the results of model execution to the invoking client. The other category of WBS programs represents a variation on the first, but with the added feature of code mobility afforded by such network programming languages as Java. Here, the simulation executes on the client rather than the server. Java-based simulation-support libraries are emerging that permit the creation of simulation programs as Java applications and applets. There are now several Java-based libraries including Simkit [3], JavaSim [19], JSim [22], DEVS-Java [5], SimJava [10], and Silk [8]. Their power, flexibility and scalability derive directly from Java: object-orientation, multi-threading, platform independence, component mobility and component-based programming.

Most commercial simulators have severe limitations in modeling and running complex real world systems. To provide detailed behavior description, the number of commands including different parameter formats is too large and complex for users to easily seek appropriate commands. Therefore, limitation on the practical number of commands and parameter formats of the commercial simulators constrains the expression of complex behaviors. Most commercial simulators are incapable of extending the problem space or describing models deeper than some fixed level.

Using a programming language like Java to build models is one solution that complex regular interconnections can be straightforwardly expressed. Based on the assumption that users are familiar with Java, they are required to model too specific programming details such as object creation and deletion. Furthermore, most of Java-based simulation libraries originate from existing simulation languages, so all implement different flavors of simulation. Integrating components that have different frameworks requires much effort. Another problem is that component models are black boxes. That is, a modeler can not understand what a model represents, except its input and output parameters. This partially explains why it is hard to do modeling cooperatively.

Another solution is the formal specification that is based on a system-theoretic formalism. DEVS [31, 32] proposes a system-theory based simulation methodology that provides expandability with modular and hierarchical features. Due to the hierarchical property, users can quickly expand their models. It offers flexibility with object-oriented messages as user-defined data structures. Since a message carries an object, information as a message type can be delivered from one model to another. In other words, the size or scope of the object can be determined by the user, and can be different from one application to another, while the message containing the object can be delivered from a source model to a destination. A DEVS model works as a timed state machine so that the state of the system is changed by external or internal events with elapsed time. However, DEVS environments, including DEVSim++ [16] and DEVS Java [5] are based on object-oriented programming languages. So they have the same limitation.

This paper presents a DEVS-based simulation modeling language that is implemented using XML on the Web. Extensible Markup Language (XML) [2] is a simplified subset of SGML [14] that maintains the SGML features of validation, structure, and extensibility. XML is making rapid progress through standardization process. It has many benefits for folks who want to improve structure, maintainability, searchability, presentation, and other aspects of their document management. In addition to modifying the syntax and semantics of document tag annotations, XML also changes our linking model by allowing authors to specify different types of document relationships. Many communities have struggled to codify the tacit knowledge of their data. First of all, the Web community members are eagerly proposing new XML-based standards for rebuilding or reengineering the Web. CDF (Channel Definition Format) [6] specification, for example, allows publishers to specify their channels, their contents available, their update schedule, and other information. Scientific applications of XML are also active. CML (Chemical Markup Language) [21] uses XML to manage complex molecular information. MathML (Mathematical Markup Language) [13] is an XML application for describing the structure and content of mathematical expressions, allowing the markup of complex formulas, which mathematicians and computer scientists have been clamoring for since the earliest days of HTML. EC(Electronic Commerce) / EDI(Electronic Data Interchange) community is also eagerly seeking for standard representation of product/catalog information and trading protocols. Note that there is ongoing effort to integrate competing standards and/or proposals based on XML for improving its effectiveness and applicability.

The remainder of this paper is structured as follows. Section 2 summarizes the DEVS formalism. Section 3 demonstrates the simulation modeling language, SimX, with an illustrative example. Section 4 discusses comparative meanings of the language. Finally, section 5 summarizes and concludes the paper with further research directions.

2. DEVS formalism

The DEVS formalism proposed by Ziegler is a set theoretical system modeling formalism which provides generality, flexibility, and expandability with modular and hierarchical features [31, 32]. To support modeling and simulation based on DEVS, environments and libraries have been developed such as DEVS-Scheme [15], DEVSim++ [16], and DEVS-Java [5]. To exchange DEVS models among DEVS environments and tools, it is required a standard model exchange format or a tool independent model description language.

OpenDEVS [26] is the first trial to propose an open and extensible model exchange format for DEVS-based modeling, analysis, and simulation tools. Compared with the proposal, we focus on the representation and analysis of simulation models themselves on the Web. Note that SimX can also be used for the interchange of simulation models among DEVS tools.

To support hierarchical modeling, DEVS formalism supports two kinds of models; atomic model and coupled model. An atomic model specifies the dynamics of a model and is defined as:

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

where

X : a set of external input event types

Y : an output set

S : a sequential state set

$\delta_{ext} : Q \times X \rightarrow S$,

an external transition function

where Q is the total state set of

$M = \{(s,e) | s \in S \text{ and } 0 \le e \le ta(s)\}$
$\delta_{int} : S \rightarrow S$, an internal transition function
$\lambda : S \rightarrow Y$, an output function
$ta : S \rightarrow R^{+}_{0,\infty}$, a time advance function
where the $R^{+}_{0,\infty}$ is the non-negative real numbers with ∞ adjoined.

A coupled model connects the basic models together in order to form a new model. This model can itself be employed as a component in a larger coupled model, thereby allowing the hierarchical construction of complex models. The coupled model is defined as:

$CM = <D, \{M_i\}, \{I_i\}, \{Z_{I,j}\}, SELECT>$
where
D : a set of component names
For each i in D
M_i : a component basic model
I_i : a set of influence of I
And for each j in I_i
$Z_{I,j} : Y_i \rightarrow X_j$, an i-to-j output translation
$SELECT : 2^M - \varnothing \rightarrow M$,
a tie-breaking selector

3. XML

Extensible Markup Language (XML) is a simplified subset of SGML that maintains the SGML features of validation, structure, and extensibility. SGML allows documents to be self-describing, through the specification of tag sets and the structural relationships between the tags. This specification is referred to as the Document Type Definition (DTD). HTML is a small hard-wired set of about 70 tags and 50 attributes, which allow HTML users to skip the self-describing aspect from a document. XML, on the other hand, retains the key SGML advantage of self-description through DTDs, while avoiding the complexity of full-blown SGML. XML is making rapid progress through standardization process. It has many benefits for folks who want to improve structure, maintainability, searchability, presentation, and other aspects of their document management. In addition to modifying the syntax and semantics of document tag annotations, XML also changes our linking model by allowing authors to specify different types of document relationships. Furthermore, there is a presentation specification language for XML documents that keep structuring and presentation information separate from actual data. The language XSL (Extensible Style Language) enables developers to more easily format information for the Web browsing.

