REAL-TIME DEVS SIMULATION: CONCURRENT, TIME-SELECTIVE EXECUTION OF COMBINED RT-DEVS MODEL AND INTERACTIVE ENVIRONMENT

Seong Myun Cho and Tag Gon Kim

Systems Modeling Simulation Lab.,
Department of Electrical Engineering, *KAIST*
373-1 Kusong-dong, Yusong-Ku, Taejon 305-701, KOREA
e-mail: smcho{tkim}@coregate.kaist.ac.kr

Keywords: Reactive, real-time DEVS simulation, Time selectivity, Concurrent model execution

ASTRACT

Real-time, discrete event simulation often employs a simulation model that interacts with a surrounding environment, such as software components, hardware components or human operators. In such simulation, a simulator must interact with its environment such that events from its environment should be processed in a timeliness manner. Simulation methodologies developed for non-interactive simulation, such as performance simulation, do not have such capabilities in event processing. We propose a real-time, interactive simulation methodology which concurrently executes real-time DEVS(RT-DEVS) models and its environment in a time selective manner. The methodology is implemented in an object-oriented environment by using multi-thread facilities running on a multi-thread OS.

1. INTRODUCTION

Discrete-event simulation is widely used not only to predict performance but also to analyze behavior, of complex systems. Real-time, discrete event simulation is a means to verify a real-time system in which a simulation model interacts with a surrounding environment, such as software components, hardware components or human operators. In such simulation, a simulator must handle external events from its environment in a timeliness manner. Simulation methodologies developed in non-interactive simulation, such as performance simulation, do not have such capabilities in event processing. This is because such methodologies employ virtual time, not real-time, in event scheduling and processing. We propose a real-time, interactive simulation methodology for combined discrete event models and a surrounding environment. Within the methodology, a real time system is modeled by the RT-DEVS formalism(Hong 1997), a real-time extension of Zeigler' s DEVS formalism(Zeigler 1984); interactive components are used as they are. The methodology concurrently executes real-time DEVS (RT-DEVS) models and its environment in a time selective manner. The methodology is implemented in an object-oriented environment by using multi-thread facilities running on a multi-thread OS. A simple example of real-time simulation for verification of a communication protocol will be given.

This paper is organized as follows. Section 2 presents the time selectivity problem which inherently appears in real-time, reactive simulation. Section 3 describes the RT-DEVS formalism used in modeling of real-time subsystems. An algorithm for real-time simulation and its realization are given in section 4. A real-time simulation of alternating bit

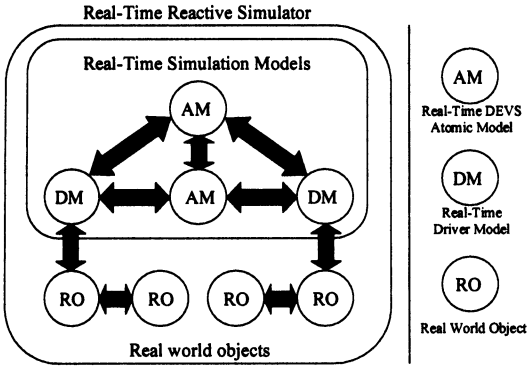

Figure 1. Reactive Real-Time Simulation

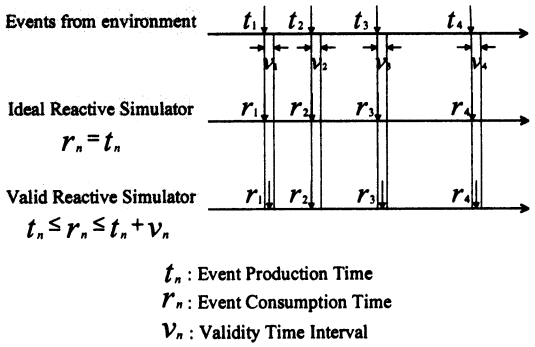

Figure 2. Time Selectivity In Event Handling

protocol is presented in section 5. Conclusion is given in section 6.

