DEVS Framework and Toolkits for Simulators Interoperation Using HLA/RTI

Tag Gon Kim and Jae Hyun Kim

Systems Modeling and Simulation Lab., Department of EECS, KAIST

373-1 Yusong-gu, Daejeon, Republic of Korea

Abstract: The Discrete Event System Specification (DEVS) formalism is a framework which supports specification of discrete event models in a hierarchical, modular form. Associated with the formalism is a method which transforms a non-modular model to modular one. The method is a theoretical basis for the exchange of information between heterogeneous simulators which interoperate together. The High Level Architecture (HLA) is a specification of common services for such interoperation, which include synchronization of simulation time and exchange of data between interoperated simulators. This paper introduces the DEVS framework and associated tools which support development of interoperable simulators under the HLA/RTI environment. The core of the tools is the DEVSimHLA environment for development of HLA-compliant simulators using the DEVS framework and the RTI interface. An architecture of interoperable war game simulators based on DEVSimHLA is presented; a list of DMSO certified federates developed using the DEVSimHLA environment is given.

Keywords: DEVS formalism; DEVSim++; DEVSimHLA; Simulators Interoperation; HLA/ RTI

1. Introduction

A distributed simulation indicates that a collection of simulation processes collaborates to describe a behavior of a huge system. This methodology is primarily preferred when the model of a target system is so big that it does not fit into a single computing environment, or when the model has much parallelism so that we can obtain speedups in distributed computing environment.

The design of a particular distributed simulation follows generally a top-down approach. A big model is partitioned and mapped to available distributed processes. Protocols for communication and time synchronization among processes are pre-defined and applied to every simulation processes.

Simulators interoperation begins how to make existing simulators work together. It is the same as distributed simulation from the aspect of collaboration of distributed simulation processes. However, it is bottom-up approach that collects existing simulators and makes them cooperative by adding common services. In other words, simulators interoperation means the cooperation of heterogeneous simulators that are developed using different simulation algorithms, programming languages, and running platforms under supervision of different organizations.

In order to make heterogeneous simulators collaborate with each other, definition of standard protocols for data exchange and time synchronization among simulators are critical.

These common services are basic requirements for all distributed simulators. If these services are provided to developers, it will reduce development costs of new distributed simulators as well as that of adding interoperable features to pre-existing simulators. In addition, the common services increase reusability of simulators because the simulators already have common interfaces and easily become parts of other distributed simulation.

The High Level Architecture (HLA) [1-3] is a specification of common services required to distributed simulators. The main purpose of HLA is to provide highly interoperable and reusable simulator architecture for construction of distributed simulation applications.

To build a complete interoperable simulator, the models of target systems and its simulation algorithm are essential as well as HLA/RTI services. The HLA does not include any simulation algorithms or methodology. In this paper, Discrete Event System Specification (DEVS) formalism [4] is used to complete the design of whole interoperable simulator. The paper introduces a layered architecture that refines the relationship between interoperation specification, simulation algorithm, model and its object-oriented implementation into a layered architecture.

DEVSimHLA toolkits are designed to develop easily

the interoperable simulator with the layered architecture of DEVS framework. The usefulness of DEVSimHLA is demonstrated by lists of practically used federates certified by Defense Modeling Simulation Office (DMSO).

Section 2 briefly introduces the concepts of simulation interoperation using HLA/RTI. The DEVS framework and the layered architecture are described in Section 3. Section 4 presents DEVSimHLA environment. Some certified federates developed under DEVSimHLA environment and their typical architectures are presented in Section 5. Section 6 concludes the paper.

2. Simulators Interoperation Using HLA/RTI

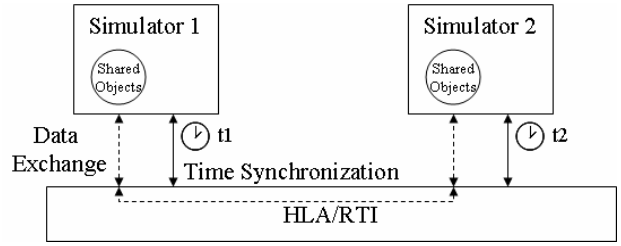

Figure 1 *Simulators Interoperation using HLA/RTI*

The HLA is a specification for interoperation among distributed heterogeneous simulations. The HLA specification is approved as IEEE std 1516 in 2000. The standard consists of two parts; Federate Interface Specification and Object Model Template (OMT).

