DEVS Framework for Component-based Modeling/Simulation of Discrete Event Systems

Young Ik Cho and Tag Gon Kim
Systems Modeling Simulation Lab
Department of Electrical Engineering and Computer Science
Korea Advanced Institute of Science and Technology (KAIST)
373-1 Kusong-dong, Yusong-gu, Taejon 305-701, Korea
Yicho@smslab.kaist.ac.kr, tkim@ee.kaist.ac.kr

Keywords: Component, reusability, DEVS, COM

Abstract
This paper applies a component-based framework to discrete event systems simulation and then develops a component-based simulation environment. The environment is based on combination of the sound modeling formalism of DEVS (Discrete Event Systems Specification) and the powerful component standard of COM (Component Object Model). The combination results in the DEVS/COM run-time infrastructure which supports binary reusability of simulation models developed with multi-languages in a modular form. It has been shown that such reusability provides modelers with flexibility in modeling as well as efficiency in models development time.

Introduction
The component-based framework, proposed by market, is a methodology to enable reusability on various development stages and discrete event simulation is a powerful tool for system design and the DEVS (Discrete Event System Specification) formalism proposed by Zeigler [1] provides sound formal modeling and simulation framework for the generic system.

There are several approaches each of which enables a DEVS (Discrete Event System Specification) framework to a component capability. The first approach is based on a Java component technology [2], which implements a visual modeling and simulation environment. The second approach is DEVS/CORBA execution environment [3] that focused on the distributed simulation of the network-friendliness of the component technology.

Both DEVS-JavaBean and DEVS/CORBA frameworks utilize the potential of component technology and their feature is strongly concerned with the component technology that each framework adopted. In this paper a suggested component-based simulation environment is based on the Microsoft's COM run-time infrastructure. COM is currently the strongest competitor on the component technology and it is a sound framework to interoperate under the many co-existing different programming language as a reusable binary component and its network friendliness may provide further extensibility for an application to be extended to a distributed environment. Combining the advantages of DEVS and COM may provide heterogeneous and reusable modeling and simulation environment.

In this paper, DEVS/COM component simulation environment is suggested to meet component reusability requirement. To set the stage the brief overview on the component technology and model reusability is provided. We then proceed to the design of DEVS/COM simulation environment with simple banking example. Some discussion about overall reusability will be followed.

COMPONENT TECHNOLOGY AND SIMULATION MODELS REUSABILITY

Component Technology and Supporting Runtime Infrastructures
Traditional software development requires application executables to be compiled and linked with their dependencies. Every time a developer wants to use a different processing logic or new capabilities, he or she needs to modify and recompile the primary application to support them. To manage with the rapid model change, the reuse of system components is strongly required to improve the software development productivity and quality.

Software components are binary units that are composed without modification [4]. Even though different definitions about the components are found on every other literature, a software component must meet the following properties

- A component is a unit of independent deployment
- A component is a unit of third-party composition
- A component has no persistent state

These properties mean that a software component is a unit of a reusable piece of software without any knowledge on the internal structure.

There are several component technologies that compete to gain the name of the standards. The three important players for component world are COM (Component Object

Model) proposed by Microsoft, JavaBean by Sun and CORBA (Common Object Request Broker Architecture) by OMG (Object Management Group). The each component approach are summarized in the following table:

Table 1. Characteristics of component approach

	COM	CORBA	JavaBeans
Platform	Microsoft's Windows[1]	Any	Any
Programming Language	Any	Any	Java
Approach to component[2]	Application data exchange format	Distributed environment	Programming language

The benefits from the component software, as embodied in the COM, are as follows: [5]

- Abstraction and black-box encapsulation
- Extensibility
- Reusability
- Cross language
- Location independence

Models Reusability in Component-Based Modeling/Simulation

A number of paradigms and programming methodologies have been offered to develop reusable and extensible code. One such paradigm, object-oriented methodology, has been received favorably in general. Under this paradigm, certain sets of related features are grouped in a single unit called an object and it creates an abstraction layer that simplifies many systems.

On the traditional procedure for modeling, the reusability is achieved by obtaining the part of the source code of the concern. On some case reusability is achieved by sharing some of the program module in a library form and program language-specific header file. With the innovation of the object-oriented paradigm, some aspects of the system can be considered as an object to be reused and that results in great reduction of source code [6]. But as the increasing size of the overall system and the number of the system designer involved, it is harder to knit each module to make a whole picture. In the object oriented development procedure, change on some object may propagate to other object and the overall system must be built in a synchronous fashion to avoid this problem.

