REAL TIME SIMULATION FRAMEWORK FOR RT-DEVS MODELS

Seong Myun Cho and Tag Gon Kim

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

Keywords: Real time simulation, RT-DEVS formalism, Real time scheduling, Concurrent model execution

ABSTRACT

This paper presents a real time simulation framework in which simulation models are specified by the real time DEVS(RT-DEVS) formalism. Here, real time simulation denotes the framework's ability to interact with a surrounding environment, such as software components, hardware components or human operators. Typical examples are man-in-the-loop simulation such as a flight simulation or simulation of a controller with real time constraints. In such simulation, a simulator must handle external events in a timeliness manner. We propose a real time, interactive simulation methodology, which concurrently executes RT-DEVS models along with its environment in a time selective manner. The proposed simulation methodology employs an event driven scheduling policy to meet timing requirement of RT-DEVS models. We developed a real time simulation kernel that supports the proposed simulation methodology. A simple example of real time, interactive simulation demonstrates effectiveness of the proposed methodology and the developed environment.

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 may interact 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(logical) time, not real(physical) time, in event scheduling and processing.

Real time simulation involves simulation models that act as a real time program with timing constraints. Satisfying the timing constraints is very difficult due to the lack of explicit specification of such constraints in the program. To solve this problem, real time objects have been proposed [MT90, ITM90, AS91, KK94]. A real time object is a basic object that has a set of operations with timing constraints and threads, each of which is an execution unit. The main reason to define the object models is to encapsulate not only data but also timing. Mercer [MT90] presented the object model used in the ARTS real time kernel, Ishikawa [ITM90, ITM92] proposed RTC++, a real time extension of C++, and defined a real time object RTO which has two types of threads, slave and master. Attoui[AS91] proposed another real time object model MO2. MO2 is an object-

oriented model which integrates the features of both DBMS and real time systems. Kim and Kopetz [KK94, Kim94, Kim97] suggested RTO.k (also called time-triggered RT-object) model. RTO.k extends the conventional object models in four ways: deadline for each execution of a method, separation of time-triggered methods and message-triggered methods, maximum validity duration of real time data, and basic concurrency constraint. They suggested execution engine model of the DREAM kernel, and implemented them [KK94]. Lee and Fishwick [LK98, LK99] suggest a real time simulation methodology called OOPM/RT, which is a multi-modeling methodology based on the idea of selecting the appropriate model among multiple model types through trading structural information for faster runtime while minimizing the loss of behavioral information. The selected optimal model guarantees delivering simulation results by the given amount of time. Garvey and Lesser proposed the Design-to-Time method [GHL93, GL93], which assumes that one has multiple methods for given tasks and tries to find a solution to a problem that uses all available resources to maximize solution quality within available time. They present an algorithm for finding an optimal solution to a real time problem under a task tree graph and task relationships. The algorithm generates all set of methods that can solve the problem and prunes those superseded by other sets of methods that generate greater or equal quality in equal or less time. In some methods [LK98, LK99, GHL93, GL93], the timing requirements are not specified by the simulation model, which are not applicable for real time simulation of *timed discrete event models*. Other methods [ITM90, AS91, KK94] can specify the timing requirement in model specification. However, they lack sound formalism in model specification, thus remaining a timing analysis as a difficult problem.

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 [JHong97], a real time extension of Zeigler's DEVS formalism [Zei84]; 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. RT-DEVS model execution environment is implemented to execute RT-DEVS models based on the proposed simulation methodology.

This paper is organized as follows. Section 2 presents the overview of our real time simulation framework. Section 3 describes the RT-DEVS formalism used in modeling of real time subsystems. The proposed methodology for real time simulation and its execution environment are given in section 4. Section 5 presents the application programming interfaces. An example system is illustrated in section 6. Conclusion is given in section 7.

2. PROPOSED REAL TIME SIMULATION METHODOLOGY

In this section, we discuss general characteristics about real-time simulation and the simulation procedure of our framework.

<< Insert Figure 1 here >>

Real-time simulation is accepted in many fields, such as training and control systems. In most cases, the term real time, when it comes to simulation, denotes that advancement of simulation time must occur in the real world time. This is a critical issue, since the simulation model represents an actual dynamic process as it occurs. Therefore, the real-time simulation framework must provide a means for simulation models to evolve

based on real world physical time. Based on the above definition, a real-time simulation of timed models is an evaluation process of models' timing behavior as physical time advances. In interactive, real-time simulation, there also exist messages between simulation models and real-world processes. This imposes an increased degree of complexity. To preserve the causality of simulation model, the simulator should process messages between models and real world processes on time. This is because a proper management of events between simulation models and real world processes is critical to evaluate a model's causality under real-time simulation. In our real time simulation framework, a target system is modeled by the RT-DEVS formalism. We implemented a real time kernel for RT-DEVS models, which employs an event driven scheduling policy to meet timing requirement of RT-DEVS simulation models.