2. TIME SELECTIVITY PROBLEM

Figure. 1 shows a real-time, reactive simulation environment in which simulation models are interacting with real world objects. In the simulation, a time selectivity problem naturally appears. The usefulness of an event in such simulation often depends upon a time interval between an event occurrence from the environment and an event consumption at a simulation model. Consequently, a time-critical event must have an associated attribute of a *validity-time interval.* Figure 2. shows time selectivity in real-time, reactive simulation. Each event produced at time t has an associated attribute of a validity-time interval v. Time denoted by r represents the instant at which the event is consumed by a simulation model.

Non-real time simulation which has no interaction with its environment can not guarantee processing events from its environment within a validity-time interval. This behavior could lead to the loss of events and the generation of wrong output events. In order to solve such time selectivity problem, simulator should process events within its validity-time interval.

A real-time, reactive DEVS simulator can solve the time selectivity problem to a certain extent by executing its models and interacting environment concurrently. To do so, the simulator instruments a driver model (DM) and associates it with each real world object. Driver models function as interfaces between simulation models and real world objects interacting with them.

Now, driver models and simulation models are executed concurrently to respond external events from real world objects in a timeliness manner. Driver models periodically monitor the external events generated from the associated real world object. Once an external event from a real world object occurs a driver model converts it to an event which a simulation model interacting with the object understands. Likewise, the driver model converts an output event generated by a simulation model to an input event for an associated real world object. By doing so, simulation models have a minimum overhead in the interaction with an environment, thus avoiding an unacceptable amount of delay. In our reactive simulation, simulation models are specified by the real-time DEVS (RT-DEVS) formalism and driver models are specified as a form of an event converter.

3. RT-DEVS FORMALISM

The real-time DEVS formalism is an extension of

the DEVS formalism for real-time systems simulation. An atomic model in RT-DEVS formalism, *RTAM*, is defined as (Hong 1997):

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

, where

X : a set of external input event types

S : a sequential state set

Y : an output set

$\delta_{ext.} : Q \times X \rightarrow S$, an external transition function, where Q is the total state set of $M = \{(s,e) \mid s \in S \text{ and } 0 \leq e \leq ta(s)\}$

$\delta_{int.} : S \rightarrow S$, an external transition function

$\lambda : S \rightarrow Y$, an output function

$ta : S \rightarrow R^{+}_{0,\infty} \times R^{+}_{0,\infty}$, a time interval function, where $R^{+}_{0,\infty}$ is the non-negative real numbers with ∞ adjoined.

Note that a time advance, *ta(s)*, for RT-DEVS is given by an interval $ta(s)|_{\min} \leq ta(s) \leq ta(s)|_{\max}$.

In the DEVS formalism, virtual simulation time advances only when a simulator calls the time advance function *ta* of an atomic model. The RT-DEVS formalism replaces virtual time advance by real time advance. The actual advance of simulation time is the real execution time of δ_{ext} and δ_{int}. This definition of RT-DEVS formalism is slightly different from previous work(Hong 1997) in advancing the real time clock. In previous work(Hong 1997), an activity mapping function ψ and an activity set A are defined to advance virtual time with an executable activity associated with an event. Here, however, we represent such executable activities in two state transition functions, δ_{ext} and δ_{int}. State transition generally involves some computational work, which can be embedded in δ_{ext} and δ_{int}. The time interval function *ta* specifies the time bounds required to execute δ_{ext} or δ_{int}.

A coupled model within the RT-DEVS formalism is defined the same way as in the original DEVS formalism(Zeigler 1984) with an exception. The exception is that there is no *SELECT* function in RT-DEVS, which has been defined in the DEVS formalism to break ties for simultaneous events scheduling. This is because such simultaneous events can not be happened in a real-time simulation environment. In real-time simulation with one processor, only one event at a time can be physically processed even if more than one event occurred from the external environment .