Figure 1 illustrates XML application process using a sample XML application that can mechanically process simple catalog information. HTML tags such as <table> and <tr> are just interpreted as a presentation language, so machines cannot identify the exact meaning of 'dictionary' and '$100'. Many problems in mechanical sharing of information are derived from the limitation. On the other hand, XML defines content rather than presentation. XML documents refer DTDs which describe the elements and attributes that appear on them. Any applications that refer to the catalog DTD of figure 1 can grasp the meaning of elements and attributes described by tags such as <catalog> and <price>. Any operations such as storage, transformation, and presentation can be easily performed on the parsed information. Furthermore, DOM (Document Object Model) [28] interface enables programs and scripts to dynamically access and manipulate the information.

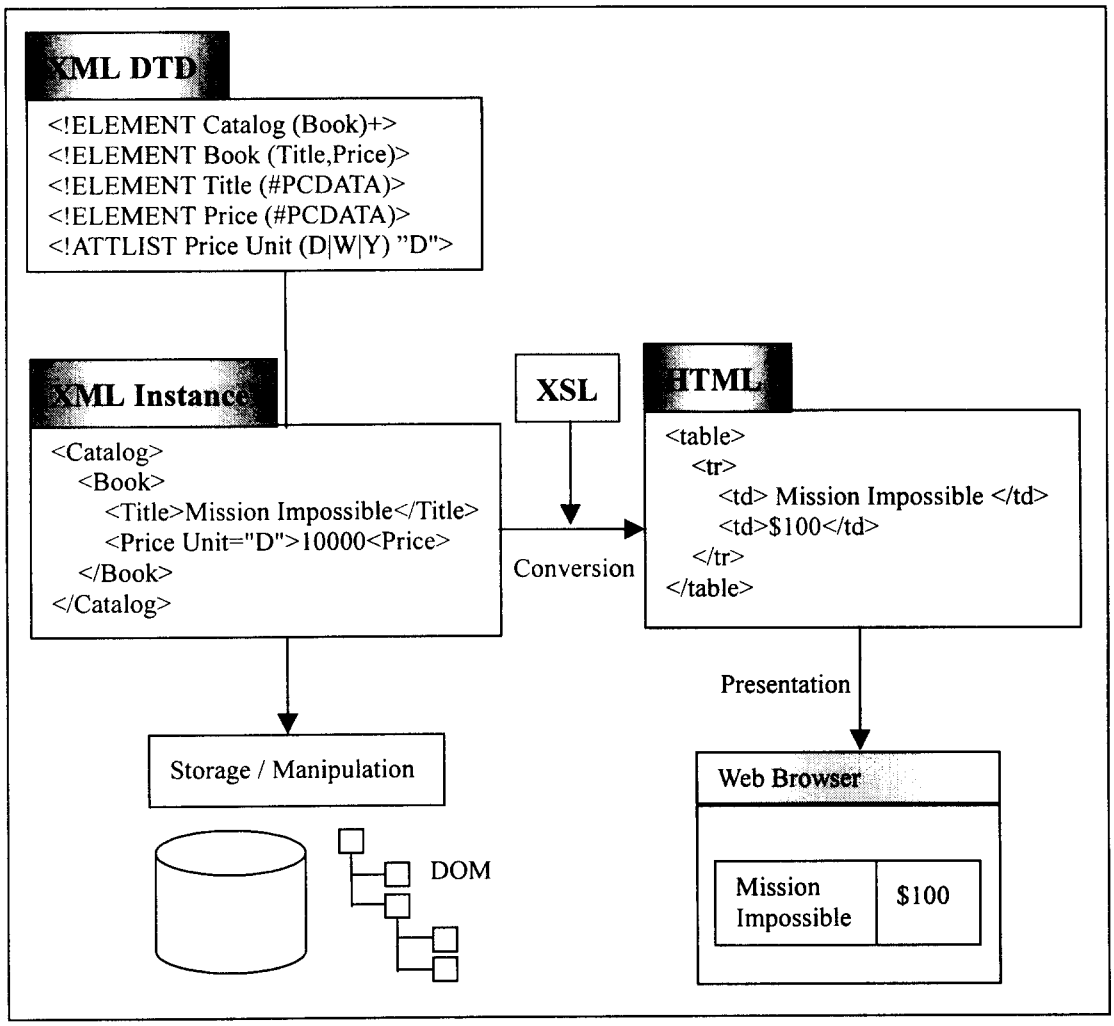

Figure 1. XML Application Process

Many communities have struggled to codify the tacit knowledge of their data. Table 1 summarizes well-known application standards and proposals. First of all, Web extensions themselves are being standardized using XML including CDF [6], OSD [9], RDF [17], WIDL [1] and multimedia applications such as SMIL [29] and VML [20]. CDF (Channel Definition Format) permits a publisher to offer frequently-updated collections of information from any server on the Web for automatic delivery. As an XML application, the CDF specification allows the publisher to specify his/her channels, the content available, the update schedule, and other information. For use in automated software distribution, OSD (Open Software Description Format) defines software packages and their interdependencies. RDF (Resource Description Format) Core Schema is a common set of properties

that may be optionally used by all other meta-data schemas such as XML-Data [18]. Web Interface Definition Language (WIDL) is to describe the inputs and outputs of programs on the Web. It is a meta-data syntax implemented in XML that defines application programming interfaces to data and services, enabling automatic and structured web access by compatible client programs, including mainstream business applications.

Table 1. XML Applications

Area	Standard/Proposal Name	Major Function
Data Management	XML-Data	Meta-data
	MCF (Meta Content Framework)	Meta-data
Multimedia	SMIL (Synchronized Multimedia Integration Language)	Presentation
	VML (Virtual Markup Language), PGML (Precision Graphics Markup Language), WebSchematics	Vector Graphics
	MusicML (Music Markup Language)	Sheet Music
	JSML (Java Speech Markup Language)	Speech
Internet, S/W	RDF (Resource Description Framework)	Resource Management
	WIDL (Web Interface Definition Language)	Web Interface
	OSD (Open Software Description)	Software Management
	CDF (Channel Definitin Format)	Push
	WebDAV (Web Distributed Authoring and Versioning)	Web Authoring
	XMI (XML Metadata Interchange)	UML
	PICS (Platform for Internet Content Selection)	Content Selection
	P3P (Platform for Privacy Preferences)	Privacy
EDI/EC	OFX (Open Financial Exchange)	Finance
	XML/EDI	EDI
	OTP (Open Trading Protocol)	Trading
	ICE (Information Content Exchange)	Digital Content
	PIX (Product Information Exchange)	Product Information
	OBI (Open Buying on the Internet)	B-to-B Trading
	CBL (Common Business Language)	Supply Chain
Science/ Education	CML (Chemical Markup Language)	Chemical Structure
	MathML (Mathematical Markup Language)	Mathematical Expression
	BSML (Bioinformatic Sequence Markup Language)	Biological Information
	TIM (Telecommunication Interchange Markup)	Technical Document
	TML (Tutorial Markup Language)	Tutorial
Language / Knowledge Representation	TMX (Translation Memory eXchange)	Translation
	OML (Ontology Markup Language)	Ontology
	CKML (Conceptual Knowledge Markup Language)	Conceptual Knowledge

Scientific applications of XML are also active. CML (Chemical Markup Language) [21] uses XML to manage complex molecular information. MathML (Mathematical Markup Language) [13] is an XML application for describing the structure and content of mathematical expressions, allowing the markup of complex formulas, which mathematicians and computer scientists have been clamoring for since the earliest days of HTML.