Federate Interface Specification defines essential services of 6 categories. These services are also listed as a form of Application Programming Interfaces (APIs) written in various programming languages. The Run-Time Infrastructure (RTI) is an implementation of HLA Federate Interface Specification. The HLA only defines service APIs and lefts detailed algorithms to RTI software developers.

OMT is a documentation format for shared objects and message types by which simulators exchange events. OMT is used to describe Federation Object Model (FOM) and Simulation Object Model (SOM). The FOM is a collection of all shared objects and interactions defined inside whole distributed simulation, i.e. federation. The SOM is a list of shared objects and interactions that a simulator, i.e. federate, sends to or receives from other simulators.

Three major functionalities are necessary to accomplish simulation interoperation; time synchronization, data exchange, and simulation control/ management. Federate Interface Specification refines these functionalities into 6 management groups. Figure 1 shows the concepts of time synchronization and data exchange among simulators using HLA/RTI.

The HLA/RTI provides explicit time management services, while traditional parallel and distributed time synchronization algorithms manage the simulation time of a local process with only input events from other processes. The HLA/RTI has a list of time services that enable a federate to facilitate event-driven, time-stepped, or optimistic time synchronization methodology.

A federate is able to use two communication methodologies for exchanging data with other federate; shared objects and interactions. There exist, however, no physically shared objects inside a federation because each federate is a modular and distributed across the network. In a modular system, a component should notify the modification of its internal variable to be shared with other components whenever the variable is changed. To implement non-modular components in a modular system, in other hands, to emulate a shared object in distributed environment, the shared objects need to be declared, first, in the FOM and to be managed by lists of notification functions in the HLA.

Finally, the HLA supports control functions to organize a federation, miscellaneous functions for synchronization points, save and restoration, monitoring and management functionality by Management Object Model (MOM).

3. DEVS Framework and Co-Modeling Methodology

3.1 DEVS Formalism

The DEVS formalism specifies discrete event models in a hierarchical and modular form. With this formalism, one can perform modeling more easily by decomposing a large system into smaller component models with coupling specification between them. There are two kinds of models: atomic model and coupled model.

An atomic model is the basic model and has specifications for the dynamics of the model. Formally, a 7-tuple specifies an atomic model M as follows.

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

where

X: a set of input events;

Y: a set of output events;

S: a set of sequential states;

δ_{ext}: $Q \times X \rightarrow S$, an external transition function,

where $Q = \{(s,e) | s \in S, 0 \le e \le ta(s)\}$ is the total state set

of M;

δ_{int}: S → S, an internal transition function;

λ: S → Y, an output function;

ta: S → $R^{+}_{0,\infty}$ (non-negative real number), time advance function.

A coupled model provides the method of assembly of several atomic and/or coupled models to build complex systems hierarchically. Formally, a coupled model is defined as follows.

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

where

X: a set of input events;

Y: a set of output events;

M: a set of all component models;

EIC ⊆ DN.X × ∪M.X: external input coupling;

EOC ⊆ ∪M.Y × DN.Y: external output coupling;

IC ⊆ ∪M.Y × ∪M.X: internal coupling;

SELECT: 2^{M} – Ø → M: tie-breaking selector.

An overall system consists of a set of component models, either atomic or coupled, thus being in hierarchical structure. Each DEVS model, either atomic or coupled model, has correspondence to an object in a real-world system to be modeled. Within the DEVS framework, model design may be performed in a top-down fashion; model implementation in a bottom-up manner.

The DEVS framework provides a basis for modeling of discrete event systems on which logical analysis, performance evaluation, and implementation can be performed *based on only one framework*. To do so, a DEVS specification language is developed, which is a language realization of the DEVS formalism with sojourn times between states unspecified. Now a DEVS model specified by this language is used in the different phases for systems design and implementation by specifying additional information. To support the methodology a set of environments has been developed, which include [5] for performance analysis, [6] for real-time simulation, [7] for virtual prototyping, [8] for specification verification, [9] for hybrid systems analysis and others.