<< Insert Figure 2 here >>

A typical simulation procedure within the framework is depicted in Figure 2. First of all, a target system is modeled with the RT-DEVS formalism that will be discussed in the next section. A modeler specifies atomic and coupled RT-DEVS models for the system to be modeled. If simulation models have interactions with external processes, IEE(Interacting External Environment) model is created. IEE represents the interacting external environment. The modeling step is followed by the scheduling feasibility analysis phase. For the analysis, a schedulability test graph is first extracted from the model and corresponding transient graphs are generated. Then, scheduling feasibility is tested on each transient graph. If the given RT-DEVS models are not feasible for real time execution in simulation, the modeler calibrates the models to satisfy the real time scheduling. In this step, a modeler often divides a time consuming activity function into smaller ones and modifies corresponding atomic models for the proposed real time scheduling to be feasible. Details of the scheduling feasibility analysis are given in [KCL01]. After the given simulation models pass the scheduling feasibility analysis, the real time scheduler for real time simulation now simulates them as follows. First, the scheduler flattens all coupled models in a recursive manner to minimize communication delays between component models. Then, it spawns model threads for corresponding atomic models and establishes communication channels between them. When real time simulation is done without any run-time exception due to timing violation, the modeler analyzes the simulation result. We now explain our real time scheduling algorithm for RT-DEVS models.

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 [JHong97], *RTAM*, is defined as:

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

, 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 $M = \{(s,e) \mid s \in S \text{ and } 0 \leq e \leq ta(s)|_{\max}\}$

$\delta_{int} : S \times I^{+}_{0,\infty} \rightarrow S$, an internal transition function

$\lambda : S \times I^{+}_{0,\infty} \rightarrow Y$, an output function

where $I^{+}_{0,\infty}$ is the non-negative integer numbers with ∞ adjoined. Note that $I^{+}_{0,\infty}$ denotes the set of time in terms of ticks that a model stays in a certain state.
$ta : S \rightarrow I^{+}_{0,\infty} \times I^{+}_{0,\infty}$, a time interval function,
Note that a time advance, *ta(s)*, for RT-DEVS is given by an interval $ta(s)|_{min} \leq ta(s) \leq ta(s)|_{max}$.
$\psi : S \rightarrow A$, an activity mapping function
A : a set of activities $A = \{a | t(a) \in I^{+}_{0\infty} and\ t(a) \leq ta|_{max}\} \cup \emptyset$

The RT-DEVS formalism replaces virtual time advance in the DEVS formalism[Zei84] by real time advance. The actual advance of simulation time is the sum of the real execution time of activity function and the time residing in the blocked state of the corresponding atomic model. A restriction of the specification is that a model cannot receive events when its current state executes an activity function associated with it.

Note that the RT-DEVS formalism defines a set of activities associated with a state, which is not defined in the DEVS formalism. Each activity is an abstraction of a task in a real system to be modeled. Within the RT-DEVS formalism, execution time for the task is specified by the time advance function defined for the state. Note also that a time advance, *ta(s)*, for RT-DEVS is given by an interval $ta(s)|_{min} \leq ta(s) \leq ta(s)|_{max}$ of integers, which is a real number in DEVS. A reason for such interval time advance is that a RT-DEVS simulator checks a specified time advance of a RT-DEVS model against a real time clock within the simulator during simulation. If the clock falls in the scheduled time interval correctness of the schedule would be verified. Integers in time advance specify computation time of an activity in terms of ticks generated by an underlying computing system. Such specification enables a modeler to specify the computation time adaptable to the implemented computing system.

A real time DEVS coupled model connects basic real time DEVS models together in order to form a new model. As in the DEVS formalism, a set of component models makes up a new coupled model. The real time coupled DEVS model, *RTCM*, is defined as:

$$RTCM = < D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} >$$

, where

D, I_i, Z_i, : same as the original
M_i : a component basic real time DEVS model

A coupled model within the RT-DEVS formalism is defined the same way as in the original DEVS formalism[Zei84] 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 cannot be happened in a real time simulation environment. In real time simulation with single processor, only one event at a time can be physically processed even if more than one event occurred from the external environment.

