Framework for Simulation of Hybrid Systems: Interoperation of Discrete Event and Continuous Simulators Using HLA/RTI

Changho Sung* and Tag Gon Kim†
Department of Electrical Engineering
KAIST, Daejeon, Republic of Korea
Email: chsung@smslab.kaist.ac.kr*, tkim@ee.kaist.ac.kr†

***Abstract*—A hybrid system is a combination of discrete event and continuous systems that act together to perform a function not possible with any one of the individual system types alone. A simulation model for the system consists of two sub-models, a continuous system model, and a discrete event model, and interfaces between them. Naturally, the modeling/simulation tool/environment of each sub-model may be different and specific to the model types and the associated modeling formalisms. This paper proposes a framework for simulating hybrid systems by means of interoperation between existing simulators for a continuous model and a discrete event model using a High Level Architecture (HLA). Each simulator is executed using different simulation algorithm, and pre-simulation methodology is applied for synchronization of the different simulators. The main advantage of the proposed approach is the reuse of simulation models, which are developed in their own simulation environments of heterogeneous types. The proposed framework was applied to water level simulation.**

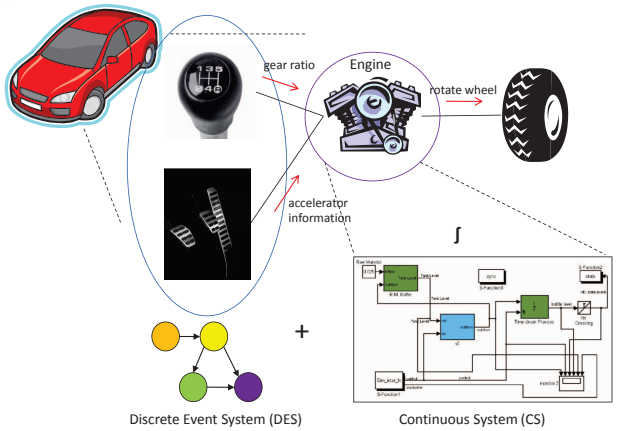


Fig. 1. Hybrid system

I. Introduction

Systems modeling and simulation process vary according to the types and modeling objectives of the systems. Given modeling objectives, modelers should determine modeling levels and simulation tools. For example, continuous models relate to the change of values via time, and discrete event models are concerned with state transitions by events. On the other hand, a hybrid system is defined as a mixture of continuous and discrete time components that act together to perform a function that is impossible with any one of the individual system types alone. An automobile is a kind of hybrid system, and each component is shown in Fig. 1. The automobile is a combination of several sub-systems. Among them, the engine system is represented by several differential equations. The equations describe compressible flow through the engine, heat transfer, mechanical vibration, overall engine dynamics, etc. On the other hand, a gear system is represented by discrete event models. The gear ratio changes from 1 to 5 or 6, and the change occurs manually or automatically. The information becomes a parameter for solving the differential equations of the automobile engine. In order to model and simulate a hybrid system, such as an automobile, each sub-systems is modeled in its own modeling environment, and the combined simulation method is needed.

Simulating models developed in different methodologies and environments is difficult because their simulation algorithms and information are disparate from each other. To overcome the difficulty in simulating hybrid systems, modeling methods based on a combined formalism, which has both discrete event and continuous features, are proposed. However, the combined modeling method is limited by the non-reusability of existing simulators. In addition, efforts have been made to integrate the simulators with a specific interface, but the method also needs modification of the existing simulator for the interface. To solve the limitations, we propose an interoperation framework for the simulation of hybrid systems with existing discrete event and continuous simulators.

In general, the modeling of hybrid systems may require distributed simulation due to system complexity, reusability, and interoperability requirements. In order to simulate different types of models, developed simulators should interoperate with others developed from various environments. Interoperation could be achieved by using the High Level Architecture (HLA) developed by the US Department of Defense (DoD) for modeling and simulation [1] [2] [3]. The Run-Time Infrastructure (RTI) is an implementation of the HLA interface, and it supports developers who use HLA services [4]. This paper proposes an interoperation framework for the simulation of hybrid systems using HLA/RTI. The framework includes hybrid simulation algorithms and considers existing simulation models.

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

This paper is organized as follows. Section II presents several related works and previous approaches to the simulation of hybrid systems. Section III explains the simulation method for hybrid systems, and the proposed framework for hybrid simulation is described in Section IV. Section V demonstrates an example of a water tank control system and its simulation results. Finally, Section VI concludes the discussion.