EC (Electronic Commerce) / EDI (Electronic Data Interchange) community is eagerly seeking for standard representation of product/catalog information and trading protocols. They include OFX (Open Financial eXchange) [4], OTP (Open Trading Protocol) [24], OBI (Open Buying on the Internet) [23], XML/EDI [30], and CBL (Common Business Language) [27]. Note that there is ongoing effort to integrate competing standards and/or proposals based on XML for improving its effectiveness and applicability. BITS [11], for instance, is driving the integration of OFX and GOLD [12] for financial transaction exchange. Another important aspect is the need to standardize some basic business logic in addition to that of content (e.g., product and catalog) information. This paper just deals with a language for sharing models within the MSOR (Management Science and Operations Research) / DSS (Decision Support System) community.

Each XML document contains one or more elements, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements by an empty-element tag. Each element has a type, identified by name (sometimes called its generic identifier), and may have a set of attributes. Each attribute has a name and a value. For example, element '100' has the type of 'Price' and its value of attribute 'Unit' is 'D'. Furthermore, elements have to be structured following the rule of referred DTDs. A 'Catalog', for instance, is composed of one or more 'Book's and they must have 'Title' and 'Price' elements. Refer to Bray et al. [2] for XML details.

4. An XML-based Simulation Language: SimX

For sharing simulation models on the Web, this paper proposes a structured markup language, SimX, based on the XML. XML DTD for the SimX is specified in Appendix A. The language will be illustrated using a simple banking system.

4.1 An Illustrative Example

A simple banking system has 2 tellers providing banking services as in figure 2. More details of the example can be found in DEVSim++ manual [16]. Figure 3 shows a coupled model containing the experimental frame EF and the system model BANK. The former has two atomic models, GENR and TRANSD, as components, while the latter consists of three atomic models, QUEUE0, TELLER0, and TELLER1. The GENR is an input generator and the TRANSD model is an output data collector. The QUEUE0 represents a waiting line and the two TELLERs correspond to the bank tellers.

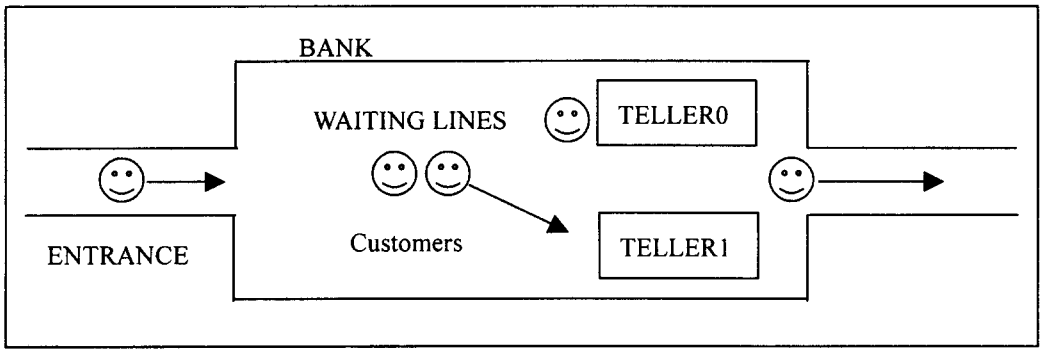

Figure 2. A Banking System (Adapted from DEVSim++ [16])

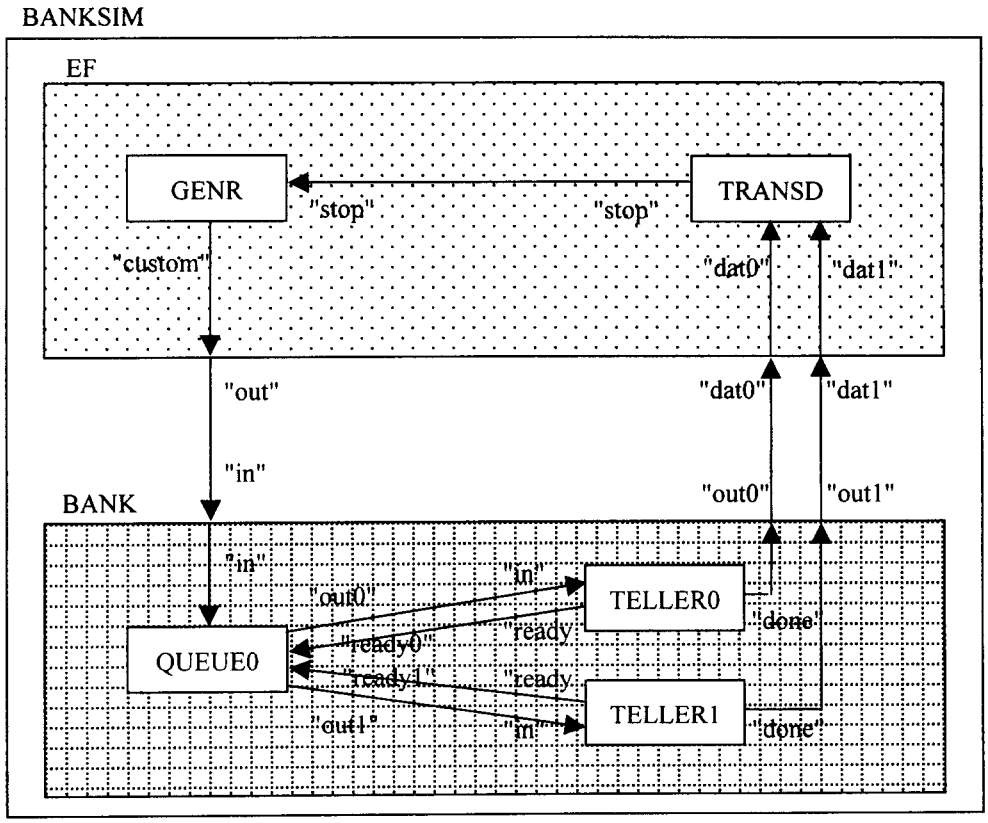

Figure 3. A DEVS Model for the Banking System (Adapted from DEVSim++ [16])

4.2 Example Modeling

Object Specification

Objects are parcels carrying information via messages. We need to specify such an object to correctly define interactions between models. An object has a 'name' attribute and 'var' elements. Each variable element has 'name' and 'type' attributes. The 'type' attribute identifies the type of values that the variable can retain. It can have a default element for describing its default value. Figure 4 shows the specification of two objects: 'Customer' and 'Message'.

```
<Object name="Customer">
    <Var name="ID" type="i4"/>
    <Var name="EntranceTime" type="time"/>
    <Var name="StartTime" type="time"><Default>0</Default></Var>
</Object>
<Object name="Message">
    <Var name="Port"/>
    <Var name="Value"/>
    <Var name="Time"/>
</Object>
```

Figure 4. Object Specification

```
<Ports>
    <Port name="Stop" type="in"></Port>
    <Port name="Customer" type="in"></Port>
</Ports>
<StateVars>
    <StateVar name="Phase"><Default>ACTIVE</Default></StateVar>
    <StateVar name="CustomerID"><Default>0</Default></StateVar>
</StateVars>
```

Figure 5. Specification of 'Ports' and 'StateVar'

Atomic Model Specification

An atomic model is represented as an 'AtomicModel' element. Generally speaking, it has two kinds of sub-elements. The one is that of static elements like 'Ports' and 'StateVars'. Figure 5 shows how they are specified. A 'Ports' groups 'Port' elements more than or equal to one. A 'Port' has two attributes, 'name' and 'type'. The 'type' attribute identifies whether the element is an input or output port.

