DEVS-HLA: HETEROGENEOUS SIMULATION FRAMEWORK USING DEVS BUS IMPLEMENTED ON RTI

Yong Jae Kim, Jeong Hun Cho, and Tag Gon Kim

Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
373-1 Kusung-dong, Yusung-ku, Taejon, 305-701, KOREA
email: {yjkim,jhcho}@smslab.kaist.ac.kr, tkim@ee.kaist.ac.kr

KEYWORDS

Simulation Methods, Heterogeneous Simulation, DEVS BUS, HLA/RTI

ABSTRACT

The DEVS BUS is a virtual, software bus on which heterogeneous simulation models are interoperable based on the HLA concepts. The framework supports a common simulation infrastructure for communication between heterogeneous simulators by means of a set of well-defined interfaces between different simulation protocols. This paper first describes two HLA supporting simulation environments of DEVS models, namely DEVSim++-HLA and DEVSimJava-HLA. DEVSim++-HLA is an HLA extension of the DEVSim++ environment using RTI; DEVSimJava-HLA is that of the DEVSimJava environment using RTI. It then presents a DEVS-HLA methodology in which DEVS models developed in DEVSim++-HLA and those in DEVSimJava-HLA communicate with each other. Examples will demonstrate a power of the DEVS-HLA framework for heterogeneous simulation in a distributed computing environment.

1 INTRODUCTION

Previous research at Systems Modeling Simulation Lab at KAIST has developed object-oriented modeling/simulation environments for discrete event systems. Such environments include DEVSim++ and DEVSimJava which are a C++ and a Java realizations of the DEVS (Discrete Event Systems Specification) simulation methodology, respectively (Park and Kim, 1992; Cho and Kim, 1998). Each environment has been successfully applied to a variety of discrete event systems modeling/simulation studies including computer architectures, communication networks, communication protocols, software systems, manufacturing systems, war games, and so on.

As systems to be studied become heterogeneous and complex, however, simulation including heterogeneous models developed in different environments is inevitable. A main issue of such heterogeneous simulation is interoperability between different models or simulators. Recently, the HLA (High Level Architecture) has been developed to facilitate the interoperability between simulation models (Dahmann et al., 1998). The HLA has been successfully applied to various examples in military domain. Also, several works have been conducted to exploit the HLA as a simulation infrastructure outside military domain (Klein et al., 1998; Pham and Bagrodia, 1998; Kim and Kim, 1998).

As one of such works, the DEVS BUS has been proposed (Kim and Kim, 1998). The DEVS BUS is a common simulation infrastructure for heterogeneous simulation consisting of different simulation models and has been developed based on the DEVS formalism and the HLA. The DEVS BUS provides a simple interface so that a simulator could easily participate a heterogeneous simulation just by implementing the interface. This paper describes the DEVS-HLA framework in which existing DEVSim++ and DEVSimJava simulation environments are interoperable on the DEVS BUS implemented on the HLA/RTI.

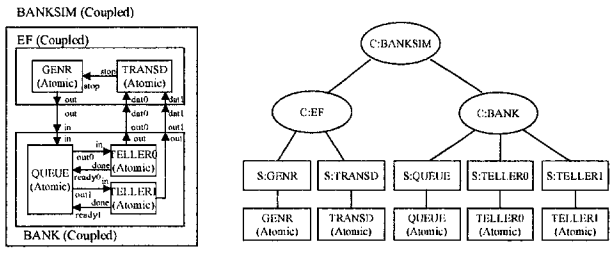
The rest of this paper is organized as follows. Section 2 describes the DEVS BUS including a brief review of DEVS formalism and an overview of the DEVS

BUS architecture. Section 3 presents several implementation issues of the DEVS-HLA framework and required modifications of existing DEVSim++ and DEVSimJava environments. Section 4 gives two examples for validating the framework. Finally, some conclusions are given.

2 DEVS BUS

2.1 DEVS Formalism

The DEVS formalism is a set-theoretic one to specify discrete event systems (Zeigler, 1984). The formalism includes two kinds of models, atomic and coupled. An atomic model specifies the dynamics of a system to be considered by using an extended timed finite state machine. On the other hand, a coupled model describes the structure of the system. The coupled model includes only connection information between its submodels, which may be atomic and/or coupled models.