II. Related Work

In the past years, some effort has been made to simulate hybrid systems. Most of the research has concentrated on specifying a unified methodology for hybrid simulation. We call the methodology *stand-alone* simulation. The *stand-alone* simulation of hybrid systems means that the systems are modeled and simulated by only one modeling tool or methodology without the help of any others. For example, the following tools are used for stand-alone simulation and modeling systems: MATLAB/Simulink, Ptolemy II [5], AnyLogic [6], and PowerDEVS [7]. The *SimEvents* toolbox [8] supports MATTLAB/Simulink supports discrete event simulation, and the *Event Translation* block enables communication between *SimEvents* and other continuous operational blocks. Ptolemy II is a computational framework for embedded systems that focuses on concurrent systems, and HyVisual [5] is a hybrid system modeling tool that provides a visual syntax. AnyLogic is a tool for modeling and simulating hybrid systems and a way of HLA support integration in the tool simulation engine. PowerDEVS proposes discrete event methods for the numerical integration of ordinary differential equations (ODEs). These approaches commonly propose the use of a unique language for the specifications of the overall system, which consists of different model types. These require new modeling formalisms or tools. Therefore, existing models may not be reusable, and the model development process has been long.

In addition, some research has simulated models separately, each of which is communicated by procedure calls. We call the methodology *integrated* simulation. The methodology is expressed by MATLAB/Simulink interfaces, such as CODIS [9] and BCVTB [10], and is applied to most of the hybrid control systems. CODIS is a framework for simulating continuous and discrete systems, and it has a co-simulation bus to integrate a SystemC model and a MATLAB/Simulink model. BCVTB is based on the Ptolemy II software environment, and it integrates with a MATLAB/Simulink model. These methodologies use a specific interface between two models, such as Inter-Process Communication (IPC) and function calls. Two models are executed using such an interface in one process and tightly coupled. Thus, certain limitations have existed on the simulation of models developed in different modeling environments.

Another simulation method consists of the interoperation between discrete event system models and continuous system models using HLA/RTI. The models are spatially separated explicitly. In a few studies, the HDEVSimHLA [11] framework provides an HLA/RTI interface for interoperating the DEVS models and the MATLAB/Simulink models. The framework uses the time management service of HLA/RTI for time synchronization between heterogeneous models, and it uses the analog-event and event-analog functions for data exchange. In addition, AnyLogic [6] and MATLAB provides HLA/RTI interfaces as a package of the application. Developers have to use the HLA support modules and implement new interface codes in each model. This interoperation methodology is the most efficient method for reusing existing models and modeling systems in different environments. In these researches, however, existing models may not also be reusable, because the models are modified to use HLA/RTI, and new interface codes have to be inserted.

In comparison with previous research, the contribution of this paper consists in the concept of reusing existing models without any modifications. The problem common to previous studies has been the difficulty of reusing existing continuous and discrete event models, which are developed in different tools, and inserting specific interfaces into them. To overcome the problems, this paper proposes using a distributed hybrid simulation framework for the interoperation of existing models developed in different tools. The framework supports interoperation of such models via the standard interface of HLA. It also supports time synchronization between models and data conversion for communicating with each other. A main purpose of the proposed approach is to exploit the reusability of existing continuous and discrete event models, each of which is simulated in independent simulation environment/tools.

III. Hybrid Simulation Algorithm

In general, a discrete event model is executed by an event-driven approach. It advances time asynchronously—that is, it skips time when there is no change in the state of the system and examines the changes. Therefore, it is important to find the minimum next-event time of the discrete event model, because interval time between current and next-event time is not considered. It takes the execution time of an event from the top of a sorted stack, calculates its effect on the system state, schedules dependent events by placing them into appropriate places in the event stack, and advances the simulation clock to the next scheduled event. On the other hand, a continuous model is commonly executed by the discrete-time simulation method. It advances synchronously—that is, it uses the continuous state at some time, t, to compute the value of a continuous state variable at some short time later at $t + \Delta t$. The continuous variable solver then advances the simulation clock to $t + \Delta t$ and continues [12]. The time step may be determined by the numerical analysis method of the continuous model.

Simulation of a hybrid system goes beyond the simple numerical integration of the continuous variables and specification of the particular interactions that may occur among continuous and discrete state variables; there is a fundamental mismatch between the way time is advanced by a discrete event-driven versus a continuous variable integrator. In order to solve the mismatch, simulation time synchronization between a discrete event and a continuous model is required. However,

the next time of the continuous model for scheduling is not predictable, unlike the time of discrete event model. To solve the problem, we use a pre-simulation method for scheduling the continuous model.

A. Pre-Simulation

Simulation time synchronization requires a chronological order of logical times between continuous and discrete event simulations. To synchronize the simulation times with different time advance mechanisms we introduce the concept of pre-simulation for continuous simulation. Pre-simulation [11] is continuous simulation except that it occurs within a predefined time window to unify the time advance mechanism between two simulations, and the next scheduled time is located through pre-simulation. Hence, *pre-simulation* is defined as *simulating a continuous time model in advance to find the next scheduled time of the model*. To explain this simulation method, we used the concept of *state event* and *time event* [13]. A state event is triggered by the system status, the fulfilling certain conditions (so-called state condition)—e.g., the crossing of a state variable through a prescribed threshold. A time event is a scheduled event whose realization time is known in advance. A time event occurs at a given point in time, independent of the continuous-time state of the model. Therefore, a time event is predictable.