3.2 Layered architecture and Co-Modeling

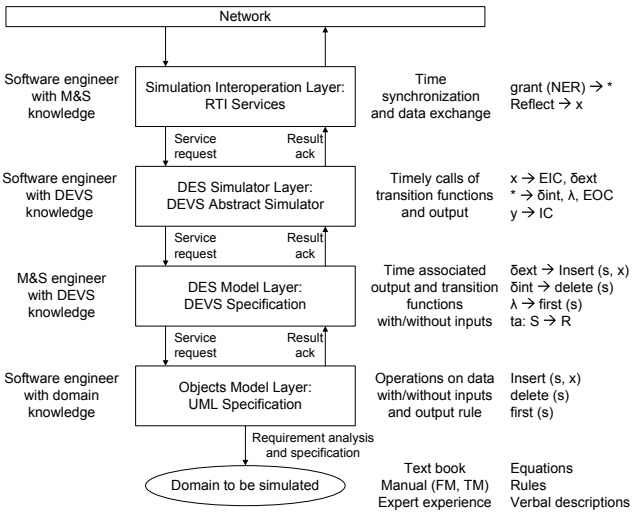

Figure 2 *Layered Architecture of Object-Oriented DEVS Modeling [10]*

Recently, layered architecture and DEVS-UML co-modeling methodology has been proposed to partition specifications of a discrete event models in two expert groups: domain experts of a target system and modeling and simulation experts [10]. To build a successful model, two groups of experts should cooperate with each other.

As shown in Figure 2, domain knowledge for the system to be modeled is represented by mathematical equations, rules or even verbal descriptions which are available in text books, field and/or technical manuals. Extraction of appropriate equations and rules from the domain knowledge which meet modeling objective can be done by domain experts. To be successful they should clearly identify modeling objectives. This is not the case for modeling of hardware and/or processes for which M&S experts without domain-specific knowledge can understand.

The approach represents a discrete event model in two layers: a discrete event system model (DES) layer and an object model (OM) layer. More specifically, the OM layer specifies details of object operations on variables of an object; the DES layer specifies abstract operations on states of the same object which determine simulation time advance and output events generation.

Above previous layers, there exist DES simulator layer and simulation interoperation layer. DES simulator layer defines simulation algorithms for underlying DES models. The simulation algorithms are clearly separated with DES models. This

separation of layers reduces development cost of a whole simulator because the verified simulation algorithms are invariant and, therefore, reusable irrespective of underlying models.

The Simulation interoperation layer is required only when the simulator should be a part of distributed simulation. A stand-alone simulator that consists of lower three layers easily turns out to be interoperable by simply adding the simulation interoperation layer. This layer consists of two parts – HLA/RTI for external interoperation, and internal protocol conversion for underlying DES simulation layer. The HLA/RTI is good for interoperation with other simulators. The HLA/RTI is, however, not enough for this layer because it does not contain any specification for simulation algorithms. The layer requires a specific protocol conversion from HLA/RTI to underlying DES simulator layer.

4. Architecture and Toolkits for War Game Models Development within DEVS Framework

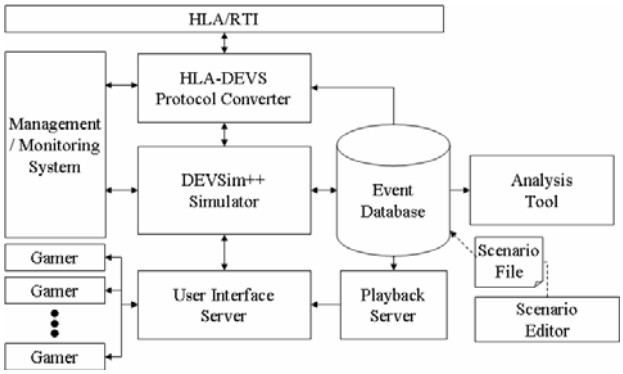

Figure 3 *Architecture for Interoperable War Game Models Development using DEVSimHLA Environment*

DEVSimHLA is a simulation toolkit for federate developers. DEVSimHLA is designed to reduce efforts and costs for developing a federate with layered architecture. To support the simulation interoperation layer, DEVSimHLA provides HLA-DEVS Protocol Converter library and a supportive application, FOM-to-Class Generator. For the DES simulator layer, DEVSim++ library is developed. Therefore, federate developers are able to focus only on DEVS models.

An example of application architecture using DEVSimHLA is demonstrated in Figure 3. A main simulator working with a lot of surrounding components is developed using DEVSim++ library. Along with the HLA-DEVS Protocol Converter, the simulator becomes interoperable through HLA/RTI.

4.1 DEVSim++

DEVSim++ is an implementation of abstract simulation algorithm of DEVS. DEVSim++ is provided as a software library to model developers, and together with DEVS models (DES layer and OM layer), it forms a stand-alone simulator.