The component-based development framework overcomes this development in large problem by immutable interface –a contract with other component cannot be modified- and binary form of component. Binary form provides true information hiding on the internal structure of the component that results in true black-box encapsulation and makes it possible for any programming language to make a component – language independence.

The full feature of the object-oriented modeling framework can be used in the component-based system designing and a proposed simulation model can be easily distributed as a form of a component. While reusing an object in the object-orient framework still applies in the component inside, the model component itself can be reused in a simple Lego-block way.

DEVS/COM ENVIRONMENT

A brief overview on the DEVS formalism is introduced in the first section and architecture and implementation of the DEVS/COM environment will be followed. Some capability on the cross language model implementation is discussed on the last section.

DEVS Formalism: A Brief Overview

The DEVS (Discrete Event System Specification) formalism introduced by Zeigler is a set-theoretic formalism and it provides a means of modeling discrete event system in a hierarchical, modular way. With this DEVS formalism, we can perform modeling more easily and correctly by decomposing large system into component models and specify the coupling between them. There are two kinds of model in DEVS formalism.

Atomic model is the basic model and has specification for the dynamics of the model. It describes the behavior of a component, which is indivisible, in a timed state transition level. Formally, a 7-tuple specifies an atomic model M as follows

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

Where

- X : input events set
- Y : output event set
- S : sequential states set
- $\delta_{int} : S \rightarrow S$ is the internal transition function
- $\delta_{int} : Q \times X \rightarrow$ is the external transition function, where
  - $Q = \{(s,e) | s \in S, 0 \le e \le ta(s)\}$ is the total state set
  - e is the time elapsed since last transition
- $\lambda : S \rightarrow Y$ is the output function
- $ta : S \rightarrow R^{+}_{0,\infty}$ is the time advance function, where
  - $R^{+}_{0,\infty}$ is the set positive reals with 0 and ∞

The four elements in the 7-tuple, namely δ_{int}, δ_{ext}, λ and ta, are called characteristic functions.

The second form of the model, called a coupled model (or coupled DEVS), tells how to couple several component models together to form a new model. This kind of model can be employed as a component in a larger coupled model,

[1] Microsoft's COM is available on other platforms like Solaris and Macintosh but it is not widely used on such a platform

[2] 'Approach to component' means 'from where these technologies are originated'

thus giving rise to the construction of complex models in a hierarchical fashion. The DEVS coupled model is defined as follows.

CM = < X, Y, M, EIC, EOC, IC, SELECT >

Where

X : input events set
Y : output events set
M : set of all component models in DEVS
EIC $\subseteq X \times \cup_i X_i$: external input coupling relation
EOC $\subseteq \cup_i Y_i \times Y$: external output coupling relation
IC $\subseteq \cup_i X_i \times \cup_i Y_i$: internal coupling relation
SELECT : $2^M - \phi$ à M is a function which chooses one model when more than 2 models are scheduled simultaneously.

Architecture of DEVS/COM environment

The Component Object Model (COM) is a Microsoft's component software architecture that allows applications and systems to be built from components supplied by different software vendors. COM specifies a component interface definition, which is independent from programming languages, such as C, C++, Java, and Visual Basic.

On the various reusability approaches like component composition and code generation, both DEVS framework and COM runtime infrastructure utilize component composition way. The hierarchical and modular specification of the DEVS formalism and the binary component of the COM component can be onto correspondence. That is, DEVS abstract simulator (Figure 1) and user model is directly correlated to a COM component, as there is no alteration on the simulation algorithm or message structure.

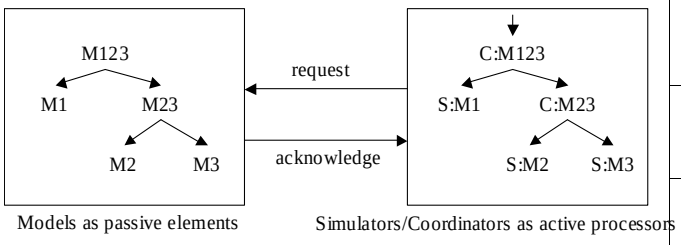

Figure 1. DEVS abstract simulator

On the DEVS abstract simulator both simulation user model and simulator/coordinator are reusable DEVS object, simulator is one to one corresponding to the component DEVS simulator, as is user model to the model component. However user model may be a single component or user model itself may utilize the reusability to be consisted of several sub model components.