Algorithm 1 shows the pre-simulation of a continuous model. Let t_C denotes the current simulation time and t_R be the next request time. The continuous model is simulated between t_C and t_R, and the interval is a predefined time window. If a state event is detected within the time window, the state event occurs, and the event time, t_E, is located in the threshold point. t_E is set to the next scheduled time. On the other hand, if not, a time event occurs and the next scheduled time is set to t_R. After simulation, reinitialization of state variables is needed. The pre-simulation process allows us to control the time advancement of a continuous model.

Algorithm 1 Pre-Simulation of Continuous Model

procedure PRE-SIMULATION(t_C, t_R)

1: t_C : current simulation time
2: t_R : next requested time
3: t_E : state event time
4: t_N : next scheduled time
5: simulation in $[t_C, t_R]$
6: **if** state event detected **then**
7: send state event and the time(t_E) is located
8: $t_N \leftarrow t_E$
9: **else**
10: send time event
11: $t_N \leftarrow t_R$
12: **end if**
13: reinitialization of state variables
14: **return** t_N

B. Time Advancement of Continuous Model

Let ΔT denotes the time window to find the next scheduled time, and let s be the state of a continuous model. The state, s, is either *ACTIVE* or *PASSIVE*. An *ACTIVE* state means that the continuous model is in the execution state for generating a state event or time event, and the *PASSIVE* state is the waiting state for events from the discrete event model. Algorithm 2 shows the time advance mechanism of a continuous model. The next-event time of a continuous model is the result of pre-simulation operation in an *ACTIVE* state. In the *PASSIVE* state, the pre-simulation process is not needed, and the next time is infinity because the continuous model does not advance its time until it receives data from the discrete event model.

Algorithm 2 Time Advancement of Continuous Model

procedure TIME-ADVANCE

1: t_C : current simulation time
2: t_N : next scheduled time
3: s : state of continuous model {ACTIVE, PASSIVE}
4: ΔT : time window
5: initialization
6: $s \leftarrow$ ACTIVE
7: **loop**
8: **if** s is ACTIVE **then**
9: $t_R \leftarrow t_C + \Delta T$
10: $t_N \leftarrow$ Pre-Simulation(t_C, t_R)
11: **else**
12: $t_N \leftarrow \infty$
13: **end if**
14: request next time advance(t_N)
15: **end loop**

C. Hybrid Simulation

1) Synchronization: Using the pre-simulation method, we can find the next scheduled time of a continuous model. The time scheduler calculates the next time of the simulators and sends the time information and interaction message between the models. In hybrid simulation, there are two types of interaction, as shown in Fig. 2.

- A discrete event changes the relation of a continuous variable.
- A continuous variable reaches a threshold point, and it causes a discrete event.

We can predict the next-event time of a discrete event model, and the time, t_1, is requested to time scheduler. This means that the discrete event model has no events and no changes of state until t_1. A continuous model executes a pre-simulation with time window ΔT until t_1 to check state events. If there is no state event in the continuous model, the discrete event on t_1 is transferred to the continuous model, and the relation of the continuous variable, S_cont, is changed. If a state event occurs during pre-simulation, the continuous model causes a discrete event at the threshold point, t_2. The event

changes the state of the discrete event model, and the model is re-scheduled.

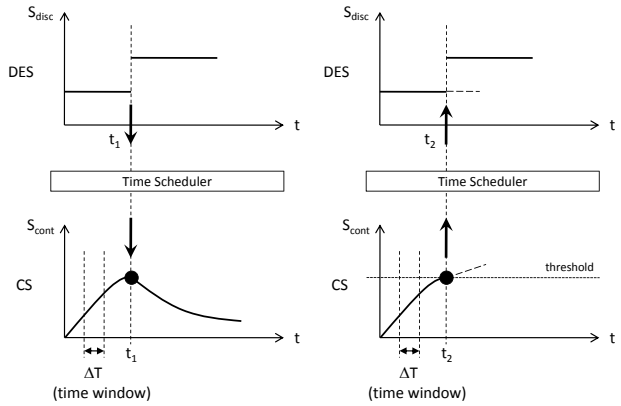


Fig. 2. Pre-simulation of CS and time scheduling

2) Data Conversion: A discrete event model requires a discrete input event for its simulation, and it generates a discrete output event. A continuous model needs an input analog signal for its simulation, and it generates other output signal. Fig. 3 shows the need of the data conversion between the models. In a hybrid simulation, an output of a discrete event model becomes an input of a continuous model, and vice versa.

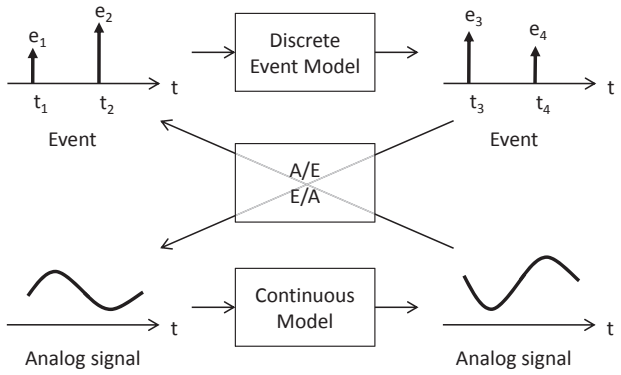


Fig. 3. Why is the data conversion necessary?

