Design and Implementation of Simulators Interoperation Layer for DEVS Simulators

Jae-Hyun Kim, Su-Youn Hong and Tag Gon Kim
Department of EECS
KAIST
373-1 Kusong-dong, Yusong-gu
Daejeon, Korea 305-701
Tel: +82-42-869-3454
Fax: +82-42-869-8054
{jhkim,syhong}@smslab.kaist.ac.kr, tkim@ee.kaist.ac.kr

Keywords: Simulators Interoperation Layer, DEVS Simulator, HLA, RTI

A simulator for a discrete event system consists of following four layers – Simulation Interoperation, Discrete Event System (DES) Simulator, DES Model, and Objects Model Layers. Among these layers, the Simulation Interoperation Layer exists on top of the DES Simulator Layer and is responsible to communicate with other simulators based on standards for distributed simulation such as IEEE 1516 High Level Architecture. We select Discrete Event System Specification (DEVS) formalism as the mathematical background for the rest three layers. The paper proposes how to design and implement HLA based Simulation Interoperation Layer for DEVS simulators. The Simulation Interoperation Layer manages mapping between HLA services and DES simulation messages used in DEVS abstract simulation algorithm. The Korean Navy war game simulator called CHUNG-HAE has been developed with the proposed architecture, and has successfully joined the confederation with US legacy models.

1 INTRODUCTION

A simulation interoperation indicates a distributed simulation that puts emphasis on collaboration of heterogeneous simulators. Especially, many stand-alone simulators that have been developed under various authorities start to support the simulation interoperation standard, i.e., High Level Architecture (HLA) [1-3]. For example, there are separate military training simulators for army, navy, air force, and marine. These simulators become interoperable with each other by supporting HLA standard. This interoperation makes possible to generate more huge and precise battlefields.

We have been developing interoperable war game simulator for several years. This experience made us gain an insight into general architecture of interoperable simulators. Some previous work [4][5] proposed four layered architecture of an interoperable simulator. (See Figure 1)

The Simulator Interoperation Layer exists on top of the DES Simulator Layer and is responsible to communicate with other simulators. The DES Simulator Layer deals with simulation algorithms in order to simulate models which are located in the DES Model Layer. The Objects Model Layer provides primitive classes and functions to DES Model Layer, of which DES models are composed.

In this paper, we choose HLA as a communication mean for Simulation Interoperation Layer, and Discrete Event System Specification (DEVS) [6] as a modeling and simulation methodology for other layers. Therefore, the DES Simulator Layer implements DEVS abstract simulation algorithm, and the DES Model Layer encompasses DEVS models. Without the Simulation Interoperation Layer, the remains of three layers build a stand-alone DEVS simulator.

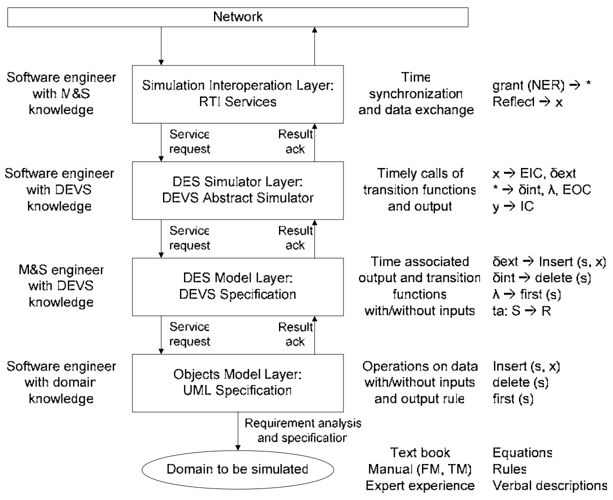

Network
Software engineer with M&S knowledge
Simulation Interoperation Layer: RTI Services
Time synchronization and data exchange
grant (NER) → *
Reflect → x
Service request
Result ack
Software engineer with DEVS knowledge
DES Simulator Layer: DEVS Abstract Simulator
Timely calls of transition functions and output
x → EIC, δext
* → δint, λ, EOC
y → IC
Service request
Result ack
M&S engineer with DEVS knowledge
DES Model Layer: DEVS Specification
Time associated output and transition functions with/without inputs
δext → Insert (s, x)
δint → delete (s)
λ → first (s)
ta: S → R
Service request
Result ack
Software engineer with domain knowledge
Objects Model Layer: UML Specification
Operations on data with/without inputs and output rule
Insert (s, x)
delete (s)
first (s)
Requirement analysis and specification
Domain to be simulated
Text book
Manual (FM, TM)
Expert experience
Equations
Rules
Verbal descriptions

Figure 1 Four Layered Architecture of an Interoperable Simulator

This paper focuses on how to design and implement HLA based Simulation Interoperation Layer for DEVS simulators. The Simulation Interoperation Layer manages mapping between HLA services and DEVS simulation messages for DEVS abstract simulation algorithm.