The classical DEVS formalism deals with systems of closed form and static structure. There have been many researches to extend DEVS formalism to express variable structure, real-time and interactive systems. DEVSim++ supports these features as well. DEVSim++ has an additional event queue to handle events from interactive components such as Graphical User Interfaces, HLA-DEVS protocol converter, and other monitor/controllers. These interactive components can be connected through networking or shared memory technology.

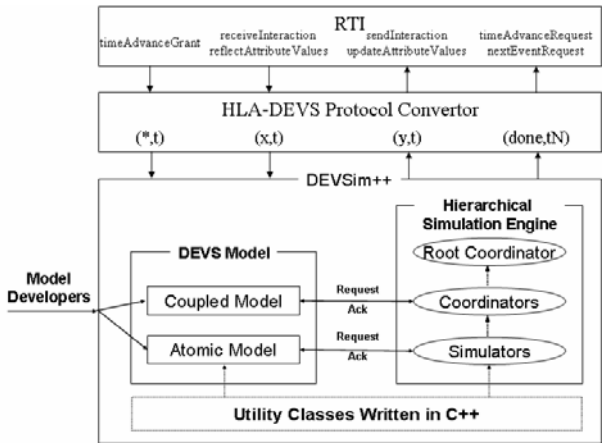

Figure 4 *DEVSim++ and HLA-DEVS Protocol Converter*

4.2 HLA-DEVS Protocol Converter

The HLA-DEVS protocol converter is a program for simulation interoperation layer. The converter includes Local RTI Component (LRC), converter library, application-specific data conversion routines. The LRC is a part of RTI software package, and provides RTI services to the federate. DEVSimHLA provides HLA-DEVS protocol converter library, which implements the protocol conversion algorithms.

Figure 4 shows the protocol conversion between DEVS abstract simulation algorithm and HLA/RTI.

The time requesting services such as *timeAdvanceRequest* and *nextEventRequest* are initiated when the protocol converter receives (done,tN) message

inidicating the internal schedule of DEVSim++ simulator are done and its next schedule time is tN. The time request is granted by RTI using its callback function – *timeAdvanceGrant*. Then, the protocol converter notifies to the simulator that it is ok to proceed to the granted time by sending (*,t). Services related to exchange data – *send/receiveInteraction*, *update/reflectAttributeValues* – can be mapped to (x,t) or (y,t) representing input and output events to/from models. The developers need to develop this application-specific data conversion routine. Generally, data to be shared with other simulators – defined in a FOM – does not exactly match with internal data structures. Sometimes, the internal data structure is much more complicated than data in the FOM that some entities should be abstracted out or be assumed to have default values.

4.3 FOM-to-Class Generator

To reduce the efforts and costs to develop the converter program, DEVSimHLA provides a converter library and FOM-to-Class Generator. Algorithms for converting services are essential and reusable to all applications, therefore they are provided as a form of library. However, data to be exchanged among simulators, i.e. FOM, are application-specific. FOM-to-Class Generator is designed to produce C++ classes that contain FOM data and provide access functions, combined with the converter library. FOM-to-Class Generator has two inputs - FDD and a template code file. The generator extracts object, attribute, interaction and parameter information from FDD and generates codes using a template code file. By this way, version updates in both FOM and the protocol converter library would be reflected without modification of FOM-to-Class Generator itself.

5. Case Study

5.1 Certified Federates Developed using DEVSimHLA Environment

Defense Modeling and Simulation Office (DMSO) provides HLA Federate Compliance Testing service. This testing is to verify, with the checklist, that a simulator is HLA-compliant. The checklist was developed as part of the HLA, along with HLA test procedures. Currently both checklists for two versions of HLA specification – DMSO v1.3 and IEEE 1516 are available.

DMSO publishes a list of compliant federates in their website that passed the HLA Federate Compliant Testing [11]. Table 1 shows the list of certified federates developed using DEVSimHLA toolkits. The applications include war game models to be used for military training operation, and simulation of defense systems.

