RESEARCH-ARTICLE

DEVS based plug-in framework for interoperability of simulators

JANG WON BAE, Korea Advanced Institute of Science and Technology, Daejeon, South Korea

TAG GON KIM, Korea Advanced Institute of Science and Technology, Daejeon, South Korea

Published: 11 April 2010

SpringSim '10: 2010 Spring Simulation Conference
April 11 - 15, 2010
Florida, Orlando

DEVS Based Plug-in Framework For Interoperability of Simulators

Jang Won Bae, Tag Gon Kim
Department of Electrical Engineering & Computer Science,
Division of Electrical Engineering
Korea Advanced Institute of Science of Technology
Daejeon, KOREA
E-mail : jwbae@smslab.kaist.ac.kr, tkim@ee.kaist.ac.kr

Keywords: DEVS, Plug-in, Interoperation, Framework, Environment

Abstract

For many years, designers of complex systems have recognized the need for interoperation among different system models. Each of models is proper to represent structures or behaviors of the target system, such as discrete event system and continuous system. Furthermore, interoperation can provide users with reusability of models and distributed simulation environment. This paper proposes a framework, called PlugSim, which provides not only distributed simulation environment but also interoperation among simulators. The PlugSim utilizes plug-in method so that users can simulate a target system with models developed in accordance with the simple interface of the PlugSim. The PlugSim provides two kinds of interfaces: DEVS model interface and Non-DEVS model interface. With plug-in method, users can synthesize an interoperable simulator with the framework and their models. Besides, users can change user models during the simulation. For a distributed and interoperation environment, the PlugSim is separated from user models and has algorithms to support data exchange and time synchronization. At the end of the paper, to show the correctness of the PlugSim, the result of an experiment about interoperation with DEVS model and MATLAB model, an example of Non-DEVS Model will be shown.

1. INTRODUCTION

Modeling methodology in the perspective of discrete event system has considered as a representative method of abstract knowledge representation about a real world. The Discrete Event Systems Specification (DEVS) formalism, a set-theoretic formalism, specifies discrete event systems in a hierarchical and modular form [1]. For Modeling and Simulation of the discrete event systems, the DEVS formalism has been widely used in many parts. For many years, designers of complex systems have recognized the need for interoperation among different system models. Each of models is proper to represent structures or behaviors of the target system, such as discrete event system and continuous system model [2]. Furthermore, Interoperation among different models can provide users with reusability and distributed simulation environment [3] [4]. There have been many researches about interoperation. A representative example is HLA/RTI. HLA (High Level Architecture) is a standard in IEEE and it describes a specification for interoperation between heterogeneous simulators. RTI (Run-Time Infrastructure) is software that is the resultant of implementing HLA [5] [6] [7]. HLA/RTI is supporting interoperability and reusability of simulators. However, it is inconvenient to interoperate among simulators using HLA/RTI. In order to use HLA/RTI, users should understand the APIs and callback functions of HLA/RTI and make interfaces that can handle HLA/RTI services. Another example is the DEVS Bus [8]. The DEVS Bus, based on HLA/RTI and DEVS formalism, is the interface that can communicate among heterogeneous simulators over the RTI. It also needs users to develop the model interface that is compatible to the HLA/RTI. There is a recent research about distributed simulation. Virtual Laboratory Environment (VLE) is the framework for integration of heterogeneous formalisms using DEVS and associated extensions [9].

This paper proposes a framework, called PlugSim, which provides distributed and interoperable simulation environment. The PlugSim utilizes plug-in method so that users can simulate a target system with models developed in accordance with the simple interface of the PlugSim [10]. In the view of users work, the PlugSim differs from DEVS Bus and HLA/RTI. In the DEVS Bus and HLA/RTI, user should implement the HLA-compatible interface but in the PlugSim, user implements DEVS atomic model or simple interface. Difference with VLE is that the PlugSim supports exchanging models during the simulation so that users can compare models in the same simulation.