The Simulation Interoperation layer can be implemented either within one simulator process or as a separate process. In this paper, we assume that the Simulation Interoperation Layer is implemented as a single process which is connected to the DEVS simulator via TCP/IP.

There are several reasons for this separation. Firstly, the Simulation Interoperation Layer is optional. The layer is only required when the simulator joins a federation. When the simulator works alone, the Simulation Interoperation Layer may affect the performance. Secondly, this separation increases robustness of the federation. When either the Simulation Interoperation Layer or the DEVS simulator fails by unexpected faults, the other layer may help recovering and resuming failed process. Finally, it is sometimes not possible to find Run-Time Infrastructure (RTI) library that supports platforms of the DEVS simulator. For example, there is no 64-bit RTI library.

The paper is organized as follows: Section 2 briefly reviews DEVS formalism. The design of Simulation Interoperation Layer is described in Section 3. The implementation of the layer, KHLAAdaptor Library, is introduced in Section 4. Section 5 shows an example simulator, CHUNG-HAE. Finally Section 6 concludes the paper.

2 DEVS FORMALISM: BRIEF REVIEW

2.1 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 $Q = \{(s,e) | s \in S, 0 \leq e \leq ta(s)\}$ is the total state set of M;

δ_{int}: $S \rightarrow S$, an internal transition function;

λ: $S \rightarrow Y$, an output function;

ta: $S \rightarrow R^{+}_{0,\infty}$ (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;

$EIC \subseteq DN.X \times \cup M.X$: external input coupling;

$EOC \subseteq \cup M.Y \times DN.Y$: external output coupling;

$IC \subseteq \cup M.Y \times \cup M.X$: internal coupling;

SELECT: $2^M - \emptyset \rightarrow M$: 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.

2.2 DEVS Abstract Simulator

The DEVS Abstract Simulator automatically generates the same hierarchy of simulation agents with DEVS models. The simulation agents exchange simulation messages in Table 1 along the hierarchy in order to run simulation. The simulator agent for an atomic model AM is called a *simulator* and is denoted as S:AM. The simulator agent for a coupled model CM is called a *coordinator* and is denoted as C:CM.

Table 1 DEVS Simulation Message

Message	Meaning
(*,t)	Internally generated event at time t that notifies the scheduled time is completely elapsed
(x,t)	External input event at time t
(done, t_N)	Synchronization event generated at time t_N that notifies the next scheduled time is t_N