The other is that of dynamic elements like 'ExtTransFtn' and 'IntTransFtn'. Figure 6 shows the specification of an 'ExtTransFtn'. Note that we call them dynamic elements because they have to capture transformation functions. Each one is represented as a set of 'CA' elements which is equivalent to a pair of 'Condition' and 'Action' elements. 'Condition' elements are boolean expressions where relational operators are used to make a compound expression. In figure 6, 'Equal' operator is applied for comparing

the 'Port' variable of the 'Message' object with the constant value of 'STOP'. Note that we are recursively using 'Reln' and 'Apply' elements for expressing relational and functional operations, respectively. They include one relational or functional operator as their first sub-element. This kind of mathematical expression follows the scheme of MathML [13] to enhance flexibility in representation and manipulation. 'Action' elements assign a value to a variable. For example, state variable 'Phase' is set to 'STOP', in the 'Action' element of figure 6. When a variable have to be set to a new object instance, we need to create an object instance. Figure 7 shows such a case in specifying an 'Action' element. A 'Create' operator is accompanied by a target object specification including its name and values for instance variables. Figure 8 demonstrates the use of a 'TimeAdvance' element for describing time advance functions. An 'Exponential' operator is used for generating a value according to the exponential distribution with the mean arrival time of 100.

```
<ExtTransFtn>
  <CA>
    <Condition>
      <Reln>
        <EQ/>
        <Apply>
          <Get/>
          <Object name="Message" var="Port"/>
        </Apply>
        <Value>STOP</Value>
      </Reln>
    </Condition>
    <Action>
      <Apply>
        <Set/>
        <Var name="Phase"/>
        <Value>STOP</Value>
      </Apply>
    </Action>
  </CA>
</ExtTransFtn>
```

Figure 6. Specification of an External Transformation Function

```
<OutputFtn>
  <CA>
    <Condition>
      <Reln>
        <EQ/>
        <Apply><Get/><Var name="Phase"/></Apply>
        <Value>ACTIVE</Value>
      </Reln>
    </Condition>
    <Action>
      <Apply>
        <Set/>
        <Object name="Message" var="Port"/>
        <Apply>
          <Create/>
          <Instance object="Customer">
            <Param var="ID">
              <Apply>
                <Get/>
                <Var="CustomerID"/>
              </Apply>
            </Param>
            <Param var="EntranceTime">
              <Apply>
                <Get/>
                <Object name="Message" var="Time"/>
              </Apply>
            </Param>
          </Instance>
        </Apply>
      </Apply>
    </Action>
  </CA>
</OutputFtn>
```

Figure 7. Specification of an Output Function

```
<TimeAdvanceFtn>
  <CA>
    <Condition>
      <Reln>
        <EQ/>
        <Apply>
          <Get/>
          <Var name="Phase"/>
        </Apply>
        <Value>ACTIVE</Value>
      </Reln>
    </Condition>
    <Action>
      <Apply>
        <TimeAdvance/>
        <Apply>
          <Exponential/>
          <Value>100</Value>
        </Apply>
      </Apply>
    </Action>
  </CA>
</TimeAdvanceFtn>
```

Figure 8. Specification of a Time Advance Function

Coupled Model Specification

A coupled model is composed of component models and I/O ports. As figure 9 shows, a 'Components' element groups a set of component models. Each component model is specified as a 'Component' element. It has two attributes, 'object' and 'name'. A 'Ports' element groups a set of I/O ports. Each port is specified as a 'Port' element. It has two attributes, 'name' and 'type'. The 'type' attribute identifies whether it is an input or output port. Additionally, 'Coupling' elements match input and output 'Port' elements. Such elements are grouped using a 'Couplings' element. As an example, figure 9 shows a coupling between the 'in' port of 'Bank' and the 'in' port of 'Queue'.

```
<CoupledModel object="BANK">
    <Components>
        <Component object="Queue" name="Queue0" src="Queue.xml"/>
        <Component object="Teller" name="Teller0" src="Teller.xml"/>
        <Component object="Teller" name="Teller1" src="Teller.xml"/>
    </Components>
    <Ports>
        <Port name="in" type="in"/>
        <Port name="out0" type="out"/>
        <Port name="out1" type="out"/>
    </Ports>
    <Couplings>
        <Coupling>
            <CPort name="in"/>
            <CPort object="Queue" name="in"/>
        </Coupling>
        ...
    </Couplings>
</CoupledModel>
```

Figure 9. Specification of a Coupled Model

5. General Discussion

The example specification has led to diverse points of discussion. Because the specification is based on XML markups and there are many standard tools and utilities, processing and analysis for validation and conversion does not require much effort. Figure 10, for example, shows the analysis of the GENR atomic model with Microsoft® XML Notepad. Furthermore, simulation models can be dynamically analyzed on the Web using ubiquitous Web browsers. Figure 11 shows the tree view of the BANK model. DOM API plays the main role for complex manipulation of objects within scripts of the Web pages. Figure 12 shows a modular structure derived from the BANK model. Note that component models are currently integrated into its parent model on the server side, using a server-side script language, ASP [7].