The objective of the PlugSim is conveniently creating an interoperable simulator that is synthesizing the provided framework and user models. Using plug-in method, users can change their models during the simulation. The PlugSim

offers two types of interface: DEVS model interface and Non-DEVS model interface. DEVS model interface is based on the atomic model specification in the DEVS formalism [1]. Non-DEVS model interface consists of network handling process using TCP/IP connection, which can provide users with the distributed simulation environment [14]. The PlugSim has algorithms for data exchange and time synchronization that are indispensable for interoperation.

The rest of the paper is organized as following: In section2, DEVS formalism, on which the PlugSim is based, is introduced. Section 3 presents an overview and internal components in the PlugSim. Section 4 shows how to make models to simulate in the PlugSim. Section 5 gives an experimental result of interoperation simulation between DEVS model and MATLAB model. Section 6 concludes the paper.

2. DEVS FORMALISM

This section briefly explains the DEVS formalism. The DEVS Formalism specifies discrete event models in a hierarchical and modular form. The DEVS formalism exhibits the concepts of system theory and modeling. With this formalism, anyone can do modeling with ease by decomposing a large system into many smaller components with coupling scheme among them. There are two kinds of models in the formalism: Atomic model and Coupled model [1]

Atomic model is a specification of basic model behavior as timed state transition. 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;
Y : a set of output events;
S : a set of sequential states;
$\delta_{ext} : Q \times X \rightarrow S$, an external transition function,
where $Q = \{(s,e)|s \in S, 0 \le e \le ta(s)\}$ is the total state set of M;
$\delta_{int} : S \rightarrow S$, an internal transition function;
$\lambda : S \rightarrow Y$, an output function;
$ta : S \rightarrow$ Real, time advance function.

Coupled model is a specification of hierarchical model structure. It provides the method of assembly of atomic and/or coupled models to build complex systems hierarchy. Formally, a Coupled model is defined as follows:

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

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: $2^M - \phi \rightarrow M$: tie-breaking selector.

An overall system consists of a set of component models, atomic or coupled models. Therefore, the system can be a hierarchical and modular form.

3. PLUGSIM : THE FRAMEWORK FOR INTEROPABILIY OF SIMULATORS

This Section introduces the PlugSim. Conceptual design, advantages and how the PlugSim supports interoperation between simulators will be introduced in detail.

3.1. Conceptual Design of the PlugSim

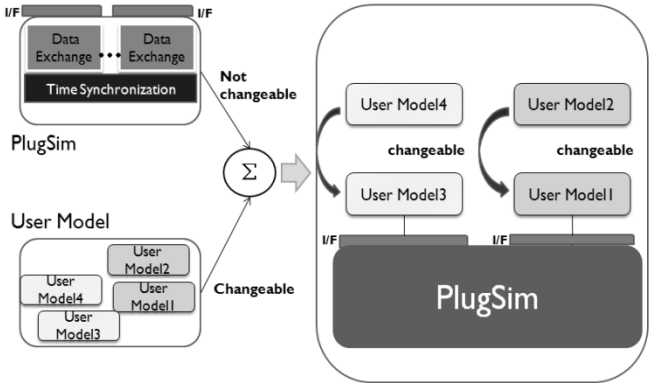

Figure 1. Conceptual Design of the PlugSim

The objective of the PlugSim is to provide distributed and interoperation environment to users and to change models during the simulation. Generally, in the simulation, there are two parts: changeable part and unchangeable part. The changeable part is such as a simulation model. It can be modified for the purpose of the simulation. The un changeable part is like data exchanging, time synchroniza tion and simulation algorithm, which are essential for the interoperation of simulation. Figure 1 shows the conceptual design of the PlugSim. The conceptual design shows that the PlugSim is providing an unchangeable part to users. Therefore, the PlugSim helps to make interoperable environments with user models as a changeable part.

Users can simulate user models in the interoperable environments by developing the models complying with the provided interfaces. The models are available to change with other models using plug-in method. It means that a model can change the internal structure, such as algorithms or behaviors, during the simulation. However, the data used

in the model cannot change. This can be helpful to simulations in which the structure of model (such as algorithms, not the data used in the model) changes frequently according to the objectives of simulation. For example, in the missile simulation, user can change the algorithm of missile model. However, input/output data of torpedo model cannot be changed. With this alternative, user can evaluate the effectiveness of the missile in one simulation.