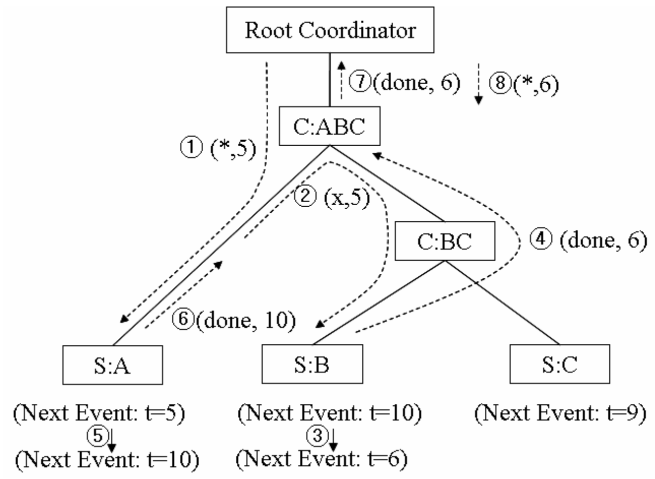


Figure 2 Example Cycle of DEVS Simulation Algorithm

The detailed algorithm will not be presented, but an example in Figure 2 will help those who are not familiar with DEVS algorithm. Let us assume that there is a coupled model ABC. The ABC is composed of an atomic model A and a coupled model BC, which is divided into two atomic model B and C. Let us assume that the initial time of A, B, C is 5, 10, 9, respectively. When the simulation starts, the Root Coordinator knows that there are no events scheduled between time 0 and 5. Therefore, it sets current time as 5, and generates * message with time stamp 5. The (*,5) is delivered to S:A, and A:λ is called according to its algorithm. Let us assume that A:λ produces an output event (x,5) which is delivered to S:B according to coupling information in model ABC and BC. S:B handles the events by calling B: δ_{ext} and B:ta. B:ta calculates time of next event and S:B reports it by sending (done,6) message. After completing output messages of A, S:A calls A: δ_{int} and A:ta. Then S:A reports its next schedule by (done, 10). Finally, C:ABC reports (done, 6) to Root Coordinator since 6 is the smallest time among all models. The Root Coordinator now knows that there are no events between time 5 and 6, and it advances simulation time from 5 to 6. The Root Coordinator repeats previous sequence by sending (*,6) to C:ABC.

The DEVS simulation algorithm is implemented within the DES Simulator Layer. If the Simulation Interoperation Layer is connected to the DES Simulator Layer, the Root Coordinator does not advance simulation time when it receives (done, tN) message. Instead, it forwards to the Simulation Interoperation Layer and waits until it gets (done, tN) message from the Simulation Interoperation Layer.

3 SIMULATION INTEROPERATION LAYER

The main function of the Simulation Interoperation Layer is to provide communication interface between HLA and the DEVS simulator which is to be interoperated with other simulators through HLA services. Technically, the layer establishes a mapping between HLA services implemented in RTI and DEVS simulation messages implemented in DEVS abstract simulation algorithm. More specifically, the services can be divided into three major categories: time management, data management, and federation management. Time management deals with time synchronization of DEVS models with other models to be interoperated through HLA services. It also supports real-time simulation under some constraints. Data management converts objects and interactions in an arbitrary Federation Object Model (FOM) defined in HLA from/to DEVS input/output events. Federation management provides synchronization points, save and restore functions.

3.1 Time Management

Simulators use two types of time request services – Time Advance Request (TAR) and Next Event Request (NER). The HLA supports the optimistic time advance service, but it will not be considered in this paper. The Simulation Interoperation Layer needs to convert two mechanisms to DEVS simulation messages in order to synchronize time between RTI and the DES Simulator Layer.

3.1.1 TAR ↔ DEVS

Some federations have policies that any participating federate should use TAR with specified time interval. The TAR mode in the Simulation Interoperation Layer requests a time advance to the time t':

t' = current time + pre-defined time step.

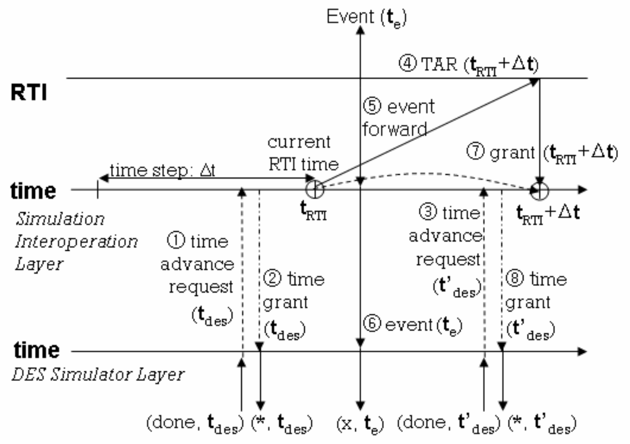


Figure 3 TAR ↔ DEVS

Figure 3 shows how to synchronize simulation time between RTI and the DES Simulator Layer. When the Root Coordinator in the DES Simulator Layer receives (done, tN) from its model, it converts (done,tN) to the time advance request message and forwards it to the Simulation Interoperation Layer. If tN is less than currently granted RTI time, the Simulation Interoperation Layer instantly grants its time advance and the message is converted to (*, tN) in the Root Coordinator. If tN is greater than current RTI time, the Simulation Interoperation Layer invokes TAR with next time step. If there exist TSO events during TAR requests, the events are converted to (x,t) and delivered to DEVS models. After RTI grants its time advance request, the Simulation Interoperation Layer generates a time grant message with the same time stamp of (done, tN), not granted RTI time. The DES Simulator Layer always receives a time grant message which is equal to the time it requests.