(a) BANKSIM DEVS model

(b) Simulator structure of BANKSIM DEVS model

Fig. 1: DEVS model and hierarchical simulation algorithm for a banking system

Consider a banking system that has an waiting line and two tellers. When each customer enters the bank, she/he waits until all waiting customers arrived before her/him are served. When she/he is the first of the waiting line and a teller becomes available, she/he is served and leaves the bank. The overall simulation model, BANKSIM, is shown in Fig. 1 (a), where QUEUE is the waiting line and TELLER0 and TELLER1 are two tellers. To execute the model with various customer arrival rates, service times of tellers, and capacity of the waiting line, we design an experimental frame which consists of a customer generator, GENR, and a transducer of statistical information, TRANSD. GENR, TRANSD, QUEUE, TELLER0, and TELLER1 are atomic models and describe the behavior of each part of the banking system. To ease model development and management, the atomic models are grouped into two coupled models, EF and BANK, which are also combined into the bigger one, BANKSIM.

The hierarchical simulation algorithm for the coupled model, BANKSIM, is shown in Fig. 1 (b). Attached to each DEVS model is an associated abstract simulator, either a simulator for an atomic model or a coordinator for a coupled model. C:BANKSIM, C:EF, and C:BANK are coordinators for BANKSIM, EF, and BANK, respectively. Also, S:GENR, S:TRANSD, S:QUEUE, S:TELLER0, and S:TELLER1 are simulators associated with GENR, TRANSD, QUEUE, TELLER0, and TELLER1, respectively.

2.2 DEVS BUS Architecture

The DEVS BUS has been proposed to provide a common simulation infrastructure between different simulation models (Kim and Kim, 1998). When a simulator wants to join a heterogeneous simulation consisting of simulation models with different simulation world views, it comes true if the simulator just implements the DEVS BUS protocol. The protocol includes four kinds of messages, (x, t), (y, t), $(*, t)$, and $(done, tN)$. The former two are for data transfer and the latter two for bus control.

Fig. 2 shows the DEVS BUS architecture. The DEVS BUS controller consists of a scheduler and a deliverer. The scheduler is to control the bus so that a simulator can exclusively use the bus for a moment. The scheduler selects a simulator among several competing simulators so that the simulator exclusively use the DEVS BUS for an instant. The deliverer is to route simulation messages between different simulators and has all connection information, called a coupling scheme. The coupling scheme is a relation in which all pairs of source and destination models are specified. The deliverer receives data from source model and forwards it to destination model as specified in the coupling scheme. Specification of connection information in the coupling scheme, not in models, gives much flexibility in changing destination simulator.

The DEVS BUS protocol operates as follows. Initially, all simulators send its minimum next scheduling time to the scheduler. Among them, a simula-

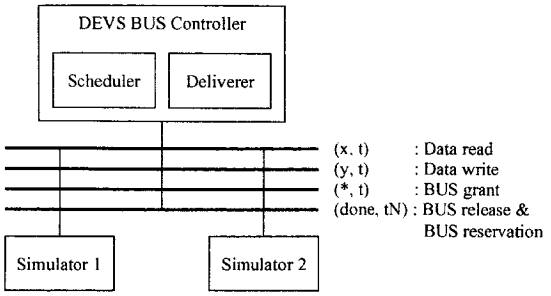


Fig. 2: DEVS BUS Architecture

tor with the minimum next scheduling time is selected by the scheduler, which sends $(*, t)$ to the simulator. Once receiving $(*, t)$, the simulator can use the bus and eventually sends $(done, tN_1)$ as a bus release/reservation to the deliverer. The bus reservation indicates that the simulator wants to be scheduled at the next event time tN_1. When the simulator produces (y, t), it sends (y, t) to the deliverer, which then translates (y, t) into (x, t) and forwards (x, t) to the destination simulator. Once receiving (x, t), the simulator executes it and sends $(done, tN_2)$ as its next minimum scheduling time to the deliverer.

3 IMPLEMENTATION ON THE HLA

The DEVS BUS has been implemented on the HLA (Kim and Kim, 1998). In this section, we first describe issues on time management when building the DEVS-HLA framework. And then, required modifications of the DEVSim++ (Park and Kim, 1992) and the DEVSimJava (Cho and Kim, 1998) environments are presented.