4. REAL TIME SIMULATION FRAMEWORK

In this section, the proposed real time simulation framework is presented in detail. First, we discuss our real time scheduling policy for RT-DEVS models.

4.1. EVENT DRIVEN SCHEDULING

Our real time simulation environment supports communication among RT-DEVS models and interactions with surrounding environment. In real time simulation, simulation models must evolve according to real world time. This is critical, because real world processes have no connection with simulation time. The only way to synchronize the time of a simulation model with that of a real world process is to use real world time as simulation time. By prioritizing the execution order of simulation models, the scheduler makes state transitions of simulation models occurred based on real world time. Such prioritizing is needed to prevent unacceptable delay, thus violating timing requirement of every simulation model. In the RT-DEVS formalism, the next scheduling time of a model is associated with t_N, which is given as the time advance value at the latest scheduling time. A RT-DEVS model with smaller t_N has higher priority and events are processed with the highest priority under our scheduling policy. One important observation is that t_N of a RT-DEVS model is dynamically changed during simulation execution. More specifically, atomic models are supposed to update their t_N whenever they receive internal or external events. When a model receives an event, the real time model scheduler sets the execution priority of that model to the highest one. This is because the corresponding model needs to make urgent rescheduling, which updates t_N. Therefore, priority reassignment is made whenever a model receives an event. That is why we refer our scheduling algorithm to as *event driven scheduling algorithm*. Table 1 summarizes the characteristics of scheduling algorithm.

Table 1. Characteristics of event driven scheduling

	Event Driven Scheduling
Task	$RTAM = < X, S, Y, \delta_{ext}, \delta_{int}, \lambda, ta, \psi, A >$ Aperiodic event driven task
Goal of Scheduling	Preservation of Causality
Factor of Priority Assignment	External/Internal events
Dependencies among tasks	Input/output dependencies influence the execution order

4.2. REAL TIME SIMULATION ENVIRONMENT

The RT-DEVS execution environment employs the event driven scheduling discussed in the previous section. Before it is implemented, a set of design goals is constructed. Such goals determine the internal structure of the RT-DEVS execution environment.

Several problems are to be considered to bridge the gap between simulation and execution. First, event processing time should be counted in real time execution. Thus, the underlying time base is changed to time intervals, and a couple of constraints are imposed on the modeler. These constraints are related to real time execution. To reduce a burden of constraints, we provide a scheduling feasibility analysis scheme. The schedulability criterion checks the feasibility of real time execution of given models under our real time execution environment. Second, timing constraints of simulation models must be satisfied. Guaranteeing timing behavior requires the system to be predictable. It is also desirable that the system attains a high degree of utilization while satisfying timing constraints of simulation models. Therefore, a predictable execution mechanism needs to be embedded in simulation engine. We implemented a real time kernel as a simulation engine, which supports scheduling facilities on the upper layer of the kernel. Our real time kernel has 10 millisecond clock granularity and context switching time of 5 milliseconds under Pentium II 266Mhz PC.

Portability is another goal of our execution environment. It is closely related to extensibility. Extensibility allows our execution environment to be easily enhanced, whereas portability enables it to be ported to a machine based on a different processor or configuration. Developing a portable software system that is easy to port is similar to writing portable code with certain guidelines. First, for the most part, source code must be written in prevailing language using portable library. Second, it is important to minimize and isolate, wherever possible, the part of code that interacts directly with an underlying platform. The RT-DEVS execution environment is largely written in the C and C++ languages that are most widely used programming language. Therefore, it can be easily ported to various platforms having any ANSI C++ compiler. However, some part of RT-DEVS micro kernel is hardware dependent, which is for context switching and synchronization. To improve its portability, we designed a layered architecture to isolate the hardware dependent part from the hardware independent one.

<< Insert Figure 3 here >>

The RT-DEVS execution environment has a layered architecture as shown in Figure 3. Each layer has its own functions based on which upper layers are built. As an execution engine for RT-DEVS atomic models, the RT-DEVS kernel was designed to be flexible linkage between RT-DEVS models and various hardware platforms. This is done by supporting real time threads with various activation and synchronization requirements. It also supports real time clock management, interrupt handling routines, and thread communication facilities. Generally, a target system consists of multiple models. Managing the execution of competing, real time model threads is the main purpose of RT-DEVS micro kernel. The RT-DEVS micro kernel provides basic services based on which a set of model threads are executed concurrently.