For this data conversion, we use an analog-to-event (A/E) and event-to-analog (E/A) converter [11]. The E/A conversion is conducted as shown in the event-function mapping table shown in Fig. 4. The table defines how the discrete events are converted to continuous signals. The signal is a function of time. On the other hands, A/E conversion is conducted by zero-crossing detection [14]. An event is triggered when a continuous signal reaches the threshold level, and the event message is determined by the hit crossing direction.

IV. Proposed Framework for Simulation of Hybrid Systems

From a software engineering perspective, the software quality has to be considered for the modeling and simulation of

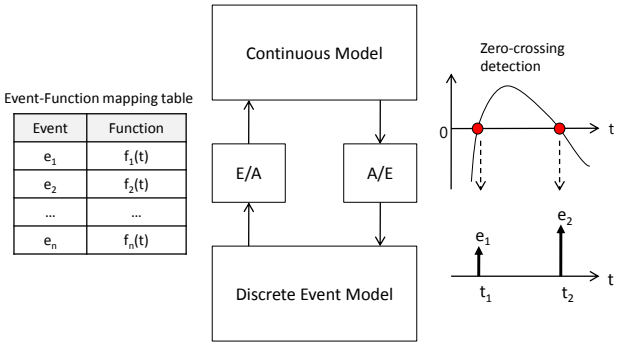


Fig. 4. Data conversion using A/E and E/A converter

hybrid systems. The new requirements for software quality are *reusability*, *maintainability*, and *interoperability* [15]. In order to satisfy the requirements, we propose an interoperation framework for hybrid systems using HLA/RTI. Our proposed framework has several assumptions about simulation environment as follows:

1) A MATLAB/Simulink model is used for the modeling and simulation of continuous systems. MATLAB is a representative mathematical computing tool that provides a flexible environment for continuous systems modeling and simulation.
2) Lookahead = 0. (This means that the models are simulated without delay.)
3) Mutually exclusive activation between DES and CS. In our approach, DES acts as a controller for CS in such matters as the relation between a controller and a plant.

A. Framework Design and Implementation

A discrete event model and a continuous model are simulated in their own ways, as explained in Section III. In addition, the simulators may be developed in different environments. To interoperate different simulators without any modification, we use an HLA interface, a so-called *HLA adaptor*, to connect the simulators and RTI. The simulator requires essential services and callbacks to simulate itself, and the HLA adaptor helps the modelers to select the necessary interfaces easily.

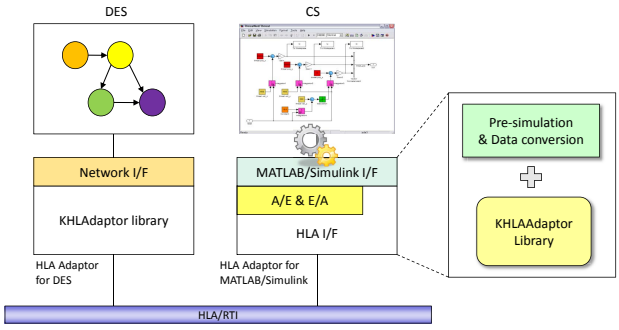


Fig. 5. Proposed framework for simulation interoperability of hybrid systems

Fig. 5 shows the proposed framework for the interoperation of the hybrid system models. A discrete event model uses

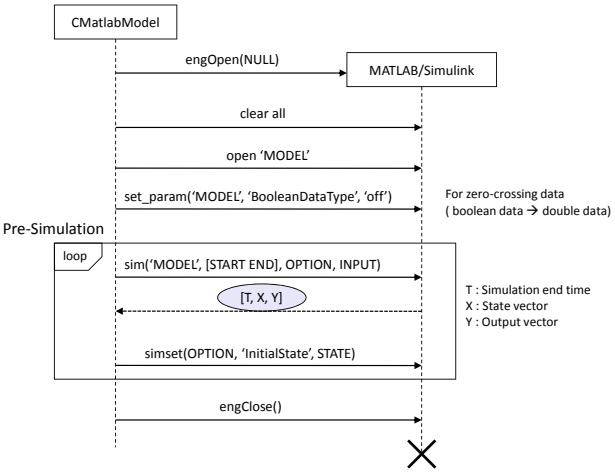


Fig. 6. Implementation of the MATLAB/Simulink interface in an HLA adaptor

an adaptor for HLA, and the adaptor is developed using a KHLAAdaptor library [16]. On the other hand, the existing adaptor library cannot be used for a continuous model, because the time advancement of a continuous model needs pre-simulation, as explained in the previous section. Therefore, an adaptor for a continuous model needs modification of the existing KHLAAdaptor library, and it includes pre-simulation algorithm and data conversion rules. To support the functions, a *CMatlabModel* class is added in the KHLAAdaptor library. The role of the class is depicted in Fig. 6. Using the class, the adaptor opens a continuous model developed using the MATLAB/Simulink, and controls the model. For pre-simulation, the adaptor executes the model during the time window, from *START* to *END* time, using input signal data, and then the return values are used for next pre-simulation. The functions for the MATLAB/Simulink interface in Fig. 6 are explained in Table I. The *sim* and *simset* functions are used for the pre-simulation of a continuous model.