3.1 Time Management

The HLA provides four kinds of time management services according to two boolean factors, time-regulating and time-constrained (HLA Working Group, 1999). The logical time of a time-regulating (joined) federate is used to determine the logical time of time-constrained (joined) federates. Time-regulating federates may send TSO messages and time-constrained federates can receive TSO messages. This paper uses logical time synchronized service where all federates are both time-regulating and time-constrained. So, every federate participates in other federate's time advance decisions and accepts such participation from other federates.

The DEVS BUS protocol uses four kinds of messages, each of which is implemented as a reliable, TSO interaction. All federates publish and subscribe to the same set of interactions.

Basically the DEVS BUS protocol does not exploit lookahead, which results in an inherent problem especially for time-regulating federates. Each time-regulating federate has its lookahead value and only can send TSO messages that contain time stamps no less than its current logical time plus its lookahead value. In the DEVS BUS protocol, however, once receiving $(*, t)$, a simulator may want to send (y, t) to the deliverer at the same logical time t, which violates the lookahead restriction. So, whenever a simulator wants to send a TSO message, a predefined non-zero lookahead value should be added to the time stamp of the message.

3.2 DEVS BUS Controller

As mentioned before, the bus controller consists of the scheduler and the deliverer. The objective of the scheduler is to give the imminent simulator a grant to use the bus for an instant. When building the DEVS BUS on the HLA, however, there is no need to give a grant explicitly to the imminent simulator. In the situation where all federates are logically synchronized, such a grant is implicitly sent to the simulator by the `TimeAdvanceGrant` callback function of the RTI. If the grant time equals to the request time, the simulator can safely advance its simulation time to the grant time, which is an implicit $(*, t)$ of DEVS BUS in the DEVS-HLA framework.

Another main function of the scheduler is to synchronize the simulation time of participating federates at the initial setup phase. This function is required for late arriving federates. Without the scheduler, late arriving federates would start its simulation from false initial simulation time. In the worst case, a federate starting in a passive state may terminate immediately after starting simulation if it is the first federate of the federation without the scheduler.

The deliverer is to route simulation messages between simulators. Basically, it should not make any advance in simulation time. Because of the lookahead restriction, however, the deliverer makes simulation time advance with the amount of L. Note that object

class of the HLA/RTI also can be used to exchange TSO events, but it's not suited for message passing style information exchange in the DEVS-HLA framework.

3.3 DEVSim++-HLA

The hierarchical simulation algorithm of the DEVSim++ is for closed loop simulation. Every coordinator forwards $(*, t)$ received from its parent to the child that has the minimum next scheduling time. The highest coordinator, previously called the root-coordinator, updates the simulation time, generates a new $(*, t)$, and sends it to the imminent child.

In the DEVS-HLA framework, the simulation algorithm of the highest coordinator should be changed as shown in Fig. 3. If the grant time equals to the request time, the coordinator generates a new $(*, t)$ as a normal processing (line 8). When receiving an event from another federate, the grant time becomes less than the request time, which results in forwarding the event to destination model by sending (x, t) (lines 11-12). After processing events, the coordinator may send events to other federates via `SendInteraction`.

```
now := the logical time of the federate
1  while (simulation in progress) {
2      get next minimum scheduling time, t_N
3      invoke NextEventRequest(t_N)
4          receive events (via ReceiveInteraction)
5      honor RTI service request for TimeAdvanceGrant
6      now = time of grant
       ▷ check time
7      if (now == t_N) {
8          send (*, t_N) to the imminent child
9      }
10     else if (now < t_N) {
11         retrieve events, if any, with time stamp now
12         send (x, now) to destination model
13     }
14     else {
           ▷ this case should not occur
15     }
16 }
```

Fig. 3: Simulation loop of the highest coordinator

3.4 DEVSimJava-HLA

Existing simulation environments for discrete event systems suffer from the lack of portability, interoperability, and capability to execute over the Internet. The first two limitations may result in inflexible models; the last limitation prevents a modeler from a large scale compound simulation (Shen, 1998).