<< Insert Figure 4 here >>

The lower layer in the micro kernel has a thread context-switching routine and a real time alarm clock management routine. The low level context-switching routine provides facilities that can be used when the thread support layer creates a model thread or makes model thread switching. Model thread switching occurs when the model scheduler calls the priority function or sends a message. Interrupt handlers also use these facilities because they can trigger model thread switching. A thread is always in one of five states: executing, ready, suspended, or finished as depicted in Figure 4. When a thread is in suspended state, it is in blocked state while waiting for the completion of a service request. This happens when a model thread sends a message to other model thread. When the request is complete, the thread is moved to the ready state. When a thread is in finished state, it finished its job and returned from the entry routine. The real time alarm clock routine supplies facilities that produce current time and supports timer services. Timer services require a periodic timer interrupt from the hardware. Each thread has a built-in timer. It serves thread sleep requests and suspension timeouts in service call. A model thread relies on the built-in timer, when it executes an internal transition function.

An interrupt is a mechanism to provide instant response to an external event. When an interrupt occurs, the processor suspends the current executing thread and transfers control to the appropriate interrupt service routine (ISR). Interrupt handling often produces the data integrity problem, when ISRs need access to kernel services that require protection of data structures from simultaneous access. The simplest solution is to disable interrupts for the duration of the service. Because fast interrupt response is very important in real time systems, disabling interrupts to protect data structures is not desirable. The RT-DEVS micro kernel splits ISRs

into low and high-level components to solve the protection problem. While the low-level ISR executes in the context of a task it interrupted, the high-level ISR executes as an independent task. Therefore, the high-level ISR can be blocked temporarily to prevent its access to the kernel data structure that is already being accessed.

In our real time simulation environment, a model has its own input ports where it receives messages from surrounding environments as well as other models. These input ports consist of communication channels, which use communication services provided by the kernel. The RT-DEVS kernel provides queues, pipes, and mailboxes for communication purposes. The communication channel provides both synchronous mode and asynchronous mode transmission. In synchronous mode, a model is suspended until its message transmission is completed. In asynchronous mode, a model continues its execution after message transmission. Synchronization services provide access to kernel semaphores, event flags, and signals. Synchronization services are used when access to a resource must be controlled to ensure integrity of the resource. They are used to gain access to these controlled resources in various part of the kernel.

<< Insert Figure 5 here >>

The model scheduler executes each RT-DEVS models as a real time thread by using facilities provided by the thread support layer. The scheduler dynamically changes priorities of RT-DEVS models based on the proposed priority assignment policy. In this sense, the micro kernel can be viewed as a kernel-thread execution engine and the scheduler as a model execution engine. When simulation models in the RT-DEVS formalism are given, the model scheduler flattens all coupled models into atomic models to minimize communication delays. After communication channels among atomic models are established, real time simulation is initiated. Interactions between the model scheduler and the micro kernel will be illustrated in the next section.

4.3. MICRO-KERNEL SUPPORTED CONCURRENT MODEL EXECUTION

<< Insert Figure 6 here >>

The internal structure of a model thread is depicted in Figure 6. The model scheduler creates a thread for each simulation model after flattening all coupled models. A model thread receives messages from other models and processes in a surrounding environment. When it receives an external event, it is inserted into the event queue, which makes the model to execute **CME:when_rcv_(x,t)** procedure. When the internal clock alarms, a model thread execute **CME:when_rcv_(*,t)** procedure. External events are passed through communication facilities.

<< Insert Figure 7 here >>

Figure 7 shows interaction between the scheduler and the micro kernel. Each atomic model spawns its own thread to execute its characteristic functions concurrently with other model threads associated with corresponding atomic models. The real time scheduler performs the Concurrent Model Execution algorithm to be explained later. The algorithm changes an execution priority of each model by using the priority function P with argument $t_N|_{max}$ of each model, which is notified to the micro kernel. Then, the kernel-thread execution engine of the micro kernel switches CPU to a model thread based on the priority. By this way, a model with the highest priority is executed the first. As shown in Figure 5, each model has an associated independent pro-

gram, which concurrently executes internal transition and external transition of the model. Such an execution is different from that of conventional DEVS simulators in which the two transitions are executed in a sequential manner.

We now explain the program associated with each atomic RT-DEVS model, which consists of a main program **CME:main()** and two concurrent model execution algorithms, **CME:when_rcv_(*,t)** and **CME:when_rcv_(x,t)**, as abstract simulators.

<< Insert Figure 8 here >>

The 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 7 of the algorithm correspond to the function calls sequence [λ, δ_{int}, ta]. After updating its time advance value, it modifies its execution priority based on that by calling priority function P, which also updates priorities of other simulation models based on current t_L value. RT-DEVS model with smaller dt_N (= $t_{N|max}$ - t_L) value is given higher execution priority.