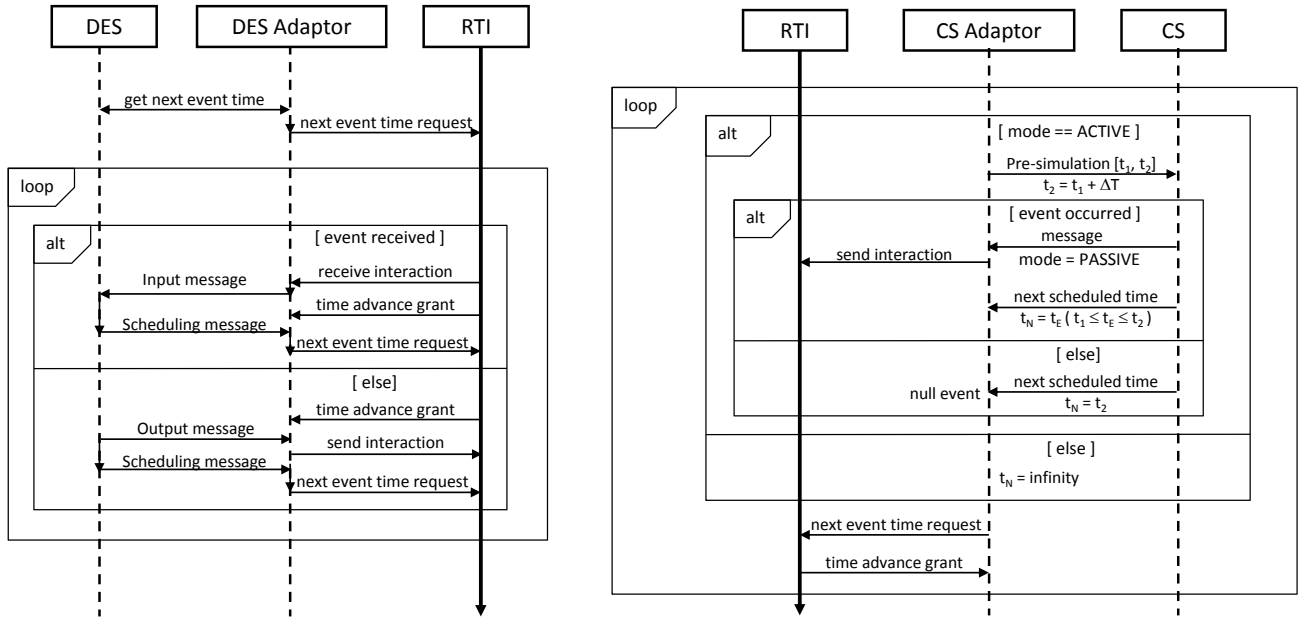
TABLE I
FUNCTIONS FOR MATLAB/SIMULINK INTERFACE

Function	Description
engOpen	State MATLAB process
open	Open Simulink model
set_param	Set simulation parameters
sim	Simulate a model with a specific input data during [start time, end time]
simset	Edit simulation parameters
engClose	Quit MATLAB process

B. Simulation of the Framework

The simulation of the proposed framework is depicted in Fig. 7. An adaptor for a discrete event model checks the next-event time from the model and requests the time to RTI. The next time is deterministic because the scheduling time is predictable in the discrete event model. If an event is received, *receiveInteraction* API of HLA is called, and the message is transferred to the discrete event model. After it calculates the message, the discrete event model is re-scheduled, and the adaptor requests the new scheduling time to RTI. If no event is received, the model generates an output message, and the adaptor sends the message to the continuous model through RTI.

An adaptor for a continuous model conducts the pre-simulation algorithm and data conversion. The adaptor controls the continuous model according to the *TIME-ADVANCE* procedure, as shown in Algorithm 2. In *ACTIVE* state, the



(a) Sequence of DES

(b) Sequence of CS considering pre-simulation

Fig. 7. Simulation of the proposed framework

adaptor executes the continuous model during $[t_1, t_1 + \Delta T]$. The pre-simulation returns information about a state event or a time event. If a state event occurs at the threshold point, the adaptor sends the event to the discrete event model through the *sendInteraction* API of RTI. The state of the adaptor changes to *PASSIVE*, and the next scheduled time is set to the event time, t_E. If no event has occurred during the pre-simulation, a time event occurs at the end of the pre-simulation. The next scheduling time is set to $t_1 + \Delta T$. The adaptor requests the next-event time, t_E of $t_1 + \Delta T$, and this process is repeated. When a discrete event is received by the adaptor for the continuous model, the event converts into a continuous signal according to an event-function mapping table. The converted signal is used for an input of the *sim* function of a *CMatlabModel* class. An adaptor for a continuous model is, as it were, a state event finder and a data converter as well as an agent to connect with HLA/RTI. The time management of HLA supports synchronization between both adaptors.