We now give a definition for a real-time driver model, which is an interface between a simulation model and a real world object. A real-time driver model, *RTDM*, is defined as:

$$RTDM = < X, Y, T_{ME}, T_{EM} >$$

, where

$X = X_M \cup X_E$: an input events set;

X_M: input events from model

X_E: input events from environment

$Y = Y_M \cup Y_E$: an output events set;

Y_M: output events to models

Y_E: output events to environment

$T_{ME} : X_M \rightarrow Y_E$: an event translation function from a model to an environment

$T_{EM} : X_E \rightarrow Y_M$: an event translation function from an environment to a model

Note that a driver model is a duplex device for bi-directional translations of incoming events.

4. SIMULATION ENVIRONMENT REALIZATION

4.1 Overall Structure

The overall structure for real-time, reactive DEVS simulation environment is shown in Figure 3. The simulation environment employs the multi-thread facilities which enables all simulation models along with driver models to be executed concurrently. Within the environment, all coupled models are flattened to minimize communication delays during real-time simulation. Each atomic model spawns its own thread to execute its characteristic functions concurrently with other threads associated with corresponding atomic models, as shown in Figure 4.

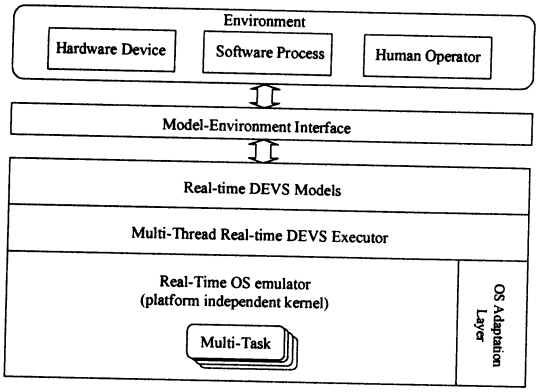

Figure 3. Simulation Framework

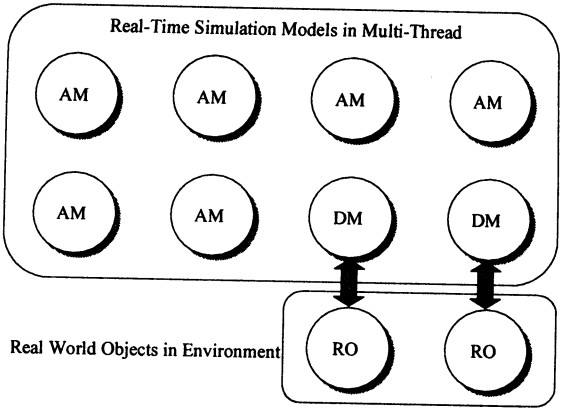

Figure 4. Concurrent Model Execution

The run-time thread scheduler activates each thread for the predefined time in a round-robin fashion. By virtue of multi-thread facilities, we can execute all models concurrently. Recall that driver models have no internal/external transition functions nor have a time advance function. It only translates events between corresponding real world object and simulator.

4.2 CME (Concurrent Model Execution) Algorithm

The execution of an atomic model is done by the following two algorithms:

1. **CME:when_rcv_(*, t):**
2. **if** $t_N|_{min} \le t \le t_N|_{max}$ **then**
3. $y := \lambda(s)$;
4. send (y, t) to associated ports;
5. signal them;
6. $s := \delta_{int}(s)$;
7. $t_L := t$;
8. $t_N := [t_L + ta(s)|_{min} - time(\delta_{int}),$
   $t_L + ta(s)|_{max} - time(\delta_{int})]$;
9. **else**
10. error;
11. **end if**

The above algorithm **when_rcv_(*, t)** handles an internal event (*, t) which notifies that a scheduled time t comes. The CME performs this procedure whenever an internal time out occurs without any interruption of external events. First, the CME checks the validity of internal timeout. Lines 3 to 8 of the algorithm correspond to the function calls sequence $[\lambda, \delta_{int}, ta]$