User models participating in the simulation are connected each other through the PlugSim. It supports interoperability among simulation models with data exchanging and time synchronization algorithms.

3.2. Overview of the PlugSim

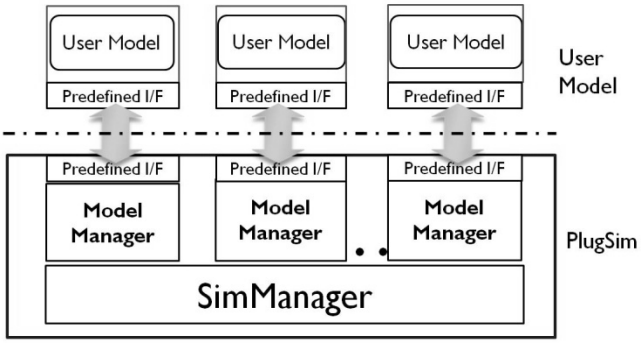

Figure 2. Overview of the PlugSim

Figure 2 shows an overview of the PlugSim. There are two kinds of internal components in the PlugSim: a SimManager, a number of ModelManagers. User model, which is a simulation model developed by user, can be connected the PlugSim with the provided interfaces. The role of internal components, SimManager and Model Manager, in the PlugSim is supporting interoperation. Their roles are like as following:

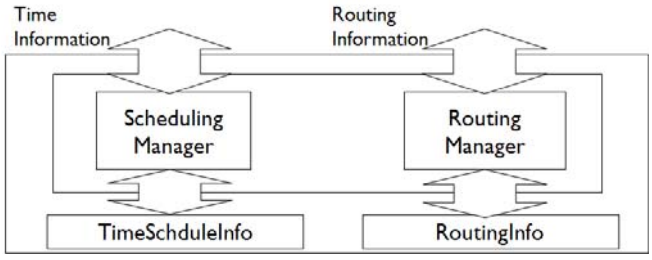

Figure 3. Structure of SimManager

SimManager: SimManager has two components: Scheduling manager and Routing manager. Scheduling manager manages time schedule in the simulation using the time information from ModelManager. Routing Manager manages data exchange information from user documents. SimManager connects to all Model managers joined in the simulation and control them to execute the simulation. Figure 3 shows the structure of SimManager.

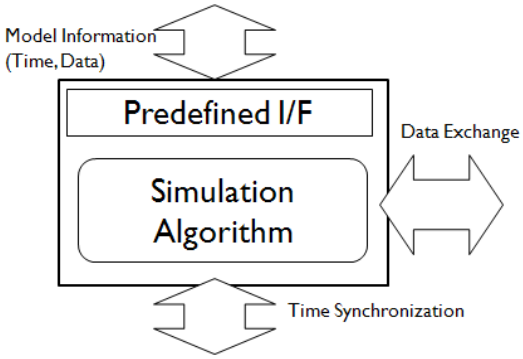

Figure 4. Structure of ModelManager

ModelManager: one ModelManager is connected for supporting one of user models. The ModelManager uses simulation algorithm based on the DEVS formalism to simulate the connected model. The ModelManager gets time and data information from the model. The ModelManager is connected to SimManager and other ModelManagers. So, The ModelManager sends data information to other ModelManager according to RoutingInfo and time information to SimManager. Figure 4 shows the structure of ModelManager.

3.3. Data Exchange in the PlugSim

The PlugSim supports data exchanging among models. It can be performed by RoutingInfo that is managed by SimManager. All ModelManagers get RoutingInfo from SimManager before the simulation. ModelManagers transfer data to other ModelManagers according to the RoutingInfo. In the following subsection, how RoutingInfo is made and detail procedure of data exchange will be explained.

3.3.1. RoutingInfo

RoutingInfo is the table containing information about which model generates what kind of data and which model receives the data. It is generated by documents that are representing input/output data of models. User should prepare this information, called Interoperation Object Model (IOM), before execution of simulation. There are two kinds of the documents in the IOM: Local Data Set (LDS) and Global Data Set (GDS). LDS is the set of data that is input/output in a model. GDS is the set of data used in the simulation. Generally, GDS is the union of LDS of all

simulation models. Using a simple example, how the RoutingInfo is constructed will be shown.