Table 1 *Certified HLA-Compliant Federates Developed using DEVSimHLA Environment*

Federate Name	Spec. Version	Pass Date	Organization
ChangGong v1.0	1.3	06/09/2005	ROK AFOC
AFECS AECS v1.0	1.3	09/03/2004	ROK ADD
AFECS HES v1.0	1.3	09/03/2004	ROK ADD
AFECS OCS v1.0	1.3	07/01/2004	ROK ADD
Cheong-hae, v1.0	1.3	07/19/2003	ROK NSMD

5.2 War Game Model

This section introduces the generic architecture of a typical war game model using DEVSimHLA (see Figure 3). The DEVSim++ simulator in the middle of the architecture includes the target war game DEVS models. The simulator has interactive and real-time features in order to communicate with surrounding components. The HLA-DEVS Protocol Converter enables the whole simulator to interoperate with other federates. The FOM-to-Class Generator, which is not described in the picture, reduces development efforts of the HLA-DEVS Protocol Converter by generating C++ classes from the FOM. The protocol converter is required only when the war game model participate to the federation for joint exercises.

In a joint military training simulation, robustness is the most important characteristics required to the simulator. The separation of the protocol converter from main simulator gives rise to higher robustness to the whole system. Along with save and restoration features, the local simulator will recover from unexpected crash causing no effect to the whole federation.

The Event Database plays a key role in save and restoration, and analysis of results. The database receives and saves events produced from the simulator. The saved events are used to restore simulator at particular time, to analysis the results of simulation and to replay by the playback server.

During the training operation, a lot of gamers take part in the training. Each gamer has different level of

accessibility to platforms. The User Interface Server routes simulation events from/to gamers.

6. Conclusion

This paper presents DEVSimHLA toolkits that implement the layered architecture of an interoperable simulator with DEVS framework. The usefulness of DEVSimHLA toolkits is demonstrated by lists of certified federates of various military applications. A generic architecture for interoperable war game models using DEVSimHLA environment is also discussed.

Using DEVSimHLA, developers need less efforts and costs to develop HLA-compliant federates of practical size. The layered architecture of the simulator increased robustness of the total system, because the loosely coupled protocol converter prevents fault propagation from local simulator to the whole federation.

References

[1] IEEE, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules, Std 1516, 2000.

[2] IEEE, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Federate Interface Specification, Std 1516.1, 2000.

[3] IEEE, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Object Model Template (OMT), Std 1516.2, 2000.

[4] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim, *Theory of Modeling and Simulation*, Academic Press, 2000.

[5] T. G. Kim, DEVSim++ User's Manual, SMSLab, Dept. of EECS, KAIST, 1994, http://smslab.kaist.ac.kr

[6] S. M. Cho and T. G. Kim, Real Time Simulation Framework for RT-DEVS Models, *in Transactions of The Society for Modeling and Simulation International*, vol. 18, no. 4, 2001, pp. 178–190.

[7] J. S. Hong, H. S. Song, T. G. Kim, and K. H. Park, A Real-time Discrete Event System Specification Formalism for Seamless Real-time Software Development, *in Discrete Event Dynamic Systems*, vol. 7, no. 4, 1997, pp. 355–375.

[8] G. P. Hong and T. G. Kim, A Framework for verifying discrete event models within a DEVS-based system development methodology, *in Transaction of the Society for Computer Simulation International*, vol. 13, no. 1, 1996, pp. 19–34.

[9] B. P. Zeigler, H. S. Song, T. G. Kim and H. Praehofer, DEVS Framework for Modelling, Simulation, Analysis, and Design of Hybrid Systems, *in Hybrid Systems, LNCS 999*. Springer Verlag, 1995, pp. 529–555.

[10] Chang Ho Sung, Su-Youn Hong and Tag Gon Kim, Layered Approach to Development of OO War Game Models Using DEVS Framework, *Proceedings of the 2005 Summer Computer Simulation Conference*, July 2005

[11] http://hlatest.msiac.dmso.mil/compliant_feds.html

Author(s) Biography

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, Busan, Korea between 1980 and 1983, and an Assistant Professor at Electrical and Computer Engineering Department at University of Kansas, Lawrence, Kansas, U.S.A. from 1989 to 1991. He joined at Electrical Engineering Department of KAIST, Daejeon, 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 is a senior member of IEEE and SCS and a member of ACM and Eta Kappa Nu. He was the Editor-in-Chief of *SIMULATION: Transactions of Society for Modeling and Simulation International* for the methodology section.

Jae Hyun Kim received a B.S. and M.S. in electrical engineering in 1998 and 2000, respectively, from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. Currently, he is a Ph D. student at the Department of Electrical Engineering and Computer Science, KAIST. His research interests include methodology for modeling and simulation of discrete event systems and high level architecture.