V. Experimentation

The proposed framework is applied to the simulation of a water level control system. In Fig. 8, a water level control system consists of two subsystems, a controller and a water tank. In the water tank, there are two water level sensors (*overflow, shortage*), and the detected state of the water tank is transmitted to the controller. The controller uses an ON/OFF switch to control the water tank, and the control signal controls the operation of the tank.

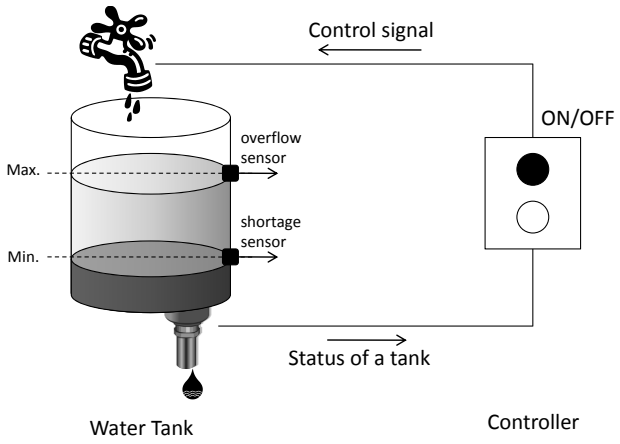


Fig. 8. Water level control example

A. Continuous Model

Fig. 9 shows water entering the tank from the top and leaving through an orifice in its base. The rate at which the water enters is proportional to the voltage, V, applied to the pump. The rate at which the water leaves is proportional to the square root of the height of the water in the tank. The differential equation for the height of water in the tank, H, is given in equation 1.

$$A\frac{dH}{dt} = bV - a\sqrt{H} \tag{1}$$

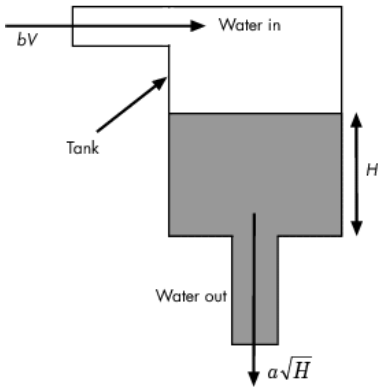


Fig. 9. Water tank example

, where

V : input voltage of water tank
A : cross-sectional area of a tank
b : a constant related to the flow rate into a tank
a : a constant related to the flow rate out of a tank

The mathematical form of the water tank model is described using a MATLAB/Simulink as shown in Fig. 10. The model contains one state, H, one input, V, and one output, H. The height of the water tank is the state of the equation, and the value of the state is changed via continuous time. The input of the model is a voltage signal, and the value determines the quantity entering the water tank.

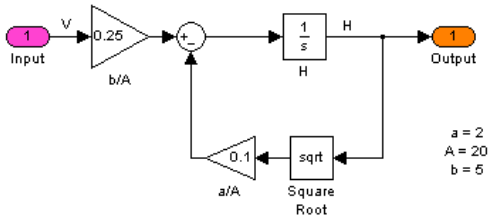


Fig. 10. Simulink model of water tank (CS)

B. Discrete Event Model

A controller is a discrete event system that controls the volume of water entering a tank. This system is modeled using DEVS formalism, and the model is shown in Fig. 11. The initial state of the model is *WAIT*, and the time advance of the state is *infinity*. When the model receives a *shortage* event from the water tank, it changes the state from *WAIT* to *ON*. Because the event means that the tank is lacking water, an *on* event occurs to control the entering water. On the other hand, when an *overflow* event is received, the state is changed to *OFF*, and then an *off* event occurs.

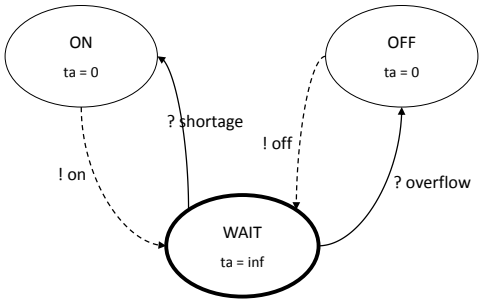


Fig. 11. DEVS model of water control (DES)

C. Data Conversion

Data conversion is necessary for co-simulation between simulators with different characteristics. A water tank needs analog signals to operate a pump, and it has continuous values represented by the height of the water in the tank. In contrast, a controller requires events for a state transition.

The continuous values for the amount of the water in the tank are detected by threshold sensors. Fig. 12 describes the relation between threshold values and events. When the height of the water reaches *4* because the water is rising, an $overflow$ event occurs. Likewise, a $shortage$ event occurs when the height reaches *1* because the water is falling. The events are used for inputs to the controller. We used the hit crossing block of the MATLAB/Simulink to convert the analog signal into the discrete event.