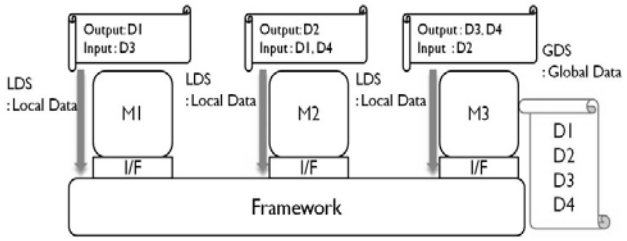

Figure 5. RoutingInfo using LDS and GDS

In Figure 5, we assume that three user models have joined the simulation using the PlugSim. In order to use the PlugSim, users should make LDSs, as many as the number of models, and one GDS. For example, In Figure 5, three models joined in the PlugSim, so three LDSs and one GDS are required. When models join, the SimManager reads LDSs and a GDS and make a RoutingInfo like Table 1.

Table 1. Example of RoutingInfo

Data Name	Source Model	Destination Model
D1	M1	M2
D2	M2	M3
D3	M3	M1
D4	M3	M2

Using the RoutingInfo table the PlugSim can recognize the data transferring information.

3.3.2. Data Exchange

In this subsection, how data exchange is carried out using RoutingInfo will be explained in detail. In the DEVS formalism, data is exchanged using the coupling relations between atomic models in a coupled model. With the coupling scheme in the coupled model of the DEVS formalism, we compare the data exchange using RoutingInfo in the PlugSim.

In Figure 6, a coupled model is represented using the RoutingInfo in the PlugSim. With Figure 6, we explained how data exchanging is performed. For example, in the coupled model, out1, one of the data used in the system, is generated in the M1 and forwards to M2 through Coupled model M23 by the coupling scheme. This data exchanging can execute in the PlugSim. In the PlugSim, D1 generated from M1goes to M2 directly according to RoutingInfo. Consequently, the data exchange in the coupled model and in the PlugSim is in the bi-simulation relation. However, in the point of the number of message, the PlugSim is more efficient than the DEVS Coupled model [11].

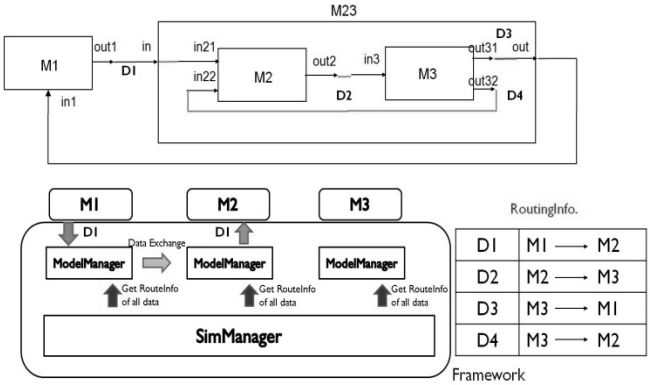

Figure 6. Comparing coupling scheme in DEVS

3.4. Time Synchronization in the PlugSim

```
SimManager
  Inputs :
    TAR(tn)  message.
  Outputs :
    TAG(*, tn) message
  variables :
    tn   // Time of Next event
    m   // ModelManger
    Tni // Next event time of ith model
Intiallization :
     Update  all TAR message(Tn) of ModelManagers
When all TAR message (tn) received
    tn = min { Tn1, Tn2, Tn3, .. ,Tnk}
    if (tn  equal to 'infinity')
        End simulation
    else
    send  TAG(*,tn) message to ModelManager m of tn
```

