Proceedings of 2001 Seoul International Simulation Conference, pp.484-488, October, 2001, Seoul, Korea.

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

Young-Ik Cho, Jae-Hyun Kim and Tag Gon Kim

Systems Modeling Simulation Lab
Department of Electrical Engineering & Computer Science
KAIST
Daejon, KOREA
E-mail: {yicho, jhkim}@smslab.kaist.ac.kr , tkim@ee.kaist.ac.kr

ABSTRACT

The component-based framework in software development exploits modularity and reusability to overcome the difficulties and time-consuming requirements in software development. This paper attempts to apply the framework to discrete event systems simulation and then to develop 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 runtime 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.

KEYWORDS

Component-based Modeling and Simulation, Models Reusability, Binary Reusable Modeling, Modeling in Multi-languages, DEVS/COM

1 INTRODUCTION

Constant innovation in computing technology made sophisticated applications available to users and such sophistication has brought with it many problems for the system designer. So a methodology is required to reduce the procedure of the development process by reusing existing module or purchasing ready-made sub-products that can easily be integrated into their own framework. The component-based software development framework, such as COM (Component Object Model), JavaBean and CORBA (Common Object Request Broker Architecture) is such a methodology to overcome that problem.

Discrete event simulation is a powerful tool for system design and the DEVS (Discrete Event System Specification) formalism proposed by Zeigler [2] 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 [3], which implements a visual modeling and simulation environment. The second approach is DEVS/CORBA execution environment [4] 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.

2 COMPONENT TECHNOLOGY AND SIMULATION MODELS REUSABILITY

2.1 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 [5]. 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: [6]

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

2.2 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 [7]. 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 make 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.

3 DEVS/COM ENVIRONMENT

A brief overview on the DEVS formalism is introduced in section 3.1 and architecture and implementation of the DEVS/COM environment will be followed. Some capability on the cross language model implementation is discussed on section 3.4.

[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'

3.1 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, 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 : $2^M - \phi \rightarrow$ M is a function which chooses one model when more than 2 models are scheduled simultaneously.

3.2 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.

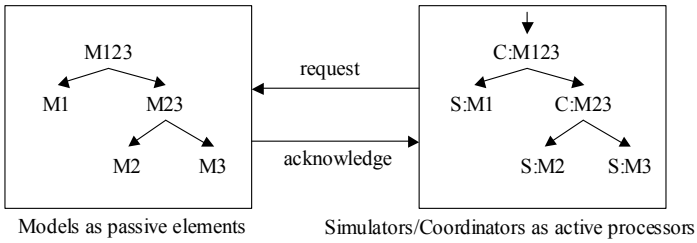

Figure 1: DEVS abstract simulator

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.

3.3 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.

Table 2: Component Simulator API for Atomic Model

Atomic Model	Function
X	`AddInport("Input");`
Y	`AddOutport("Output");`
S	`StateVar.put("State","Initial");`
δ_{ext}	`public void ExtTransFn(Variant stateVar, double timeE, Messages msg);`
δ_{int}	`public void IntTransFn(Variant stateVar)`
λ	`public void OutputFn(Variant stateVar, Messages msg)`
ta	`public timeType TimeAdvanceFn(Variant stateVar)`

Table 3: Component Simulators API for Coupled Model

Coupled Model	Function
X	`AddInport("Inport");`
Y	`AddOutport("Output");`
M	`StateVar.get("State");`
EIC	`AddCoupling(null, "sport", dst, "dport);`
EOC	`AddCoupling(src, "sport", null, "dport);`
IC	`AddCoupling(src, "sport", dst, "dport);`
SELECT	`GetPriority();` `SetPriority(int p);`

3.4 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.

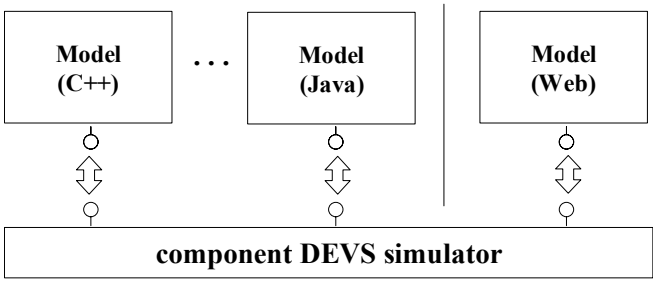

Figure 2: Cross Language Interoperability

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

.

4 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.

4.1 Example

This section presents a simple Banking example, which show how the component simulator works. The customer is generated from the EF (environment frame) to the Bank if the Bank is free and the system can be regarded as a single-size buffer queue system. The DEVS coupled model of the example is depicted as Figure3.