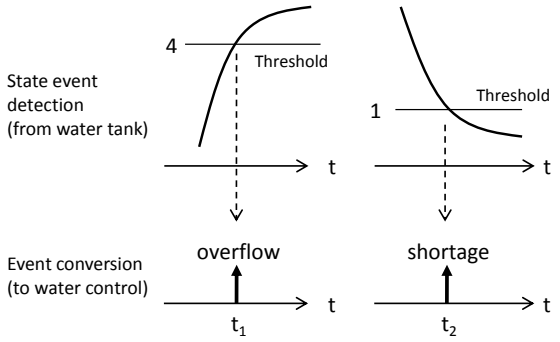


Fig. 12. A/E conversion: state event detection from a water tank model

When events occur from a water level controller, they have an effect on the operation of the water tank. The events are converted to a segment function, in that the water tank is a continuous model. The conversion relation is shown in Table II. An event $on(t = t_1)$ is transformed into $f_1(t) = u(t - t_1)$ which means that input value of a water tank is one for a positive argument, i.e. after t_1 time. Likewise, an event $off(t = t_2)$ is transformed into $f_2(t) = u(t_2 - t)$.

TABLE II
EVENT-ANALOG CONVERSION OF WATER LEVEL SYSTEM

Event	Analog
$on(t = t_1)$	$f_1(t) = u(t - t_1)$
$off(t = t_2)$	$f_2(t) = u(t_2 - t)$

D. Federation Design and Implementation

In order to simulate a controller and a water tank model with RTI, an HLA interface should be implemented for each simulator. The HLA interface is a form of adaptor, as explained in Section IV. In the case of the controller, the design method of its adaptor is shown very well in [17]. The design and implementation of the HLA adaptor for the water tank as a continuous model is conducted using an adaptor for the MATLAB/Simulink model described in the previous section.

```
(OMDT v1.3.5.17)
(ObjectModel (Name " ")
             (VersionNumber " ")
             (Type FOM)
             (ModificationDate 11/29/2010)
             (MOMVersion "1.3")
             (FEDname "FederationName")
  (EnumeratedDataType (Name "CONTROL_COMMAND")
        (AutoSequence No)
        (StartValue 1)
        (Enumeration (Enumerator "ON")
                     (Representation 1))
        (Enumeration (Enumerator "OFF")
                     (Representation 2))
  )
  (Interaction (ID 1)
     (Name "CONTROL")
     (ISRType IR)
     (DeliveryCategory "reliable")
     (MessageOrdering "timestamp")
     (Parameter (Name "ID")
                (DataType "unsigned long")
                (Cardinality "1")
                (Accuracy "perfect")
                (AccuracyCondition "always")
     )
     (Parameter (Name "COMMAND")
                (DataType "CONTROL_COMMAND")
                (Cardinality "1")
                (Accuracy "perfect")
                (AccuracyCondition "always")
     )
  )
)
```

Fig. 13. Common data for simulation of water level system

The common data between simulators, the so-called Federation Object Model (FOM), includes one interaction, and it has one parameter, as shown in Fig. 13. The interoperability data is a discrete event, ON or OFF. The event is converted to the segment function defined in Table II in the HLA adaptor for a continuous model.

In the initialization stage, input ports and hit crossing points are registered in the HLA adaptor. The initial signal of the input port $control$ is 1, which denotes a positive value of $u(t)$. In the E/A conversion stage, the input signal function is determined by input events. When an input event is OFF, the function of the input port $control$ is set to 0, which means a negative value for $u(t_1 - t)$, and t_1 is event occurrence time. In the case of ON, the function is 1, i.e., $u(t - t_2)$.

The overall architecture of the water level control system is shown in Fig. 14. The controller model is implemented using DEVSim++ [18], which is C++ library to support implementation of the DEVS models, and the water tank model is a form of the MATLAB/Simulink model. Two models are connected to their adaptors, which support the HLA service using MÄK RTI 3.4.

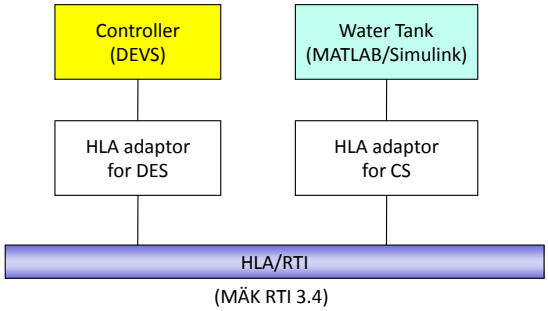


Fig. 14. Overall architecture of water level control system

E. Simulation Result

Fig. 15 shows a simulation result. The water fills the tank when the *ON* switch of a controller is turned on. When an overflow sensor, which is located at height 4, detects the water, the height of the water in the tank goes down. In contrast, when the water decreases to height 1, a shortage sensor is activated, and the height of the water begins to rise. The controller and water tank model were not modified for simulation interoperability, and the proposed framework enables reuse of the simulators.