The DEVSimJava simulation environment, an implementation of the DEVS formalism using Java, has been developed to resolve these limitations (Cho and Kim, 1998). DEVSimJava is focused on remote model-base simulation, which exploits a Web technology for reusing simulation models available on the Internet. Within DEVSimJava, a modeler can use DEVS simulation models that exist elsewhere on the Internet by downloading from a remote site to the local site or by executing the models at the remote site using RMI (Remote Method Invocation). The environment also supports graphical capabilities, which enable a modeler to develop simulation models in intuitive manner.

The DEVSimJava-HLA is an extension of the DEVSimJava to exploit the HLA. The simulation algorithm of DEVSimJava also should be modified as shown in Fig. 3.

4 EXPERIMENTAL RESULTS

This section describes two examples, a banking system and a hypercube multiprocessor computer. Throughout the experimentation, the RTI 1.3v6 on Windows98/NT and Sun Solaris 2.6 environments with Visual C++ 5.0 and JDK 1.2 is used.

4.1 Banking System

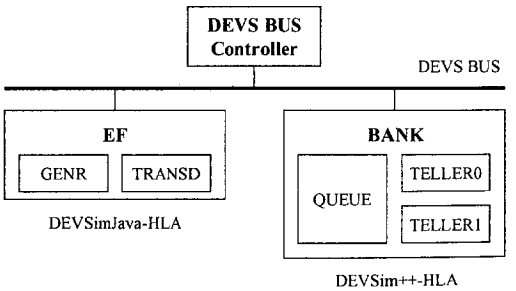


Fig. 4: Simulation of Banking System

Consider the banking system previously shown in Fig. 1. Simply, we partition the overall simula-

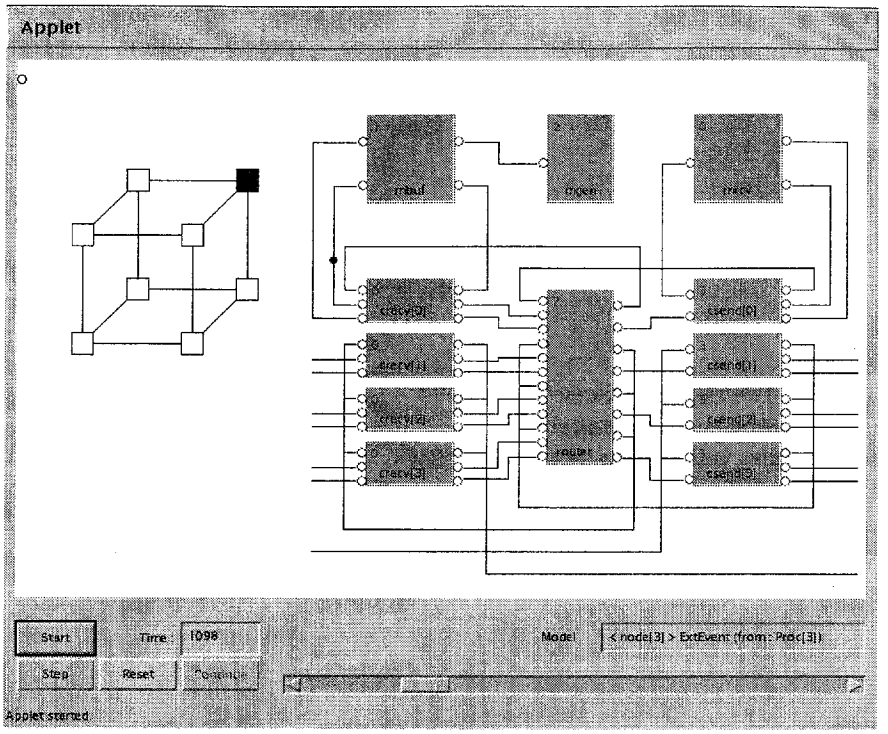


Fig. 5: Simulation of Hypercube Computer Model

tion model into two parts, EF and BANK, each of which is considered to be a federate. EF and BANK have been modeled in the DEVSimJava-HLA and the DEVSim++-HLA environment, respectively, and they are connected through the DEVS BUS as shown in Fig. 4.