Figure 7. Algorithm of Time Synchronization

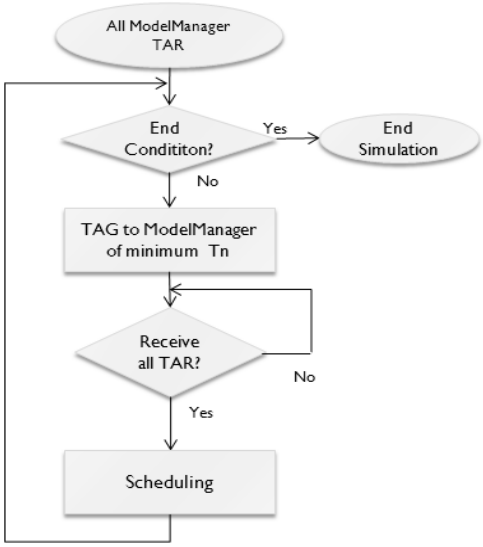

Figure 8. Flowchart of Time Synchronization

Time Synchronization in the PlugSim has performed using the next event time of joined models. Figure 7 shows the algorithm of time synchronization in the PlugSim and Figure 8 shows the flowchart of the time synchronization.

When the simulation has begun, each of the joined models informs the connected ModelManager of next event time of the model. The informed ModelManager sends TimedAdva nceRequest (TAR) to SimManager with the next event time. After SimManager gets TARs of all ModelManagers, the SimManager schedule the next event time of the whole simulation. In other words, the SimManager sends Time AdvanceGrant (TAG) to the ModelManager sent the minimal next event time among all ModelManagers [12]. The TimeAdvanceGranted ModelManager informs the model that the model can be executed until the granted time. After execution, the model request the next event time to the ModelManager. This process will be iterated until the end condition of the simulation. End condition of the simulation is when the minimum next event time is 'infinity'. It means the whole simulation has already performed.

4. USER MODEL DEVELOPMENT

The PlugSim supports two types of models: DEVS model and Non-DEVS model. In order to use the PlugSim, user should do extra works: making LDS and GDS file and implementing models with provided interfaces. Especially, According to the type of user model, user should develop model with a proper interface. In the following subsection, using DEVS model and MATLAB model that is a representative of Non-DEVS model, how to make LDSs and GDS and implement with the provided interfaces is represented case by the model type.

4.1. Making LDS and GDS

LDS and GDS are used to make RoutingInfo in the SimManager. LDS is the set of input/output in a model, so the number of LDSs is same as the number of models. GDS is the set of using data in the whole simulation. User should make LDSs and GDS as the form provided in the PlugSim.

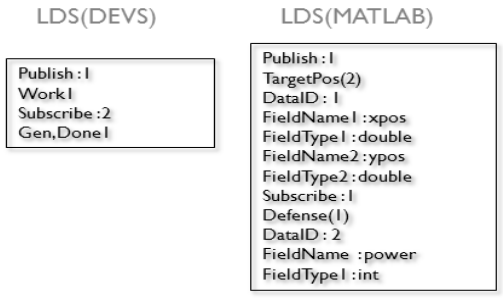

Figure 9. Example of LDSs

Figure 9 shows the examples of LDS according to a type of model. In DEVS model case, users should specify data names, the number of data according to published and subscribed data, respectively. In MATLAB model case, users should describe more information of exchanged data: data names, the number of data, packet ID, the number of field, the name and type of each field according to published and subscribed data. This information uses for making packets to communicate between MATLAB model and Model Manager in the PlugSim. GDS is made with the union of all LDSs of models.