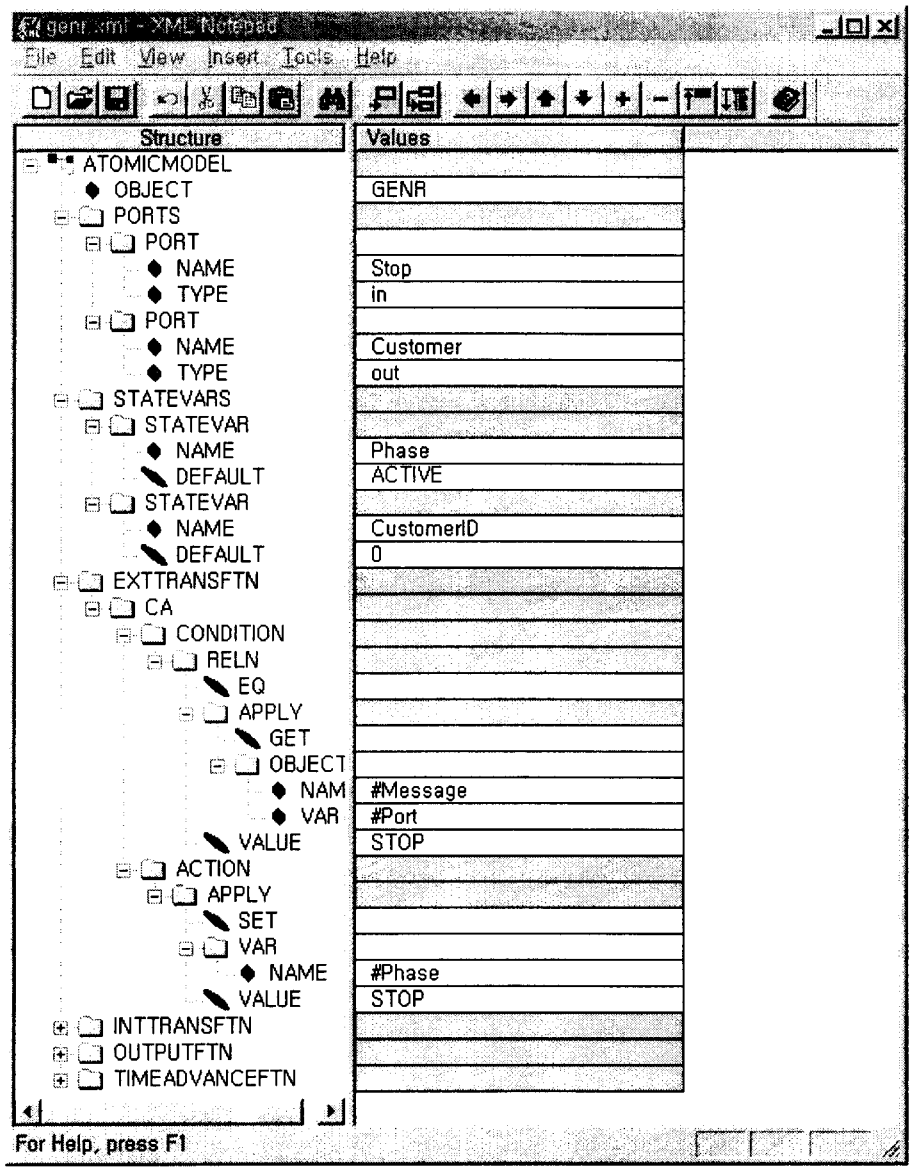

Figure 10. Analysis of the GENR Model

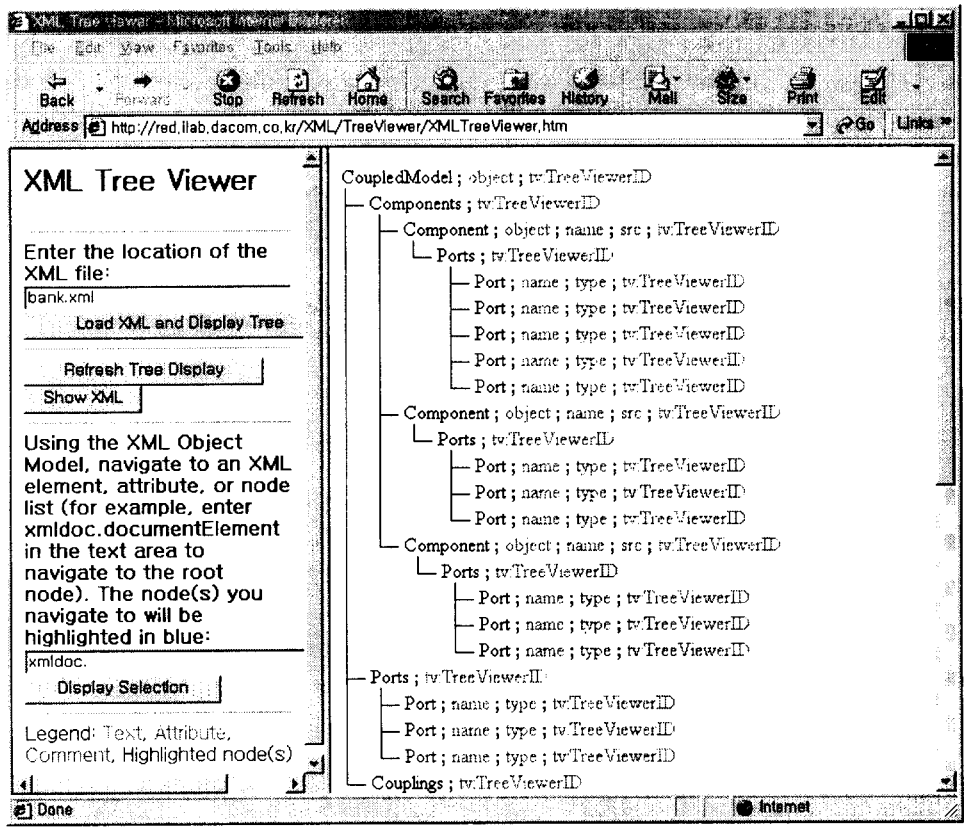

Figure 11. Analysis of the BANK Model

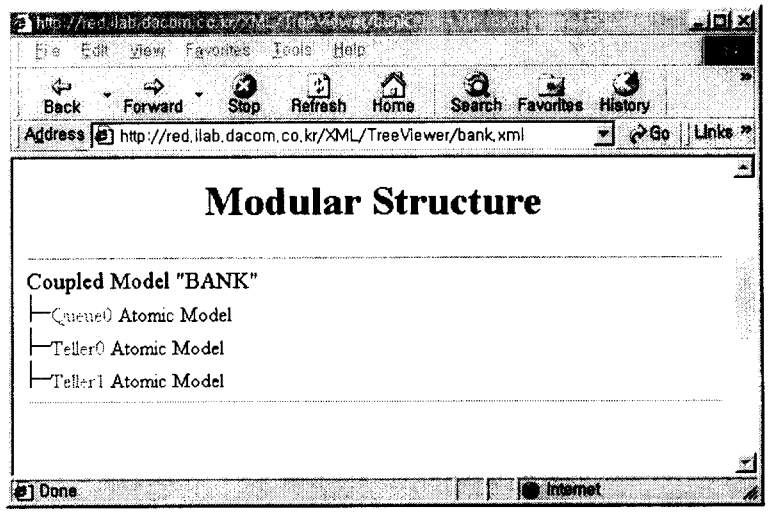

Figure 12. Modular Structure Analysis

Simple and structured analysis based on XSL is also useful for local manipulation and conversion of simulation models. A default XSL document converts an XML document into an HTML file to show the source of the document whenever we view it without any specific XSL document on the Web. Figure 13 is an XSL document for analyzing the validity of each interconnection of ports. Note that XSL documents are also XML documents. They just traverse the tree structure of an XML document and match/select some components. In the figure, a set of <xsl:for-each> and </xsl:for-each> tags define that the contents between the two tags should be repeated for each matching element. Figure 14 shows the result when we apply the XSL document to the BANK model. Complex semantic constraints can also be applied to those simulation models based on such an XSL document. Note that such constraints are largely related with the DEVS framework.

```
<?xml version="1.0"?>

<HTML xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<BODY>
<CENTER><H1>Port Analysis</H1>
<hr size="1"/>
Model "<xsl:value-of select="/CoupledModel/@object"/>"
<TABLE border="1" cellspacing="0">
<TR><TD>Ref. Model</TD><TD>Port Name</TD><TD>In/Out</TD>
<TD>Ref. Model</TD><TD>Port Name</TD><TD>In/Out</TD></TR>
<xsl:for-each select="//Coupling"><TR>
<xsl:for-each select="./CPort"><TD>
<xsl:if match=".[@comp]"><xsl:value-of select="./@comp"/></xsl:if>
<xsl:if match=".[not(@comp)]">self</xsl:if>
</TD>
<TD><xsl:value-of select="./@name"/></TD>
<TD>
<xsl:choose>
<xsl:when match=".[@comp]">
<xsl:for-each select="//Component[@name=context()/@comp]">
<xsl:for-each select="./Ports/Port[@name=context(-2)/@name]">
<xsl:value-of select="./@type"/>
</xsl:for-each></xsl:for-each>
</xsl:when>
<xsl:otherwise>
<xsl:for-each
select="/CoupledModel/Ports/Port[@name=context()/@name]">
<xsl:value-of select="./@type"/>
</xsl:for-each>
</xsl:otherwise>
</xsl:choose>
</TD></xsl:for-each></TR></xsl:for-each>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

Figure 13. XSL Source for Port Analysis

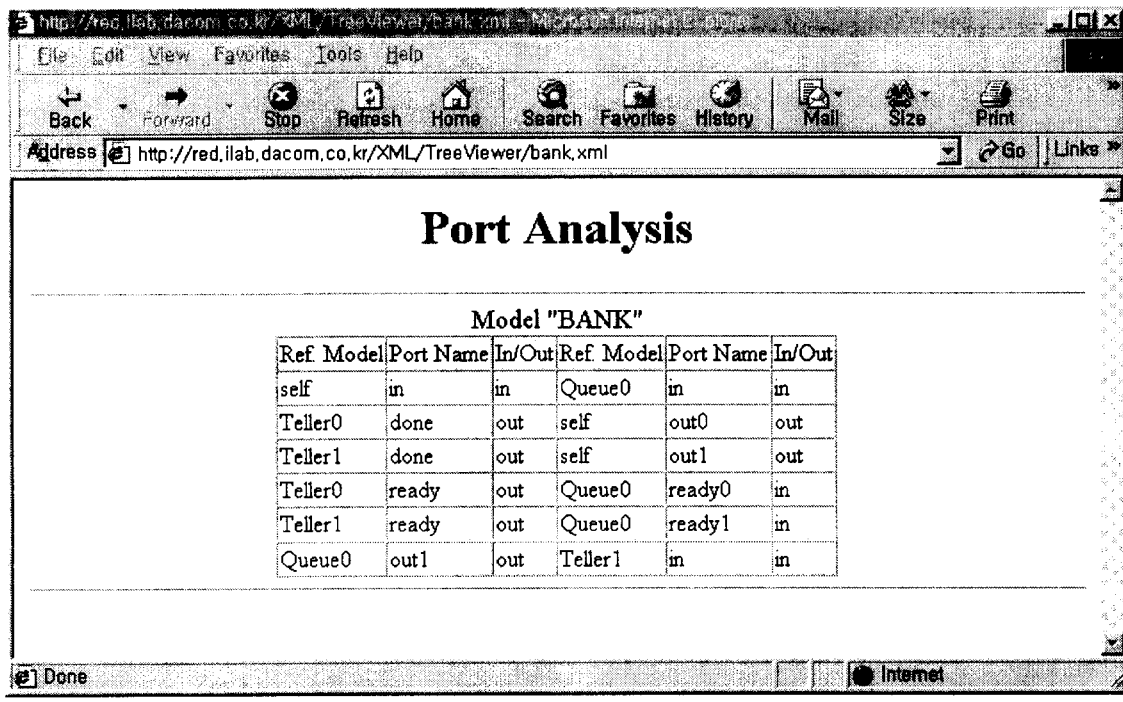

Figure 14. Port Analysis

With the help of the meaningful tags, simulation models can be easily understood by different kinds of users on the Web. Simulation models expressed by SimX, however, become more voluminous than those of traditional simulation languages because of the meaningful start and end tags. Another thing to be mentioned is about tradeoff between ways to design an XML-based language, especially in expressing mathematical relationships. SimX follows the approach of MathML, which is based on the concept of an expression tree. The tree structure of a mathematical expression is directly encoded by the model elements. The 'Apply' element is perhaps the most important mathematical element. It is used to apply a function to a collection of arguments. The positions of the child schemata is again significant, with the first child denoting the function to be applied, and the remaining children denoting the arguments of the function, with order preserved. Note that the 'Apply' construct always uses prefix notation.

5. Concluding Remarks

This paper presents a structured modeling language for sharing simulation models on the Web. We adopt DEVS as a simulation modeling formalism and XML as a representation /implementation language. The former provides a neutral framework that component models can be produced. The latter provides an easy way to design a tag-based language, SimX, of which models can be shared on the Web without loss of knowledge about models. Furthermore, we have demonstrated how to analyze SimX models using DOM API and XSL.

Currently we are working on the development

of a general-purpose Java agent that can interpret and execute SimX models. Just by selecting component models, it would be possible to implement a local simulation agent. Compared with Java simulation programs, users are able to search or analyze simulation contents expressed in SimX. Trading of such models will be another new business chance in the age of electronic commerce. The language can also be used for supporting cooperative simulation modeling on the Web, generating conventional simulation programs, and facilitating distributed coordination of simulation models.