The DEVS BUS controller is the first federate of the federation, BANKSIM, and waits without advancing its simulation time until two other federates, EF and BANK, join the federation. During the simulation, the controller forwards received messages to destination federate.

Execution speed is not so good. There are several reasons for slow execution speed. First, speedup is not the objective of incorporating the HLA. We lose better speedup while we get interoperability between different simulation environments. Second, time advance algorithm of the logical time synchronized service is basically conservative. So, speedup is very sensitive to lookahead value, but the DEVS-HLA framework inherently does not exploit lookahead. Finally, during the simulation, all federates exchange messages via the deliverer of the central controller. If the deliverer is distributed, more speedup can be achieved.

4.2 Hypercube Computer

This section considers a 3-dimensional hypercube multiprocessor system of (Ahn and Kim, 1993). Each node computer, NODE, consists of a processor, PROC, and a communication module, COMM. PROC includes a message source and a message sink. COMM consists of a router and 4 input and 4 output channels, in which one channel is for self routing and three for the outside of a node. Each message generated at PROC of a node is routed to one of neighboring nodes according to a routing policy. When a set of messages need to be routed through a single link at any instant, channel conflict may arise. For a real-time task, since messages to be transmitted is time-constrained, such conflict may result in missing deadline of the task. The purpose of this simulation is to find an optimal routing policy of time-constrained messages.

The federation includes the DEVS BUS controller and 8 federates, each of which executes a node computer model. Fig. 5 shows a federate executed in the DEVSimJava-HLA environment as an applet.

5 CONCLUSIONS

We have developed a simulation framework, DEVS-HLA, including the DEVSim++-HLA and the DEVSimJava-HLA, which are a C++ and Java realizations of DEVS formalism on HLA/RTI. DEVSim++-HLA and DEVSimJava-HLA environments communicate with each other via a virtual, software bus, the DEVS BUS.

We also have developed two examples, a banking system and a hypercube computer, to validate the DEVS-HLA framework. The results indicate that the framework provides interoperability between different simulation environments, but it does not give speedup. We believe that the DEVS-HLA framework gives much flexibility in modeling and simulation of discrete event systems.

REFERENCES

Ahn, M. S. and Kim, T. G. (1993). "DEVS Methodology for Evaluating Time-Constrained Message Routing Policies". *Discrete Event Dynamic Systems: Theory and Applications*, 3:173–192.

Cho, J. H. and Kim, T. G. (1998). "DEVSimJava: Web-based Environment for Simulation of DEVS Models". In *Proceedings of the 1998 Fall Conference of the Korea Society for Simulation*, pages 91–95. (in Korean).

Dahmann, J. S., Kuhl, F., and Weatherly, R. (1998). "Standards for Simulation: As Simple As Possible But Not Simpler The High Level Architecture For Simulation". *SIMULATION, The Journal of the Society for Computer Simulation International*, 71(6):378–387.

HLA Working Group (1999). *IEEE P1516.1/D4, Draft Standard for Modeling and Simulation(M&S) High Level Architecture(HLA) - Federate Interface Specification.* IEEE.

Kim, Y. J. and Kim, T. G. (1998). "A Heterogeneous Simulation Framework Based on the DEVS Formalism and the High Level Architecture". In *Proceedings of the 1998 Winter Simulation Conference*, pages 421–428.

Klein, U., Schulze, T., Straβburger, S., and Menzler, H.-P. (1998). "Traffic Simulation Based on the High Level Architecture". In *Proceedings of the 1998 Winter Simulation Conference*, pages 1095–1103.

Park, S. B. and Kim, T. G. (1992). "The DEVS Formalism: Hierarchical Modular Systems Specifications in C++". In *Proceedings of the 1992 European Simulation Multiconference.*

Pham, C. D. and Bagrodia, R. L. (1998). "Building Parallel Time-Constrained HLA Federates: A Case Study with the PARSEC Parallel Simulation Language". In *Proceedings of the 1998 Winter Simulation Conference*, pages 1555–1562.

Shen, C.-C. (1998). "Discrete-Event Simulation on the Internet and the Web". In *Proceedings of the 1998 SCS International Conference on Web-Based Modeling & Simulation.*

Zeigler, B. P. (1984). *Multifacetted Modelling and Discrete Event Simulation.* Academic Press Inc., London.