1. **CME:when_rcv_(x, t):**
2. **if** $t_L \le t \le t_N|_{max}$ **then**
3. $e := t - t_L$;
4. $s := \delta_{ext}(s,e,x)$;
5. $t_L := t$;
6. $t_N := [t_L + ta(s)|_{min} - time(\delta_{ext}),$
   $t_L + ta(s)|_{max} - time(\delta_{ext})]$;
7. **else**
8. error;
9. **end if**

The above algorithm **when_rcv_(x, t)** handles an external event (x, t). The CME performs this procedure whenever an external event is received. As in the previous algorithm, it first checks the validity of the external event' s generation time. Lines 3 to 6 of the algorithm correspond to the function calls sequence $[\delta_{ext}, ta]$

1. **CME:main():**
2. $s = s_0$ /* initialize */
3. $t_N := ta(s_0)$;
4. **concur_forever for each RT-DEVS model**
   /* main loop */

```
5.      wait for a signal until ITO
6.      if an external event then
7.          when_rcv_(x, t);
8.      else if an internal time out then
9.          when_rcv_(*, t);
10.     end if
11. end concur_forever
```

The above algorithm **main()** shows the main routine of the CME. Each atomic model waits for an external event until an internal timeout occurs. If an external event is occurred before an internal timeout expires, it proceeds **when_rcv_(x,t)**. If an internal time expires without external event, it proceeds **when_rcv_(*,t)** and sets new internal timeout. The internal timeout (*ITO*) is calculated as follows:

$ITO = ta - time(\delta_{ext})$ for external transition or
$ITO = ta - time(\delta_{int})$ for internal transition, where $time(\delta_{ext})$ and $time(\delta_{int})$ are the actual execution times of δ_{ext} and δ_{int}, respectively.

To synchronize time between atomic models and environment, simulator suspends atomic models for the time specified by its internal timeout. If an external event occurs during that interval, simulator wakes up the corresponding atomic model.

5. EXPERIMENT : ALTERNATING BIT PROTOCOL

5.1 Model Description

The alternating bit protocol is used to check if it guarantees the correct data delivery between a sender and receiver connected by an error channel that loses or corrupts messages. It has the name since it uses only one additional control bit in message and this control bit only alternates when the previous message is correctly received. The RT-DEVS model for the alternating bit protocol is shown in Figure 5.

Each state is explained as follows:

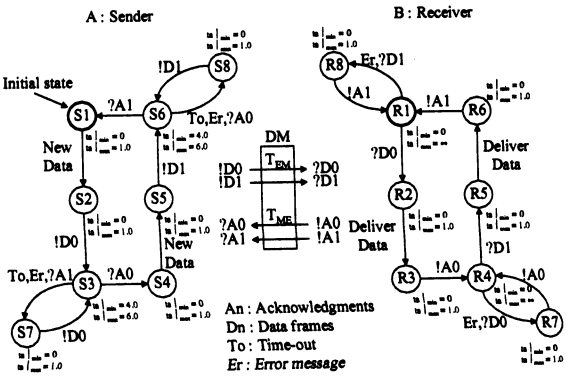

Figure 5. Alternating Bit Protocol as RT-DEVS

1) S1(4) : The sender will generate new data frame.
2) S2(5) : The sender will send a data frame with an alternating bit 0(1).
3) S3(6) : The sender waits an acknowledgment until time-out occurs.
4) S7(8) : The sender will resend previous data frame with an alternating bit 0(1).
5) R1(4) : The receiver waits a data frame with an alternating bit 0(1).
6) R2(5) : The receiver will deliver a data frame to upper layer.
7) R3(6) : The receiver will send an acknowledgment with an alternating bit 0(1).
8) R7(8) : The receiver will resend an acknowledgment with an alternating bit 0(1).