References

[1] Allen, C., WIDL: Application Integration with WIDL, http://www.webmethods.com/technology/widl.html, 1997.

[2]Bray, T., J. Paoli and C.M. Sperberg-McQueen, Extensible Markup Language (XML) 1.0, http://www.w3.org/TR/1998/REC-xml-19980210, 1998.

[3] Buss, A.H. and K.A. Stork, "Discrete Event Simulation on the World Wide Web Using Java," Proceedings of the 1996 Winter Simulation Conference, Coronado, CA, 1996, pp. 780-785.

[4] CheckFree et al., Open Financial Exchange Specification 1.5, http://www.ofx.net/ofx/default.asp, 1998.

[5] Cho, H.J. and Y.K. Cho, *DEVS-Java Reference Guide*, The University of Arizona, 1997.

[6] Ellerman, C., Channel Definition Format (CDF), http://www.w3.org/TR/NOTE-CDFsubmit.html, March 1997.

[7] D. Fleet, M. Warren, J. C.-H. Chen, and A. Stojanovic, "Teach Yourself Active Web Database Programming in 21 Days", SAMS Publishing , May 1997.

[8] Healy, K.J. and R.A. Kilgore, "Introduction to Silk and Java-based Simulation," Proceedings of the 1998 Winter Simulation Conference, Piscataway, NJ, 1998.

[9] Hoff, A., H. Partovi, T. Thai, Open Software Description Format (OSD), http://www.w3.org/TR/NOTE-OSD, August 1997.

[10] Howell, F. and R. McNab, "SimJava: a Discrete Event Simulation Package for Java with Applications in Computer Systems Modeling," Proceedings of the First International Conference on Web-based Modeling and Simulation, San Diego, CA, Society for Computer Simulation, January 1998.

[11] Integrion Financial Network, Integrion News: BITS, http://www.integrion.com/news/story040798.html, 1998.

[12] Integrion Financial Network, The GOLD Standard, http://www.integrion.com/gold/index.html,, 1998.

[13] Ion, P. and R. Miner, Mathematical Markup Language, http://www.w3.org/TR/WD-math, January 1998.

[14] ISO, Information processing -- Text and office systems -- Standard Generalized Markup Language (SGML), Standard 8879, 1986.

[15] Kim, T.G., "Hierarchical Class Development in the DEVS-Scheme Simulation Environment," *Expert Systems with Applications*, Vol. 3, No. 3, 1991, pp. 343-351.

[16] Kim, T.G., *DEVSim++ User's Manual*, Korea Advanced Institute of Science and Technology, 1994.

[17] Lassila, O., Resource Description Framework (RDF), http://www.w3.org/RDF, February 1998.

[18] Layman, A. et al., XML-Data, http://www.w3.org/TR/1998/NOTE-XML-data, January 1998.

[19]Little, M.C., "The C++SIM Home Page," http://cxxsim.ncl.ac.uk/, 1997.

[20] Mathews, B. et al., Vector Markup Language, http://www.w3.org/TR/NOTE-VML, May 1998.

[21] Murray-Rust, P., Chemical Markup Language (CML) 1.0, http://www.venus.co.uk/omf/cml/, January 1997.

[22] Nair, R.S., "JSim: A Java-Based Query Driven Simulation and Animation Environment," MSc Thesis, Univ. of Georgia, 1997.

[23] OBI Cnsortium, Open Buying on the Internet (OBI) Technical Specifications Release V1.1, http://www.openbuy.org/obi/library/, 1998.

[24] OTP Consortium, Open Trading Protocol, http://www.otp.org, 1998.

[25] Page, E.H., R.L. Moose and S.P. Griffin, "Web-based Simulation in SimJava Using Remote Method Invocation," Proceedings of the 1997 Winter Simulation Conference, Atlanta, GA, December 1997, pp. 468-474.

[26] Thomas, C., H. Luckhoff, and T.G. Kim, "OpenDEVS: A Proposal for a Standardized DEVS Model Exchange Format," *AIS'96*, San Diego, CA, Mar. 23, 1996, pp. 371-377.

[27] Veo Systems, The XML Revolution in Internet Commerce, http://www.veosystems.com/white_paper.htm, 1998.

[28] W3C, Document Object Model (DOM) Level 1 Specification, http://www.w3.org/ TR/REC-DOM-Level-1/ , October 1998.

[29] W3C, Synchronized Multimedia Integration Language (SMIL) 1.0 Specification, http://www.w3.org/TR/REC-smil, June 1998.

[30] XML/EDI Group, XML/EDI, http://www.xmledi.com, 1998.

[31] Ziegler, B, *Theory of Modeling and Simulation*, Robert E. Krieger Publishing Company, 1976.

[32] Ziegler, B, *Multifacetted Modeling and Discrete Event Simulation*, Academic Press, Orlando, FL, 1984.

Appendix A. XML DTDs for SimX

<DTD for Objects>