DEVS Models Implementation in DEVS/COM

As component DEVS user model also is a component, an implementation of the user model is somewhat sophisticated and it can be some overhead to modeler. Like the inheritance of the object-oriented framework, COM supports a reusability and extensibility by a kind of binary inheritance called containment and aggregation even though COM does not support strict-sense implementation inheritance.

A basic skeleton model is provided to reduce the modeling load and modeler use the basic model by either containment or aggregation. Functions by basic skeleton model for the DEVS atomic and coupled model are as followed (Table 2, Table 3). The function names are not changed in the comparison with DEVS object oriented simulator. Actually, as DEVS/COM simulator communicates with user model by interface, these API's are the interfaces provided by DEVS/COM simulator.

Table 2. Component API's for Atomic Model

Atomic Model	Function
X	AddInport([in] BSTR input)
Y	AddOutport([in] BSTR output)
S	
δ_{ext}	ExtTransFn([in] double e, [in] Messages msg)
δ_{int}	IntTransFn()
λ	OutputFn([in] IMessages* msg)
Ta	double TimeAdvanceFn()

Table 3. Component API's for Coupled Model

Coupled Model	Function
X	AddInport([in] BSTR inport)
Y	AddOutport(([in] BSTR output)
M	AddChild[in] IUnknown* child)
EIC	AddCoupling([in] IUnknown* model1, [in] BSTR port1, [in] IUnknown* model2, [in] BSTR port2)
EOC	AddCoupling([in] IUnknown* model1, [in] BSTR port1, [in] IUnknown* model2, [in] BSTR port2)
IC	AddCoupling([in] IUnknown* model1, [in] BSTR port1, [in] IUnknown* model2, [in] BSTR port2)
SELECT	AddPriority([in] IUnknown* model);

Multi-Language Support

It is unrealistic to believe that any single programming language is best adequate for every task. For example, visual basic can hide sophisticated simulation model initialization code but results in inevitable performance degradation. Programming language independence on the COM framework may elevate the system modeling performance by adopting easy-to use program language such as java to implement the coupled model and performance-critical C++

language to an overall system bottleneck component. Figure 2 shows how the system interoperating with other model in a different language.

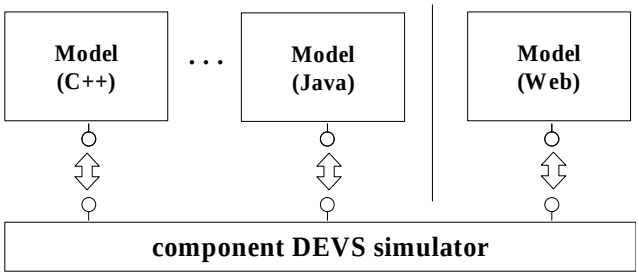

Figure 2. Cross Language Interoperability

Some comments about the rightmost model named Web are followed. Microsoft proposed some means of interoperability between component on the web server and client. This implies that a web-based simulation client can use model components without any modification on component even though it is not possible to make a new model component on the worldwide web.

COMPONENT-BASED MODELING/SIMULATION IN DEVS/COM

In this section we discuss how to use component-based simulation framework to do a discrete event simulation by an example and the benefits of the component-based environment.

To verify the proposed framework, a system of cache memory model is offered and comparison of input/output traces between DEVS/COM and existing C++-based DEVS simulator, DEVSim++ [7] validates the correctness. Effectiveness measures between DEVS/COM and DEVSim++ is also shown to compare performances of these simulators. The procedures are shown on figure 3.