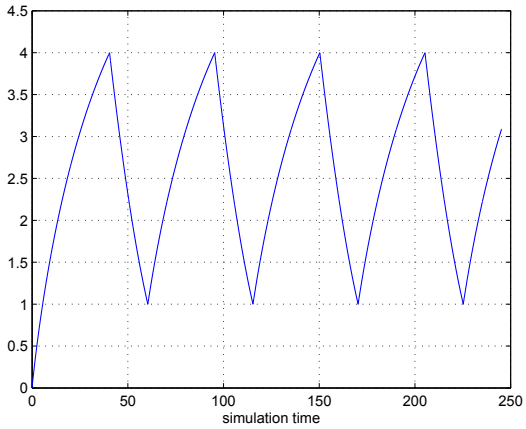


Fig. 15. Variation of water height

VI. Conclusions and Future Works

This paper proposes an interoperability framework for the simulation of hybrid systems. A hybrid system is a mixed form of discrete events and continuous systems. Each system has different characteristics, and the models are developed in different environments. Our adaptor-based interoperation framework using HLA/RTI makes it possible to reuse existing models and to interoperate with other simulators. The adaptor includes an HLA interface, pre-simulation algorithms, and data conversion. The water level experiment shows the hybrid simulation using an HLA adaptor, and we could reuse the existing models without any modification. Further studies are needed to simulate discrete event models and general continuous models in addition to MATLAB/Simulink models. Moreover, faster and more efficient pre-simulation algorithms should be discussed.

Acknowledgment

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

References

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

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

[3] "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.

[4] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly, "The Department of Defense High Level Architecture," in *Proceedings of the 29th conference on Winter simulation*, ser. WSC '97, Atlanta, GA, United States, Dec. 1997, pp. 142–149.

[5] H. Zheng, "Operational semantics of hybrid systems," Ph.D. dissertation, Berkeley, CA, USA, 2007, adviser-Edward A. Lee.

[6] A. Borshchev, Y. Karpov, and V. Kharitonov, "Distributed simulation of hybrid systems with AnyLogic and HLA," *Future Gener. Comput. Syst.*, vol. 18, no. 6, pp. 829–839, 2002.

[7] E. Kofman, M. Lapadula, and E. Pagliero, "PowerDEVS: A DEVS-Based Environment for Hybrid System Modeling and Simulation," School of Electronic Engineering, Universidad Nacional de Rosario, Tech. Rep. LSD0306, 2003.

[8] M. Clune, P. Mosterman, and C. Cassandras, "Discrete Event and Hybrid System Simulation with SimEvents," in *Discrete Event Systems, 2006 8th International Workshop on*, Ann Arbor, MI, USA, Jul. 2006, pp. 386–387.

[9] F. Bouchhima, G. Nicolescu, E. M. Aboulhamid, and M. Abid, "Generic discrete-continuous simulation model for accurate validation in heterogeneous systems design," *Microelectron. J.*, vol. 38, no. 6-7, pp. 805–815, 2007.

[10] M. Wetter and P. Haves, "A Modular Building Controls Virtual Test Bed for The Integration of Heterogeneous Systems," in *SimBuild 2008 - 3rd National Conference of IBPSA-USA*, Berkeley, CA, USA, Jul. 2008, pp. 69–76.

[11] S. Y. Lim and T. G. Kim, "Hybrid Modeling and Simulation Methodology based on DEVS Formalism," in *SCSC '2001*, Orlando, FL, USA, Jul. 2001, pp. 188–193.

[12] J. F. Klingener, "Programming combined discrete-continuous simulation models for performance," in *WSC '96: Proceedings of the 28th conference on Winter simulation*, Coronado, CA, USA, Dec. 1996, pp. 833–839.

[13] F. E. Cellier, "Combined continuous/discrete system simulation by use of digital computers: techniques and tools," Ph.D. dissertation, Zürich, Switzerland, 1979.

[14] P. A. Fishwick, *Handbook of Dynamic System Modeling (Cpaman & Hall/Crc Computer and Information Science)*. Chapman & Hall/CRC, 2007.

[15] J. P. Cavano and J. A. McCall, "A framework for the measurement of software quality," in *Proceedings of the software quality assurance workshop on Functional and performance issues*, 1978, pp. 133–139.

[16] J.-H. Kim, S.-Y. Hong, and T. G. Kim, "Design and Implementation of Simulators Interoperation Layer for DEVS Simulator," in *M&S-MTSA'06*, Calgary, Canada, Jul. 2006, pp. 195–199.

[17] T. G. Kim, C. H. Sung, S.-Y. Hong, J. H. Hong, C. B. Choi, J. H. Kim, K. M. Seo, and J. W. Bae, "DEVSim++ Toolset for Defense Modeling and Simulation and Interoperation," *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*.

[18] T. G. Kim, *DEVSimHLA User's Manual*, 2007. [Online]. Available: http://smslab.kaist.ac.kr