The 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 [δ_{ext}, ta]. As before, it modifies its execution priority based on updated dt_N value by calling priority function P.

Whenever the priority function is called, the kernel-thread engine switches CPU to the model thread with highest priority.(Line 8 in Figure 8 (a) and Line 7 in Figure 8 (b)) A context switching is also occurred, when a model performs output function.(Line 4 in Figure 8 (a)) This is because a model that has a pending external event is given highest priority. Therefore, a model with pending external event gets the CPU.

The 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.

To synchronize time between atomic models and an external environment, the model scheduler suspends an atomic model after finishing its activity until the time specified by its time advance function. If an external event occurs during that interval, the scheduler wakes up the corresponding atomic model.

The execution flow of our simulation framework is depicted in Figure 9. At first, user provides simulation models and coupling schemes between them. Then, the model scheduler flattens all coupled models so that the target system consists of atomic models only. It now creates corresponding model threads and schedules them under the event driven scheduling policy.

<< Insert Figure 9 here >>

5. APPLICATION PROGRAMMING INTERFACE

In this section, we explain the application programming interfaces that are used in the specification of RT-DEVS models. Before explaining the APIs of RT-DEVS execution environment, it worthwhile to summarize its conceptual basis. Table 2 shows the tuples required for simulation and execution. For a RT-DEVS atomic model, five tuples, from X to λ are commonly required. A time interval function ta is used for execution. The semantics of an external transition function δ_{ext} is slightly different only in the maximum time bound of an external event. An activity mapping function ψ and a set of activities A are newly devised for execution.

Table 2. Comparison of atomic model in DEVS/RT-DEVS formalism

Atomic model	DEVS	RT-DEVS
X	An input event set	
Y	An output event set	
S	A states set	
δ_{int}	$S \to S$	
λ	$S \to Y$	
δ_{ext}	$Q \times X \to S$,	$Q \times X \to S$,
	$Q = \{(s,e) \mid s \in S \text{ and } 0 \le e \le ta(s) \}$	$Q = \{(s,e) \mid s \in S \text{ and } 0 \le e \le ta(s)\|_{\max} \}$
ta	$S \to R^{+}_{0,\infty}$	$S \to I^{+}_{0,\infty} \times I^{+}_{0,\infty}$
ψ		$S \to A$
A		$A = \{a \mid t(a) \in I^{+}_{0,\infty} \text{ and } t(a) \le ta\|_{\max}\} \cup \emptyset$

Table 3 summarizes the programming interfaces of RT-DEVS for modelers. For some tuples such as X and M, there are operators as well as functions. Operators are more convenient specification method to modeler. X, Y, S, M, and C objects correspond to X, Y, S, M_i and $I_i/Z_{i,j}$ defined in the RT-DEVS formalism, respectively. To define the elements of a set, the = and , operators are used. For example, the statement X = "x1", "x2" means that there are two input ports, named x1 and x2. The characteristic functions of the RT-DEVS formalisms have a one-to-one correspondence with implementation. For example, an external function is defined as δ_{ext} : $Q \times X \to S$, where $Q = \{(s,e) \mid s \in S \text{ and } 0 \le e \le ta(s)|_{\max} \}$. Thus, the API of the external function is **void extTransfn** (**Time** e, **XPortSet** &X). The mapping of two parameters, e and X, is straightforward. However, there is no parameter for a state set S, Intuitively, atomic models and coupled models are mapped into respective classes. Then, a state set can be thought of as a member variable in one such class. Although a state set could be given as a parameter of the function, our approach simplifies use of APIs.

Table 3. Application programming interfaces of RT-DEVS

Atomic model	Operator	Function
X	X = “x1”, “x2”	addInports(2, “x1”, “x2”)
	X[“x1”]	X.from(“x1”)
	X[“x1”] >> v	v = (Type*)&X.receive(“x1”)
Y	Y = “y1”	addOutport(“y1”)
	Y[“y1”] << *v	Y.send(“Y1”, *v)
S	S = new State(“phase”, phase)	addState(new State(“phase”, phase))
	phase = IDLE	phase = IDLE
δ_{ext}	void extTransfn(Time e, XPortSet& X)	
δ_{int}	void intTransfn(void)	
λ	void outputfn(YPortSet& Y)	
ta	TimeInternval timeIntervalfn(void)	
ψ,A	void activityfn(Time e, XPortSet& X)	