3.1.2 NER ↔ DEVS

The NER mode in the Simulation Interoperation Layer invokes NER with time that the DES Simulator Layer has requested. The RTI grants the time of the first next event if there is an event less than requested time. If there is no event, the RTI grants the requested time.

If there is an event, the Simulation Interoperation Layer delivers it to the DES Simulator Layer. Then it is converted to (x,t) and delivered to the appropriate model along the coupling scheme. The delivery of (x,t) causes (done, tN) message as a response. Therefore the simulator is able to continue time advance.

If there is no event, the Simulator Interoperation Layer delivers time grant message of requested time to the DES Simulator Layer. The message is converted to (*, t) as usual. The procedure is illustrated in Figure 4.

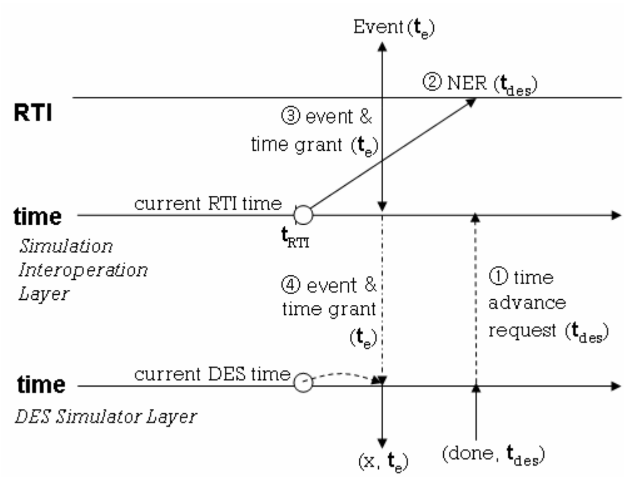
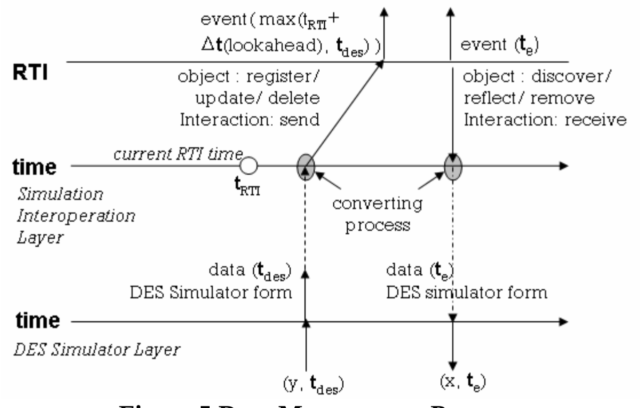


Figure 4 NER ↔ DEVS

3.2 Data Management

The RTI supports declaration, object and data distribution management for data exchange. Declaration management includes publication, subscription and supporting control functions. Federates must declare exactly what they are able to publish or subscribe. The object management includes instance registrations and instance updates on the object production side and instance discoveries and reflections on the object consumer side. The object management also includes methods associated with sending and receiving interactions, controlling instance updates based on consumer demand, and other miscellaneous support functions [7].



event(max(t_{RTI}+ Δt(lookahead), t_{des}))
event (t_e)
RTI
object : register/ update/ delete
Interaction: send
object : discover/ reflect/ remove
Interaction: receive
time
current RTI time
t_{RTI}
Simulation Interoperation Layer
converting process
data (t_{des}) DES Simulator form
data (t_e) DES simulator form
time
DES Simulator Layer
(y, t_{des})
(x, t_e)

Figure 5 Data Management Process

Figure 5 shows how the Simulation Interoperation Layer converts events between RTI and the DES Simulator Layer. Let us assume that a DEVS model generates an output event that is marked to be delivered to RTI. The output event is denoted as (y,t) in Figure 5. The DES Simulator Layer forwards the output event to the Simulation Interoperation Layer. The Simulation Interoperation Layer has the mapping information between the output event and corresponding RTI objects or interactions. After conversion is made, the Simulation Interoperation Layer calls series of proper RTI services in order to send the event. For example, if the event is mapped to an object update, the Simulation Interoperation Layer will call the *updateAttributeValues* service followed by the *registerObjectInstance* service. If the event is mapped to an interaction, the Simulation Interoperation Layer will invoke the *sendInteraction* service.