Each event is explained as follows:

1) !D0(1) : The sender sends a message with an alternating bit 0(1).
2) ?D0(1) : The receiver receives a message with an alternating bit 0(1).
3) !A0(1) : The receiver sends an acknowledgment with an alternating bit 0(1).
4) ?A0(1) : The sender receives an acknowledgment with an alternating bit 0(1).
5) To : The sender detects time-out due to the loss of a message during transmission or delayed acknowledgment of the receiver.
6) Er : The message received is corrupted.

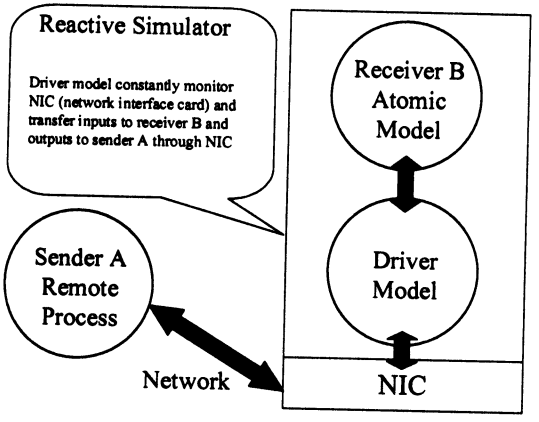

Figure 6. Reactive Simulation for Alternating Protocol

5.2 Reactive Real-Time Simulation

The receiver B is implemented as a RT-DEVS atomic model in a reactive simulation environment. The sender A runs as a software process on a remote workstation and communicates with B via network. A driver model is implemented to transfer inputs between terminal A and B by driving network interface card.

We verified correctness of our reactive simulator by observing communication between A and B in real time. Figure 7 shows event occurrences of the receiver B in real-time simulation. As shown in Figure 7, the time at which event occurs is bounded within the time intervals specified in RT-DEVS model in spite of non-deterministic execution times. However, we can manage to have state transition occur within its time interval by calculating *ITO* and suspending model accordingly.

6. CONCLUSION

A real-time, reactive simulation for combined RT-DEVS models and an interacting environment employed concurrent execution of RT-DEVS models and assoicated driver models. Such concurrent model execution has solved the time selectivity problem to a certain extent.

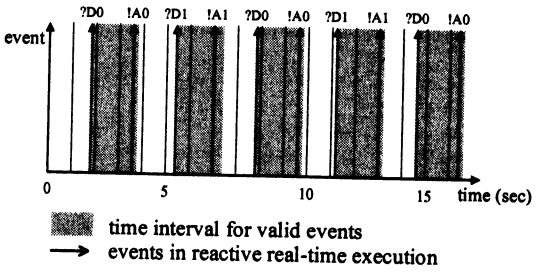

Figure 7. Events of the receiver B

Currently, a simple round robin scheduling is exploited in concurrent model execution. To improve the real-time simulator's performance, a priority-based scheduling for concurrent model execution is desirable. Such priority should be defined for each model based on both its time advance value and the validity-time interval of an event which is supposed to be received from an environment. Our work is in progress toward this direction.

References

(Coen-Porisini 1997) Alberto Coen-Porisini, Carlo Ghezzi and Richard A. Kemmerer, "Specification of real-time systems using ASTRAL." *IEEE transactions on software engineering*, vol. 23, no. 9, 1997.

(Hong 1997) Joon S. Hong, Hae S. Song, Tag G. Kim and Kyu H. Park, "A Real-Time Discrete Event System Specification Formalism for Seamless Real-Time Software Development." *Discrete Event Dynamic Systems: Theory and Applications*, 7, 355-375, 1997.

(Motus 1994) Leo Motus and Michael G. Rodd. *Timing analysis of real-time software.* Redwood Books, Oxford, 1994.

(Zeigler 1984) Zeigler, B.P., *Multifacetted Modelling and Discrete-Event Simulation.* Academic Press, New York, 1984.