```
<!ELEMENT Object Var+>
<!ATTLIST Object name ID #REQUIRED>

<!ELEMENT Var Default?>
<!ATTLIST Var name ID #REQUIRED
      type CDATA #IMPLIED>

<!ELEMENT Default #PCDATA>
```

<DTD for Atomic Models>

```
<!ENTITY % nameAttr 'name ID #REQUIRED'>

<!ELEMENT AtomicModel (Ports,StateVars,ExtTransFtn,IntTransFtn,OutputFtn,TimeAdvanceFtn)>
<!ATTLIST AtomicModel object ID #REQUIRED>

<!ELEMENT Ports Port*>
<!ELEMENT Port EMPTY>
<!ATTLIST Port %nameAttr;>

<!ELEMENT StateVars StateVar*>
<!ELEMENT StateVar Default?>
<!ATTLIST StateVar %nameAttr;>

<!ELEMENT Default #PCDATA>

<!ELEMENT ExtTransFtn CA+>
<!ELEMENT IntTransFtn CA+>
<!ELEMENT OutputFtn CA+>
<!ELEMENT TimeAdvanceFtn CA+>

<!ELEMENT CA (Condition,Action)>

<!ELEMENT Condition Reln>
<!ELEMENT Action Apply+>

<!ENTITY % BinOp '((Plus|Minus|Times|Divide|Power),(Apply|Value),(Apply|Value))'>
<!ENTITY % SetOp '(Set,(Object|Var),(Apply|Value))'>
<!ENTITY % GetOp '(Get,(Object|Var))'>
<!ENTITY % CreateOp '(Create,Instance)'>
<!ENTITY % TimeOp '(TimeAdvance, (Apply|Value))'>
<!ENTITY % StatOp '(Exponential,(Apply|Value))'>
<!ELEMENT Apply (%BinOp;|%SetOp;|%GetOp;|%CreateOp;|%TimeOp;|%StatOp;)>

<!ENTITY % CompReln '((EQ|GT|GE|LE|LT),(Apply|Value),(Apply|Value))'>
<!ENTITY % BinReln '((AND|OR),Reln,Reln)'>
```

```
<!ENTITY % NegReln '(NOT,Reln)'>
<!ELEMENT Reln (%CompReln;|%BinReln;|%NegReln;)>

<!ELEMENT Plus EMPTY>
<!ELEMENT Minus EMPTY>
<!ELEMENT Times EMPTY>
<!ELEMENT Divide EMPTY>
<!ELEMENT Power EMPTY>
<!ELEMENT Create EMPTY>
<!ELEMENT TimeAdvance EMPTY>
<!ELEMENT Exponential EMPTY>
<!ELEMENT EQ EMPTY>
<!ELEMENT GT EMPTY>
<!ELEMENT GE EMPTY>
<!ELEMENT LT EMPTY>
<!ELEMENT LE EMPTY>
<!ELEMENT AND EMPTY>
<!ELEMENT OR EMPTY>
<!ELEMENT NOT EMPTY>