4.2. User Interfaces

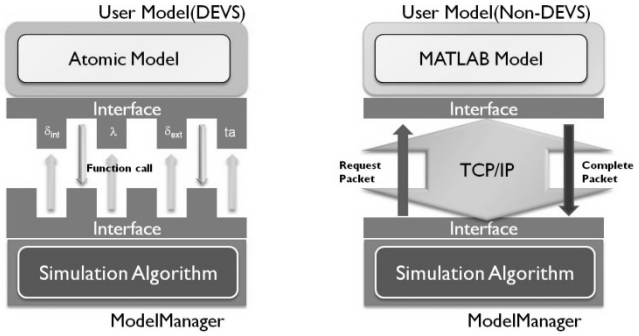

Figure 10. Interfaces in the PlugSim

Figure 10 shows two kinds of provided interfaces in the PlugSim. User models connect to the PlugSim through the provided interfaces. DEVS Model interfaces in the ModelManager connect to user model by four functions (internal, external transition, output, time advance function) in the DEVS atomic model. In other words, DEVS Model Interface enables the ModelManager to call state transition, time advance and output function of DEVS model directly. Therefore, ModelManager can get time and data information of DEVS model through DEVS model interface. Using DEVS formalism as an interface of the PlugSim, DEVS models can be reused in the PlugSim [13]. User implements DEVS model with the interfaces and make it as a DLL (Dynamic Link Library) file. With this DLL file, DEVS model can change during the simulation using Dynamic DLL Loading method. In MATLAB model, the PlugSim can communicate with it using TCP/IP interface [14]. MATLAB Model interfaces convert MATLAB model time information to TAR and TAG messages. The PlugSim automatically generates packets for communicating with MATLAB model according to LDS of the MATLAB model. With these packets, the ModelManager gets Time and Data information.

5. CASE STUDY

The PlugSim applied to simulation of simple war game models depicted in Figure 11.

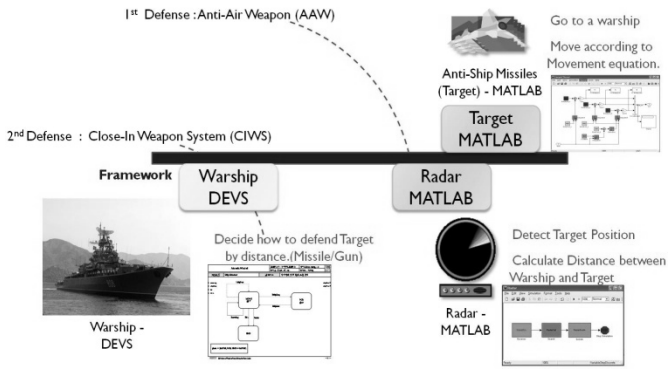

Figure 11. Defense of warship against anti- ship missiles

The objective of this simulation is to evaluate the survivability of a war ship. This simulator can be useful to increase the survivability by testing various parameters of defense systems in a warship. The brief scenario of the war game simulation is as follows.

1: A warship is moving to a certain destination point.

2: Radar is detecting any hostile missiles toward the warship

3: A warship has two steps of defense systems: Anti-Air Weapon (AAW) and Close-In Weapon System (CIWS). The systems have different effective ranges and so they are controlled by command and control (C2) system.

4: If an Anti-Ship Missile (ASM) hits the warship or it is shot down, the simulation ends.

The example simulation model consists of three models. The models and their roles are as follows.

- Warship Model
  It sends initial variables and states to other models. This model takes roles of maneuver and engagement analysis. The warship model sends Warship Position to Radar Model in some period. When it receives a distance from Radar Model, it assigns a defense system by the range of systems. This model is modeled by the DEVS formalism, because it is activated by state transition with some events.

- ASM Model
  It calculates next position using following movement equations by period and sends the position information to Radar model.

$$x = V_x t + x_o$$
$$y = V_z t + y_0$$
$$z = \int_0^t (V_z + gt)dt + z_0 \, , g = -9.8 m/s^2$$

- Radar Model
  It detects positions of a warship and an ASM periodically and sends the information to the warship. Mathematical functions of MATLAB/Simulink are used to decide whether the ASM is in the range of detection.

Overall simulation architecture of the war game model is described in Figure 11. Models consist of one DEVS model and two MATLAB models. All models are implemented with the provided interfaces and LDS and GDS about the simulation is prepared. The purpose of the simulation is to analyze survivability of a warship according to detection range of a radar model. The simulation is executed with parameters in Table 2.an ASM moves toward the warship using movement equations. Detecting range changes at an interval of 3km from 10km to 25km. Figure 12 shows the simulation result of the war game model. As the longer detection range of radar, the probability of a warship's survival gets higher.