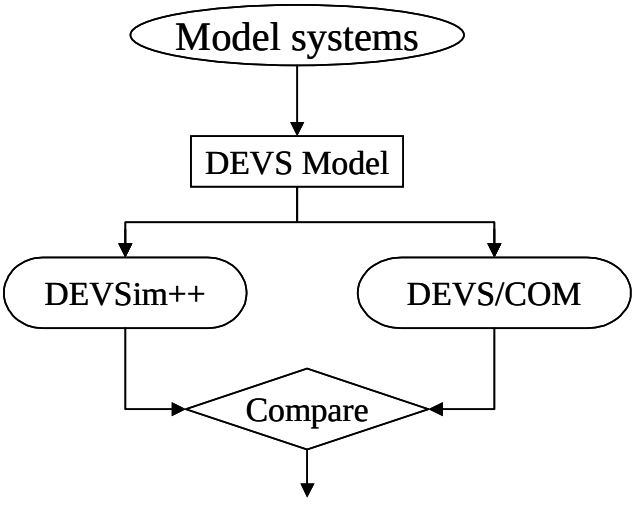

Figure 3. Validation and Verification Procedure

Cache System

System is configured with a CPU, a cache memory, a main memory, and an associated memory controller. The system objective is to undertake a performance evaluation with various cache sizes, cache block sizes, memory sizes, word sizes and CPU cycle time. Overall cache model is shown on figure 4. With this example, the input/output comparison and performance evaluation is done between DEVS/COM and DEVSim++ simulator.

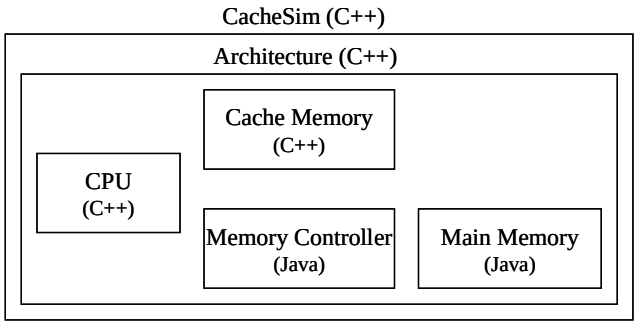

Figure 4. Cache System Coupled Model

COM runtime infrastructure offers an environment that multiple programming languages can be used to organize the whole system. To show multiple programming language interoperation, CPU and cache memory atomic mode are implemented in C++ programming language and memory controller and main memory atomic model are implemented in Java programming language. Multiple language interaction is depicted on figure 4. As CPU atomic model reads input traces files, it is implemented on C++ programming language to maximize I/O performance. Cache memory model is implemented on C++ to use system functions. Memory controller and main memory model are implemented on Java for easy implementation. Simplified source code for memory controller written in Java is shown on figure 5. There are redundant codes on the figure and this section is used to call simulation library component and need to be included by user to interact with simulator.

```
import codevs.*
public class MemController implements com.ms.com.IUnknown, IAtomic
{
    private IAtomic m_pAtomic;

    String status;
    int rw;

    public MemController() {
        m_pAtomic = = (IAtomic)new CoAtomic();

        m_pAtomic.setName("Memory Controller");
        m_pAtomic.AddInPorts("ca_req_in ");
        m_pAtomic.AddOutPorts("done");
        status = "WAIT";
        rw = 0;
    }
    public boolean ExtTransFn(IMessage pMsg) {
        if (pMsg.getPort()== "ca_req_in" && status == "WAIT")
            status = "BUSY";
        return true;
    }
    public boolean WhenRcvX(IAtomic env,IMessage pMsg) {
        return m_pAtomic.WhenRcvX((IAtomic) env, (IMessage) pMsg);
    }
}
```

state variable
input/output ports
state initialization
δ_{ext}
redundant code

Figure 5. Simplified Example Code

Characteristic functions of DEVS atomic model are checked on figure 5. Sequential state set on DEVS atomic model corresponds to state variable. Input/output event set are implemented as addinports/addoutports function. 4 characteristic functions of DEVS atomic model are need to be implemented on DEVS/COM atomic component by user but only external transition function is shown on figure 5.

Simulation and Analysis

To check for the correctness, several input sets with various scenarios are tested and input/output traces between DEVS/COM and DEVSim++ are compared. The results are shown on table 4.

Table 4. Simulation Results (Correctness)

Associativity	DEVSim++	DEVS / COM
1	70cycles(18/34/18)	70cycles(18/34/18)
2	37cycles(18/18/1)	37cycles(18/18/1)

The first instruction takes 18 cycles and the second one takes 34 as writeback operation. On the associativity 2, the second instruction does not need to perform a writeback operation and the last instruction can be read from the existing cache and so the last operation takes only one cycle.

For performance evaluation, following input vectors are used as an input instruction. Each of input traces are pseudo code of real-world application of c compiler, tex document formatting program, and spice simulation program in a specific format, *dinero* format[8]. Input vectors are listed on table 5 and the simulator execution performance comparison between DEVS/COM execution and DEVSim++ is shown on table 6.