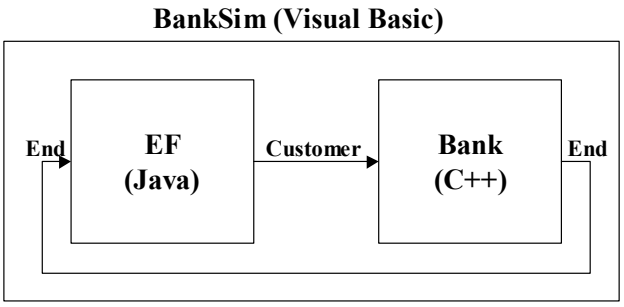

Figure 3: Banking System

Each of the coupled models is implemented as a Visual Basic, Java, and C++ programming language. Every coupled or atomic model is a binary COM component and each model can be implemented in any programming language that support COM runtime infrastructure. Figure 4, 5 and 6 prescribe simplified source code for coupled model of BankSim example.

```
Private Sub BankSim
  Dim pCM As New CoDevs.Coupled

  Dim M_EF = new EF
  …
 pCM.AddCoupling(M_BANK,"End",M_EF,"End");
  …
End Sub
```

Figure 4: Coupled Model BankSim in Visual Basic

```
import codevs.*;

public class EF
{
  Coupled pCM;
  EF(java.lang.String name)
  {
     // Initialize coupled model component
     // "new" does not only mean memory
     // allocation but component initialization.
     pCM = new Coupled();

     // Add input/output ports
     pCM.AddOutports("Customer");
     pCM.AddInports("End");

     // Define subordinate model
     Trans M_Trans = new Trans("Transducer");
     pCM.AddChild(M_Transducer);
     ...

     // Configure model couplings
     pCM.AddCoupling(M_Trans,"Stop",M_Gen,"Stop");
        ...
  }
}
```

Figure 5: Coupled Model EF in Java

```
class Bank : public CCmdTarget
{
  COMMON_INITIALIZE()

  // interface definition
  BEGIN_INTERFACE_PART(CBank, IBank)
   STDMETHOD(Bank) (BSTR);
  END_INTERFACE_PART(CBank)
  …
}

STDMETHODIMP Bank::XCBank::Bank(PBSTR pbs)
{
  // Initialize coupled model component
  USE_COMPONENT();

  pCM.AddInports("Customer");
  …

  // declare internal components
  USE_MODEL("Queue");
  pCM.AddChild(M_Queue);
  …

  pCM.AddCoupling(M_Bank,   "Customer",   M_Queue,
"customer");
  …
}
```

Figure 6: Coupled Model Bank in C++

4.2 Some discussion

By adopting the component technology to the DEVS simulation framework, reusing of the existing model component like various queue models may reduce the overall modeling time and binary component specification results in the true black-box encapsulation of the internal structure of the component and programming language independence. Component-based development framework is truly asynchronous that each of the model components can be developed independently in the large scale. But on the small-scale software, component-based approach does not show a specific strong point over the object-oriented approach in the development framework.

Besides the user model component, simulator is also a component and can be reused. On some case like web-based simulation, simulator component is re-used as a component of the whole system. Some large-scale software for imitating the real world system may re-use a simulator component and it is another point of the reusability – model reusability vs. simulator reusability.

5 CONCLUSION AND FUTURE WORK

In this paper we have 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. Secondly, various system architectures can be a ready-to-deploy component that a user may re-use it.

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. A quantitative measure on the performance degradation and the improved reusability is need to the systematic comparison between component-based methodology and object-oriented methodology.

Like other network-oriented component technologies, COM 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 efforts. The comparison between DEVS/CORBA and DEVS/HLA [8] (High Level Architecture) environment will be need on that time.

REFERENCE

[1] B. P. Zeigler, *Multifacetted Modeling and Discrete Event Simulation*. Academic Press. 1984

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

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

[4] 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
[5] Clemens Szyperski, *Component Software*, ACM Press, New York
[6] Tradeep Tapadiya, *COM+ Programming*, Prentice Hall PTR, Upper Saddle River, NJ 07458
[7] Ted J. Biggerstaff and Alan J. Perlis, *Software Reusability*, ACM Press, New York
[8] Zeigler, B.P. *Implementation of the DEVS Formalism over the HLA/RTI: Problems and Solutions*. In SIW. 1999. Orlando, FL.

AUTHOR BIOGRAPHY

YOUNG-IK CHO is currently a graduate student persuing received the M. S degree in the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea.

JAE-HYUN KIM received the B. S. and M. S. degrees in Electrical Engineering in 1998 and 2000, respectively, from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. His research interests include methodology for modeling and simulation of discrete event systems and mobile agent systems.

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 was a Full-time Instructor at Communication Engineering Department of Bukyung National University, Pusan, Korea between 1980 and 1983, and an Assistant Professor at Electrical and Computer Engineering at University of Kansas, Lawrence, Kansas, U.S.A. from 1989 to 1991. 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 the book *Theory of Modeling and Simulation* (2nd ed.), Academic Press, 2000. He is the Editor-in-Chief of *Transactions of Society for Computer Simulation* published by Society for Computer Simulation International (SCS). He is a senior member of IEEE and SCS and a member of ACM and Eta Kappa Nu.