Coupled model	Operator	Function
X	X = “x1”, “x2”	addInports(2, “x1”, “x2”)
Y	Y = “y1”	addOutport(“y1”)
D, M_i	M = new M1(“M1”), M2	addModels(2, new M1(“M1”), M2)
I_i, $Z_{i,j}$	C = (X[“X1”] >> M[“M1”].X[“x1”])	addCoupling(this, “x1”, M[“M1”], “x1”)

6. EXPERIMENT: ALTERNATING BIT PROTOCOL

To show the RT-DEVS based simulation within the proposed framework, design of Alternating Bit Protocol (ABP) is to be considered. ABP is a communication protocol for secure transmission of the messages from a source to a destination. Generally the existing communication media is not perfect. Thus, there always exists such possibility that messages generated from the sender might be lost, duplicated or corrupted before arriving to the destination. Thus a secure communication protocol is needed to ensure a correct transfer of the messages between the two entities. This section exemplifies implementation of ABP. We demonstrate real time, interactive simulation by executing simulation models interacting with a software process in a remote host. Correctness of our real time framework is verified by observing communication between simulation models and remote software process in real time.

6.1. INFORMAL MODEL DESCRIPTION AND RT-DEVS MODELING

<< Insert Figure 10 here >>

Figure 10 shows an overall system model of ABP. The *Sender* and the *Receiver* exchange messages through local area network(LAN). The *Sender* delivers a message, and then the *Receiver* acknowledges it. When a message is sent, *Receiver* process responds in 20 ticks. To determine whether the message is lost or not, both the *Sender* and the *Receiver* are notified by a timeout event if no message is arrived in a specified time interval. Once a timeout event is notified, retransmission is made assuming that a message transmitted previously has been lost. Messages are sent tagged with a bit 0 and 1 alternatively, and also the acknowledgements are constituted of the bits. The *Receiver* is implemented as a real software process in a remote host.

Gen/*Sender*/*Timer* models are implemented as RT-DEVS model in our real time simulation framework. The model-environment interface contains a driver that directly communicates with network interface card and forwards messages bi-directionally. The *Sender* sends a message using functions provided by the network card driver and receives a message that the interrupt handler in the driver delivers.

<< Insert Figure 11 here >>

Figure 11 shows RT-EVS models for scheduling feasibility analysis. IEE represents only I/O behavior of a remote process by hiding details of the process. Therefore, IEE has no activity functions. When the scheduling feasibility is analyzed, external processes are not directly concerned. Instead, IEE is concerned on behalf of them.

Each RT-DEVS model has the following state diagram. The notation *?event* represents receiving an event and *!event* represents sending an event. Time interval $@(ta|_{min}, ta|_{max})$ is given in ticks, where one tick is 100 milliseconds.

<< Insert Figure 12 here >>

Initially, *Sender* waits for a message from *Gen(erator)* at *accept0* state. Upon receiving a message, it moves to *send0* state where it performs activity function *packet*, which makes a message packet with control bit 0. Then, it transmits the message packet and moves to *wait0* state. In *wait0* state, it waits for an ack(nowledge) message with control bit 0 from the *Receiver*. Upon receiving an ack message with control bit 0, it moves to *ack0* state7 where it sends done message to Gen. When an ack message does not come in due time or an ack message with control bit 1 is received, it moves to *resend0* state, where it retransmit the message. *Sender* repeats this sequence with alternating control bit 0/1.

<< Insert Figure 13 here >>

Gen has an initial *send* state where it generates a data with an activity function. It sends the message to *Sender* and moves to *wait* state. In *wait* state, it waits for done message. Upon receiving done message, it generates a new message. *Gen* repeats this sequence. *Timer* waits for a message in its initial *wait* state. Upon receiving a message, it moves to *alarm* state. Then, it expires and sends timeout message.

6.2. SCHEDULING FEASIBILITY

<< Insert Figure 14 here >>

<< Insert Figure 15 here >>

Figure 14 shows the state transition diagram of the given system. To check the scheduling feasibility of given simulation models, we build a schedulability test graph from Figure 14. We assume that processing time for each activity function is 5 ticks at most. The schedulability test graph and transient graphs are depicted in Figure 15. After applying the scheduling feasibility criterion to all transient graphs, we conclude that the given simulation model can be simulated in real time under our real time simulation environment.

6.3. RT-DEVS MODEL IMPLEMENTATION

When a modeler creates a real time atomic model, he/she defines a new model class that inherits from *AtomicModel* class. To show one to one correspondence between a model specification and an implementation code, the *Sender* model specification and its implementation are presented.