Table 2. Parameters using Simulation

Parameter	Value
Total simulation count	100
# of warships	1
Initial position of warships $(x,y,z)km$	(0,0,0)
# of targets	1
Initial position of targets $(x,y,z)km$	(20,20,0)
# of AAWs	6
# of CIWSs	100
Range of CIWSs (km)	≤ 5

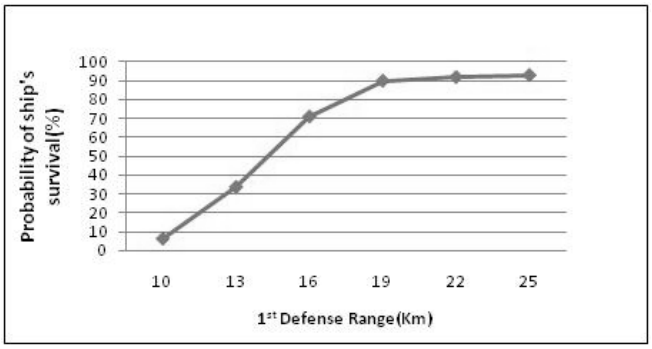

Figure 12. Result of Simulation

The paper that simulates the same models in a different environment, HLA/RTI environment, exists [14]. The result of [14] shows same trend as this simulation.

6. CONCLUSION

This paper proposes the framework, called PlugSim, which supports interoperation among simulation models and provides users with a distributed environment. Because of using plug-in method in the PlugSim, users can change their models during the simulation without any changes in the other models. Changing models during the simulation is an advantage to simulations for developing internal algorithms of model. To support interoperation, the PlugSim has algorithms for data exchange and time synchronization. In order to use the PlugSim, users only define the shared data in the simulation and implement their models with the provided interfaces.

As a further work, we think automatically generating IOM using standard methods and extending user model to a Coupled model that supports hierarchical and modular design.

ACKNOWLEDGEMENT

This work was supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract UD080042AD, Korea.

REFERENCES

[1] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim, *Theory of Modeling and Simulation.* ACADEMIC PRESS, 2001.

[2] Ören T.I. and Zeigler B.P., "Concepts for Advanced Simulation Methodologies," *Simulation, vol. 32, no. 3*, pp. 69-82, 1979.

[3] Basili V., L. Briand and W. Melo, "How Reuse Influences Productivity in Object-Oriented Systems," *Communications of the ACM, Vol.39, No.10*, pp104-116. 1996.

[4] Ören T.I., " Future of Modeling and Simulation: LDSe Development Areas," *Proceedings of the 2002 Summer Computer Simulation Conference*, pp. 3-8, 2002.

[5] "IEEE standard for modeling and simulation (M&S) High level architecture (HLA) - framework and rules," *IEEE Std 1516-2000*, pp. i–22, Sep 2000.

[6] "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification," *IEEE Std 1516.1-2000*, pp. i–467, 2001.

[7] "IEEE standard for modeling and simulation (M&S) High level architecture (HLA)-object model template (OMT) specification," *IEEE Std 1516.2-2000*, pp. i–130, 2001.

[8] Young Jae Kim and Tag Gon Kim, "A Heterogeneous Simulation Framework Based on The DEVS Bus and The High Level Architecture," *WSC-98*, Washington, D.C, USA, pp. 421 - 428, Dec. 1998.

[9] Gauthier Quesnel, Rapha¨el Duboz, E´ric Ramat and Mamadou K. Traor´e, "VLE – A Multimodeling and Simulation Environment," Proceedings of the 2007 SCSC, San Diego, California, USA, pp. 367 – 374, July 2007.

[10] Jan Himmelspach and Adelinde M. hrmacher ,"Plug'n simulate," *Proceedings of the 40th Annual Simulation Symposium (ANSS'07),* pp.137 -143, March 2007.

[11] Wan Bok Lee and Tag Gon Kim, "Simulation Speedup for DEVS Models By Composition-based Compilation," *Summer Computer Simulation 2003,* Montreal, Canada, pp. 395 - 400, July 2003.

[12] R.M.Fujimoto, *Parallel and Distributed Simulation Systems*, WILEY INTERSCIENCE, 2000.

[13] Tag Gon Kim and Myung Soo Ahn, "Reusable Simulation Models in an Object-Oriented Framework," *Chapter 4, Object-Oriented Simulation*, IEEE Press, 1996, U.S.A.

[14] ChangHo Sung, JeongHee Hong and Tag Gon Kim, "Interoperation of DEVS Models and Differential Equation Models using HLA/RTI," *Proceedings of the 2009 Spring Simulation MultiConference*, pp. 387 − 392, 2009.