Table 5. Input Instruction Sets

Input Type	Size	Characteristics
gcc.strace	9264kB	1000002 memory references
tex.strace	7776kB	832476 memory references
spice.strace	9214kB	1000001 memory references

Each of DEVSim++ and DEVS/COM simulator runs 10 times for each input trace with one random parameter on cache memory model and table 6 shows the average execution time for each simulation run. As shown on the table 6, the simulator execution speed of DEVS/COM environment is slower than DEVSim++ by 10 times. The possible reason for performance degradation is primarily the interface overhead of the component invokation and cross-context call overhead of DEVS/COM run-time environment. Despite this severe performance degradation, COM component technology is known to offer best execution performance in comparison with other component frame like CORBA or JavaBeans.

One benefit of DEVS/COM environment is that the whole system can be easily re-configured without any re-compiling or linking of the overall system; several atomic model changes to reflect changes in model parameters does not necessitate model rebuilding.

Table 6. Simulator Performance Comparison

Input Type	DEVSim++	DEVS/COM
gcc.strace	5 min 4 sec	48 min 5 sec
tex.strace	2 min 28 sec	22 min 34 sec
spice.strace	4 min 8 sec	37 min 16 sec

CONCLUSION AND FUTURE WORK

This paper presented a simulation environment for discrete event modeling and simulation using COM component technology. By employing DEVS/COM runtime environment, the following benefits can be expected. First, COM technology is an industry standards enabling integration of the modeling in the multi-language. Second, various system architectures can be a ready-to-deploy component that a user may re-use it. And component framework makes a model component independent with other component so possible problem in one component does not propagate into other model and each model can be distributed in a binary level without recompiling or linking by end user.

A component approach brings many benefits to the DEVS framework with some side effects. One is an interface overhead for a component and the other is performance degradation. DEVS/COM execution framework is about 10 times slower than non-component simulation environment. And both knowledge on DEVS formalism and component framework that are needed to build a robust component-based simulation environment yields other intrinsic problem is its implementation complexity. The most acceptable solution is to develop a graphical user interface to aid a modeler to make a model and to automate complicate but repeated component model implication. As component can be used a unit of deployment, this approach can be supported by component framework itself.

Like other network-oriented component technologies, COM component framework evolves to the DCOM (distributed COM) and COM+ to meet the current trends on the Internet. The model and simulator component may be easily used in the network environment with least modification. The comparison of distributed DEVS/COM with DEVS/CORBA and DEVS/HLA [9] (High Level Architecture) environment will be need on that time.

REFERENCES

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

[2] Yun-Hsin Wang and Szu-Hsuan Ho, "Implementation of a DEVS-JavaBean simulation environment", *Proceedings of Simulation Symposium, 2001. 34h Annual*, pp333-338

[3] Bernard P. Zeigler, Doohwan Kim, and Stephen J. Buckley, "Distributed supply chain simulation in a DEVS/CORBA execution environment", *Proceedings of the 1999 Winter Simulation Conference*, pp1333 – 1340
[4] Clemens Szyperski, *Component Software*, ACM Press, New York
[5] Tradeep Tapadiya, *COM+ Programming*, Prentice Hall PTR, Upper Saddle River, NJ 07458
[6] Ted J. Biggerstaff and Alan J. Perlis, *Software Reusability*, ACM Press, New York
[7] Tag Gon Kim, *DEVSim++ User's Manual*, 1994, ftp://sim.kaist.ac.kr/pub/.
[8] J. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, San Mateo, CA: Morgan Kaufman, 1990
[9] Zeigler, B.P. *Implementation of the DEVS Formalism over the HLA/RTI: Problems and Solutions*. In SIW. 1999. Orlando, FL.

BIOGRAPHY

YOUNG IK CHO received his M. S degree in the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea.

TAG GON KIM received his Ph.D. in computer engineering with specialization in methodology for systems modeling/simulation from University of Arizona, Tucson, AZ, 1988. He joined at Electrical Engineering Department of KAIST, Tajeon, Korea in Fall, 1991 as an Assistant Professor and has been a Full Professor since Fall, 1998. His research interests include methodological aspects of systems modeling simulation, analysis of computer/communication networks, and development of simulation environments. He has published more than 100 papers on systems modeling, simulation and analysis in international journals/conference proceedings. He is a co-author (with B.P. Zeigler and H. Praehofer) of *Theory of Modeling and Simulation* (2nd ed.), Academic Press, 2000. He is the Editor-in-Chief of *Transactions of Society for Computer Simulation*. He is a senior member of IEEE and SCS and a member of ACM and Eta Kappa Nu.