Formal definition of *Sender* model depicted in Figure 12 is as follows.

<< Insert Figure 16 here >>

The following code is a part of C++ code that corresponds to δ_{ext}.

<< Insert Figure 17 here >>

Line 7, 8 describe the external state transition at of δ_{ext}-(1) of Figure 16. And Line 9, 10 describe δ_{ext}-(2) of Figure 16. The C++ code is written with APIs of RT-DEVS shown in Table 3.

The following code is a part of C++ code that corresponds to δ_{int}. λ, *ta*, ψ.

<< Insert Figure 18 here >>

As shown above, a modeler defines member functions that have one to one correspondence with the formal definition of a model.

6.4. REAL TIME EXECUTION = REAL TIME SIMULATION

The RT-DEVS simulation framework supports interaction between RT-DEVS models and an external environment, which can be either hardware, software or a human operator. The presented experiment demonstrates this capability by implementing *Receiver* model as a software process in a remote host. *Trans*/*Ack* models originally represent a LAN environment. Therefore, the LAN environment is used as it is. *Gen*/*Sender*/*Timer* models are implemented as RT-DEVS models in our real time simulation framework.

We verified correctness of our real time simulator by observing communication between *Sender* and *Receiver* in real time. Figure 19 shows the state trajectories of the target system in real time simulation. The time at which event occurs is bounded within the time intervals specified in RT-DEVS models. This shows that the event time measured during real time simulation coincides with the time interval in model specification. Therefore, the correctness of real time simulation is confirmed.

<< Insert Figure 19 here >>

7. CONCLUSION

We proposed a real time simulation methodology by extending the DEVS formalism and developed an asso-

ciated simulation environment by employing concurrent execution of RT-DEVS models. The RT-DEVS simulation methodology and the simulation environment provide the uniform and flexible representation of real time systems. In the proposed real time simulation framework, multiple models are scheduled based on events they received. To meet timing constraints during execution of simulation models, a real time simulation kernel for RT-DEVS models is implemented and it employs an event-driven scheduling policy to execute RT-DEVS models concurrently. The simulation kernel has layered architecture to provide portability and guarantees predictable execution time. The kernel has 10 millisecond clock granularity and context switching time of 5 milliseconds under Pentium II 266Mhz PC. A simple example of real time, interactive simulation demonstrated an overall procedure of the proposed real time simulation methodology. The correctness of our simulation framework is verified by observing events between simulation models and external real world process.

Our current work is to extend the analysis method that checks upon the scheduling feasibility of given simulation models. The limitation of the current methodology is that only simulation models with deterministic behavior in event time can be analyzed for the scheduling feasibility. Analysis of non-deterministic event time models is much harder and remains as future work.

References

[AS91] A. Attoui and M. Schenider, “An object oriented model for parallel and reactive systems.” *Proceedings of IEEE CS 12th Real time Systems Symposium*, 1991, pp 84-93.

[BW94] Alan Burns and Andy Wellings, “HRT-HOOD: A structured design method for hard real time systems.” *Real time Systems*, vol. 6, 1994, pp 73-114.

[CK98] Seong Myun Cho and Tag Gon Kim, “Real time DEVS Simulation: Concurrent, Time-Selective Execution of Combined RT-DEVS Model and Interactive Environment.” *The proceedings of the summer computer simulation conference*, July 1998, pp 410-415.

[CP97] 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, pp. 572-598.

[GHL93] Alan Garvey, Marty Humphrey and Victor Lesser, "Task Interdependencies in Design-to-time Real time Scheduling." *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 1993.

[GFS94] Kaushik Ghosh, Richard M. Fujimoto and Karsten Schwan, “PORTS: a parallel, optimistic, real time simulator.” *ACM SIGSIM Simulation Digest*, vol. 24, no. 1, July 1994, pp 24-31.

[GL93] Alan Garvey and Victor Lesser, “Design-to-time real time scheduling.” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 5, 1993, pp 1491-1502.

[ITM92] Yutaka Ishikawa, Hideyuki Tokuda and Clifford Mercer, “An object-oriented real time programming language.” *IEEE Computer*, vol. 9, no. 5, 1992, pp 66-73.

[ITM90] Yutaka Ishikawa, Hideyuki Tokuda and Clifford Mercer, “Object-oriented real time language design:

Constructs for timing constraints," *Technical Report CMU-CS-90-111*, Carnegie Mellon University, 1990.