When the Simulation Interoperation Layer converts the event, it also checks the time stamp of the event. The new time value t’ for the event will be the following:

t’ = max(current RTI time + lookahead, original event time)

A federate is prohibited to generate a TSO event with time less than current RTI time + lookahead if the federate enabled time regulation. If this is the case, the event should be delayed. This case happens especially if the federate uses TAR services.

3.3 Federation Management

Federation management includes such tasks as creating federations, joining federates to federations observing federation-wide synchronization points, effecting federation -wide saves and restores, resigning federates from federations and destroying federations[7].

These services are usually not considered in DEVS simulators. Therefore the Simulation Interoperation Layer needs to provide an easy way for users to use these services.

In our implementation, we provide a control application that is able to connect to the application of the Simulation Interoperation Layer. Users can give orders to the federate by clicking buttons in order to perform federation management services.

4 IMPLEMENTATION: KHLAADAPTOR LIBRARY

KHLAAdaptor library is the implementation of core library for the Simulation Interoperation Layer. Using KHLAAdaptor library, developers are easily able to build the Simulation Interoperation Layer for a simulator. It offers time management, data management, and federation management and offers the abstract callback message handler for developers.

Figure 6 shows the architecture of KHLAadaptor library. CMsgHandler classes process RTI callback functions. External messages from the DES Simulator Layer can be handled by adding them to message queues defined in KHLAAdaptor library. Message Handlers and message queues are explained in detail in the following sections.

4.1 Message Handlers and Message Queues

There are several message handlers installed inside the library. These message handlers are for user-defined handling of RTI callback functions. These message handlers are categorized by 6 RTI management services and contain callback functions. For example, there is a class CFederateMgtHandler as a federation message handler. CFederateMgtHandler class has member functions such as *announceSynchronizationPoint.* Users can define their own callback functions.

There are three message queues – time message queue, Receive Order (RO) message queue, and Time Stamp Order (TSO) message queue.

The time message queue handles a time advance message. When the DES Simulator Layer sends a time advance request message, the Simulation Interoperation Layer adds this message into the time message queue, and KHLAAdaptor requests the time advance using the message in the time message queue.

The RO message queue handles RTI callback messages and external messages. When RTI callback function is called, CFederateAmbassador class makes suitable messages and adds this message to RO message queue. So, the class inherited from CFedereateMgtHandler can process theses messages.

The TSO message queue handles RTI callback messages with time stamp and external messages with time stamp.

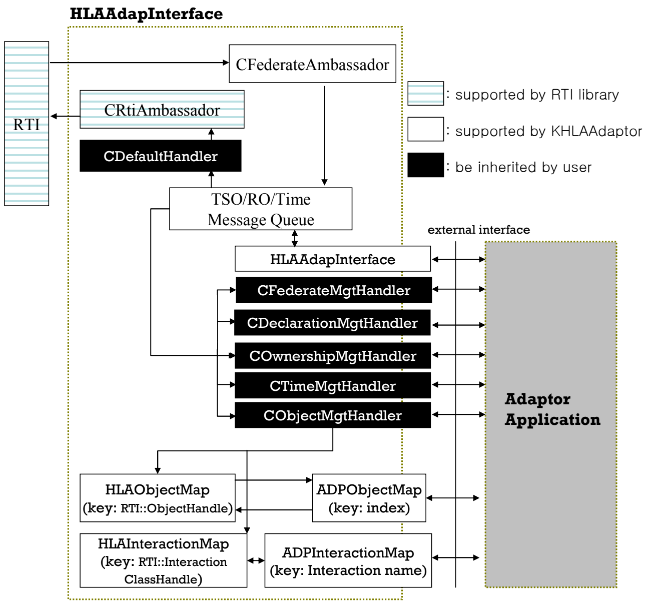

HLAAdapInterface
RTI
CFederateAmbassador
CRtiAmbassador
CDefaultHandler
TSO/RO/Time Message Queue
HLAAdapInterface
CFederateMgtHandler
CDeclarationMgtHandler
COwnershipMgtHandler
CTimeMgtHandler
CObjectMgtHandler
HLAObjectMap (key: RTI::ObjectHandle)
ADPObjectMap (key: index)
HLAInteractionMap (key: RTI::Interaction ClassHandle)
ADPInteractionMap (key: Interaction name)
▤ : supported by RTI library
□ : supported by KHLAAdaptor
■ : be inherited by user
external interface
Adaptor Application

Figure 6 The Architecture of KHLAAdaptor Library

4.2 Time Management

The Simulation Interoperation Layer receives a time request message from the DES Simulator Layer in the external simulator. There are two time management modes: TAR and NER as explained in Section 3.