<!ELEMENT Instance Param+>
<!ATTLIST Instance object IDREF #REQUIRED>

<!ELEMENT Param Apply>
<!ATTLIST Param var IDREF #REQUIRED>
```

<DTD for Coupled Models>

```
<!ELEMENT CoupledModel (Components,Ports,Couplings)>
<!ATTLIST CoupledModel object ID #REQUIRED>

<!ELEMENT Components (Component)+>
<!ELEMENT Component EMPTY>
<!ATTLIST Component model IDREF #IMPLIED
                    name ID #REQUIRED
                    src CDATA #REQUIRED>

<!ELEMENT Ports (Port)+>
<!ELEMENT Port EMPTY>
<!ATTLIST Port name ID #REQUIRED
                    type (in|out) "in">

<!ELEMENT Couplings (Coupling)+>
<!ELEMENT Coupling   (CPort, CPort)>
<!ELEMENT CPort EMPTY>
<!ATTLIST CPort comp IDREF #IMPLIED
                    name IDREF #REQUIRED>
```

Appendix B. Specification of GENR

```
<?xml version="1.0"?>
<!DOCTYPE AtomicModel SYSTEM "atomic.dtd">
<AtomicModel object="GENR">
    <Ports>
        <Port name="Stop" type="in"></Port>
        <Port name="Customer" type="out"></Port>
    </Ports>
    <StateVars>
        <StateVar name="Phase"><Default>ACTIVE</Default></StateVar>
        <StateVar name="CustomerID"><Default>0</Default></StateVar>
    </StateVars>
    <ExtTransFtn>
        <CA>
            <Condition>
                <Reln>
                    <EQ/>
                    <Apply>
                        <Get/>
                        <Object name="Message" var="Port"/>
                    </Apply>
                    <Value>STOP</Value>
                </Reln>
            </Condition>
            <Action>
                <Apply>
                    <Set/>
                    <Var name="Phase"/>
                    <Value>STOP</Value>
                </Apply>
            </Action>
        </CA>
    </ExtTransFtn>
    <IntTransFtn>
        <CA>
            <Condition>
                <Reln>
                    <EQ/>
                    <Apply>
                        <Get/>
                        <Var name="Phase"/>
                    </Apply>
                    <Value>ACTIVE</Value>
                </Reln>
            </Condition>
            <Action>
                <Apply>
                    <Set/>
                    <Var name="Phase"/>
                    <Value>ACTIVE</Value>
```

```
            </Apply>
            <Apply>
                <Set/>
                <Var name="CustomerID"/>
                <Apply>
                    <Plus/>
                    <Apply>
                        <Get/>
                        <Var name="CusotmerID"/>
                    </Apply>
                    <Value>1</Value>
                </Apply>
            </Apply>
        </Action>
    </CA>
<,/IntTransFtn>
<OutputFtn>
    <CA>
        <Condition>
            <Reln>
                <EQ/>
                <Apply>
                    <Get/>
                    <Var name="Phase"/>
                    </Apply>
                <Value>ACTIVE</Value>
            </Reln>
        </Condition>
        <Action>
            <Apply>
                <Set/>
                <Object name="Message" var="Port"/>
                <Apply>
                    <Create/>
                    <Instance object="Customer">
                        <Param var="ID">
                            <Apply>
                                <Get/>
                                <Var name="CustomerID"/>
                            </Apply>
                        </Param>
                        <Param var="EntranceTime">
                            <Apply>
                                <Get/>
                                <Object name="Message" var="Time"/>
                            </Apply>
                        </Param>
                    </Instance>
                </Apply>
            </Apply>
        </Action>
    </CA>
```

```
    </OutputFtn>
    <TimeAdvanceFtn>
        <CA>
            <Condition>
                <Reln>
                    <EQ/>
                    <Apply>
                        <Get/>
                        <Var name="Phase"/>
                    </Apply>
                    <Value>ACTIVE</Value>
                </Reln>
            </Condition>
            <Action>
                <Apply>
                    <TimeAdvance/>
                    <Apply>
                        <Exponential/>
                        <Value>100</Value>
                    </Apply>
                </Apply>
            </Action>
        </CA>
        <CA>
            <Condition>
                <Reln>
                    <NOT/>
                    <Reln>
                        <EQ/>
                        <Apply>
                            <Get/>
                            <Var name="Phase"/>
                        </Apply>
                        <Value>ACTIVE</Value>
                    </Reln>
                </Reln>
            </Condition>
            <Action>
                <Apply>
                    <TimeAdvance/>
                    <Value>INFINITY</Value>
                </Apply>
            </Action>
        </CA>
    </TimeAdvanceFtn>
</AtomicModel>
```

Appendix C. Specification of BANK

```
<?xml version="1.0"?>
<!DOCTYPE CoupledModel SYSTEM "coupled.dtd">
<CoupledModel object="Bank">
    <Components>
        <Component model="Queue" name="Queue0" src="queue0.xml"/>
        <Component model="Teller" name="Teller0" src="teller0.xml"/>
        <Component model="Teller" name="Teller1" src="teller1.xml"/>
    </Components>
    <Ports>
        <Port id="in" type="in"/>
        <Port id="out0" type="out"/>
        <Port id="out1" type="out"/>
    </Ports>
    <Couplings>
        <Coupling>
            <CPort name="in"/>
            <CPort comp="Queue0" name="in"/>
        </Coupling>
        <Coupling>
            <CPort comp="Teller0" name="done"/>
            <CPort name="out0"/>
        </Coupling>
        <Coupling>
            <CPort comp="Teller1" name="done"/>
            <CPort name="out1"/>
        </Coupling>
        <Coupling>
            <CPort comp="Teller0" name="ready"/>
            <CPort comp="Queue0" name="ready0"/>
        </Coupling>
        <Coupling>
            <CPort comp="Teller1" name="ready"/>
            <CPort comp="Queue0" name="ready1"/>
        </Coupling>
        <Coupling>
            <CPort comp="Queue0" name="out0"/>
            <CPort comp="Teller0" name="in"/>
        </Coupling>
        <Coupling>
            <CPort comp="Queue0" name="out1"/>
            <CPort comp="Teller1" name="in"/>
        </Coupling>
    </Couplings>
</CoupledModel>
```

저자소개

김형도

Kim, HyoungDo received his Ph.D and M.S. from Korea Advanced Institute of Science and Technology(KAIST), majoring in Management Science. He received his B.S. in Industrial Engineering from Seoul National University. He has been working for Dacom Corp. in the research of EC(Electronic Commerce) and Internet. His current research areas include decision support systems, object-oriented modeling and simulation, workflow support, personalized multimedia services, and XML-based EC applications.

김종우

Kim, JongWoo is an assistant professor of Statistics at Chungnam National University. He received B.S in Mathematics from Seoul National University. He received M.S. in Management Science and Ph.D. in Industrial Management from KAIST. His current research areas include decision support systems, object-oriented software engineering, business process reengineering, workflow systems, and EC.