Figure 7 shows the sequence diagram for TAR mode. KHLAAdaptor requests a time advance by using the front message of the time message queue. The Simulation Interoperation Layer invokes TAR service to RTI according to the message.

Figure 8 describes the sequence diagram for NER mode.

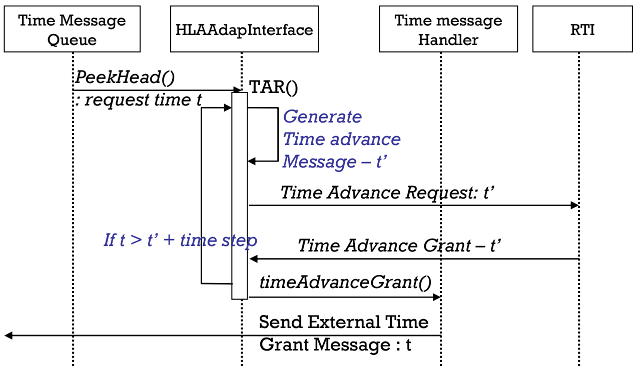


Figure 7 Sequence Diagram for TAR mode

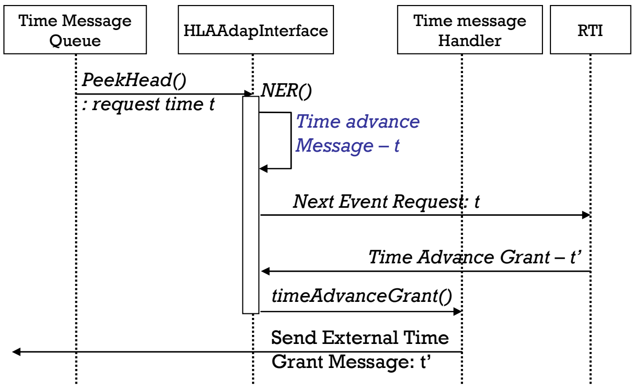


Figure 8 Sequence Diagram for NER mode

4.3 Data Management

Generally, the data format used in the simulator can be different from that defined in the FOM. The KHLAAdaptor library provides schemes for automatic conversion between both sides of data. However, developers need to implement actual conversion functions used in the library. The library automatically calls provided functions whenever the events happen.

The mapping between HLA objects/interactions and application-specific data is stored inside the library. KHLAAdaptor manages objects and interactions registered on RTI with maps. HLAObjectMap in Figure 9 manages Objects registered on RTI using RTI Object Handle and ADPObjectMap saves corresponding application-specific data.

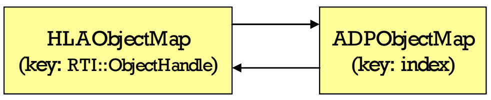

HLAObjectMap (key: RTI::ObjectHandle)
ADPObjectMap (key: index)

Figure 9 Object Map

There exists a HLA class generator that generates source codes for the HLA objects defined in the FOM [5].

ADP objects mean the objects which are created by developers as counter parts of HLA objects. HLAObject and ADPObject have pointers indicating each other. KHLAAdaptor creates them as a pair and saves in maps.

KHLAAdaptor library also deals with interactions in the same way.

On the object production side, the Simulation Interoperation Layer calls register and/or update APIs when it receives the requests from the DES Simulator Layer. The Simulation Interoperation Layer accomplishes object discoveries, reflections on the object consumption side by callback functions of RTI, too. The discovered object is added into HLA/ ADP object map illustrated in Figure 9 automatically, and KHLAAdaptor calls *decodeHLAObject* functions whenever the object is reflected. Figure 10 shows the sequences for the discovery and reflection of objects. The italic letters means that this function is processed automatically by KHLAAdaptor library.

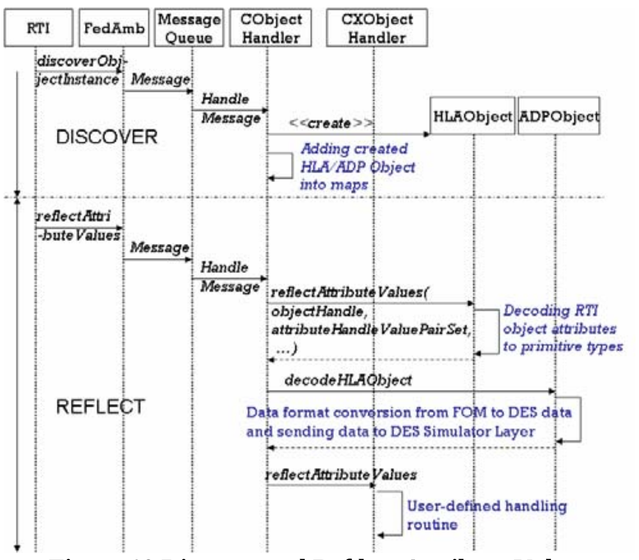

RTI
FedAmb
Message Queue
CObject Handler
CXObject Handler
discoverObjectInstance
Message
Handle Message
<<create>>
HLAObject
ADPObject
DISCOVER
Adding created HLA/ADP Object into maps
reflectAttri-bute Values
Message
Handle Message
reflectAttributeValues(objectHandle, attributeHandleValuePairSet, ...)
Decoding RTI object attributes to primitive types
decodeHLAObject
REFLECT
Data format conversion from FOM to DES data and sending data to DES Simulator Layer
reflectAttributeValues
User-defined handling routine

Figure 10 Discover and Ref lect Attribute Values

5 Example: CHUNG-HAE

The layered structure of the simulator is applied to development of war game simulators for Korean Navy, Air Force, and Marine. The Korean Navy simulator, called CHUNG-HAE, is in the testing stage, while others just started development. Figure 11 shows the architecture of CHUNG-HAE. CHConverter is the implementation of the Simulation Interoperation Layer. The management/ monitoring system provides graphical interfaces for simulation control and federation management of the federate.

While CHUNG-HAE is under development, it has successfully joined the confederation with US legacy models. Since 2004, CHUNG-HAE has passed the ROK-US Confederation Test (CT), Functional Test (FT) which is held under the auspices of the Korean Combined Battle Simulation Center successfully. CHUNG-HAE joined Ulchi Focus Lens – the Korean CPX simulation, and verified its stability and utility, too.

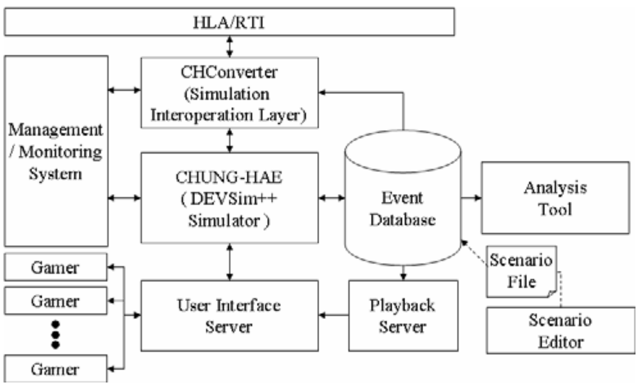


Figure 11 CHUNG-HAE

6 Conclusion

The paper presents the design and implementation of the Simulation Interoperation Layer for the DEVS simulators. The Simulation Interoperation Layer supports conversion between two time management methods - TAR and NER and DEVS abstract simulation algorithm. The Layer provides automatic transformation of HLA object/ interactions into DEVS events and vice versa. The core functionality of the Simulation Interoperation Layer is implemented as a library form called KHLAAdaptor. Using this library, developers can easily build the Simulation Interoperation Layer for a specific DEVS simulator.

7 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] 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 Summer Computer simulation Conference*, 2005

[5] Tag Gon Kim and Jae Hyun Kim, "DEVS Framework and Toolkits for Simulators Interoperation Using HLA/RTI," *Proceedings of Asia Simulation Conference/the 6th International Conference on System Simulation and Scientific Computing*, pp. 16 - 21, Oct 24-27, 2005, Invited Paper.

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

[7] DMSO. High Level Architecture Run-Time Infrastructure RTI 1.3-Next Generation Programmer’s Guide Version 5, 1999