[JHong97] Joon Sung Hong, Hae Sang Song, Tag Gon Kim and Kyu Ho Park, "A Real time Discrete Event System Specification Formalism for Seamless Real time Software Development." *Discrete Event Dynamic Systems: Theory and Applications*, 7, 1997, pp 355-375.

[KCL01] Tag Gon Kim, Seong Myun Cho and Wan Bok Lee, "DEVS Framework for Systems Development: Unified Specification for Logical Analysis, Performance Evaluation and Implementation." *Discrete Event Modeling and Simulation: A Tapestry of Systems and AI-based Theories and Methodologies, A Tribute to the 60th Birthday of Bernard P. Zeigler* (H.S. Sarjoughian and F.E. Cellier ed.), Springer Verlag, 2001, pp 31-166.

[Kim91] Tag Gon Kim, "Hierarchical development of model classes in the DEVS-Scheme simulation environment." *Expert Systems With Applications*, 3, 1991, pp 343-351.

[Kim94] K. H. Kim, "A utopian view of future object-oriented real time dependable computer systems." *International Workshop on Real Time Computing Systems and Applications*, December 1994, pp 59-69.

[Kim97] K. H. Kim, "Object structures for real time systems and simulators." *IEEE Computer*, vol. 30, no. 8, August 1997, pp 62-70.

[KK94] K. H. Kim and Hermann Kopetz, "A real time object model RTO.k and an experimental investigation for its potentials." *International Computer Software and Applications Conferences*, November 1994, pp 392-402.

[KS97] C.M. Krishna and Kang G. Shin, *REAL TIME SYSTEMS.* McGraw-Hill Book, Oxford, 1994.

[LK98] Lee, K. and P. A. Fishwick, "Generation of multimodels and selection of the optimal model for real time simulation." *SPIE Aerosense Conference*, April 1998.

[LK99] Kangsun Lee and Paul A. Fishwick, "OOPM/RT: A multimodeling methodology for real time simulation." *ACM Transactions on Modeling and Computer Simulation*, vol. 9, no. 2, April 1999, pp 141-170.

[MO94] Leo Motus and Michael G. Rodd, *Timing analysis of real time software.* Redwood Books, Oxford, 1994.

[MT90] Clifford W. Mercer and Hydeyuki Tokuda. "The ARTS real time object model." *Proceedings of IEEE CS 11th Real time Systems Symposium*, 1990, pp 2-10.

[Par93] Sung Bong Park, DEVSIM++: A semantic based tool for object-oriented modeling of discrete event systems. Master's thesis, Dept. of Electrical Eng., KAIST, 1993.

[RAMA94] Ramamrithan, K., and Stankovic, J. "Scheduling Algorithms and Operating Systems Support for Real time Systems." *Proceedings of the IEEE*, January 1994.

[RP94] Alan Burns and Andy Wellings, *Real time Systems and Programming Languages.* Addison-Wesley, 1994.

[Zei84] Bernard P. Zeigler, *Multifacetted Modelling and Discrete-Event Simulation.* Academic Press, New York, 1984.

[ZK93] Bernard P. Zeigler and Jinwoo Kim, "Extending the DEVS-scheme knowledge-based simulation environment for real time event-based control." *IEEE Transactions on Robotics and Automation*, vol. 9, no. 3, 1993, pp. 351- 356.

FIGURES

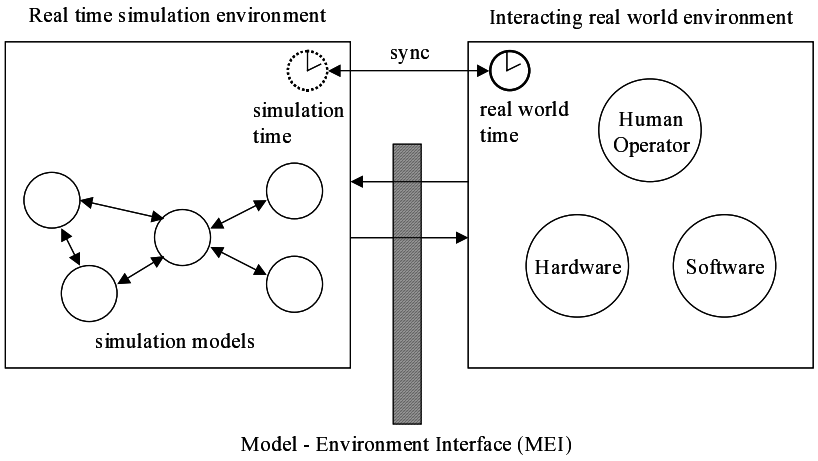


Figure 1. Real time, interactive simulation

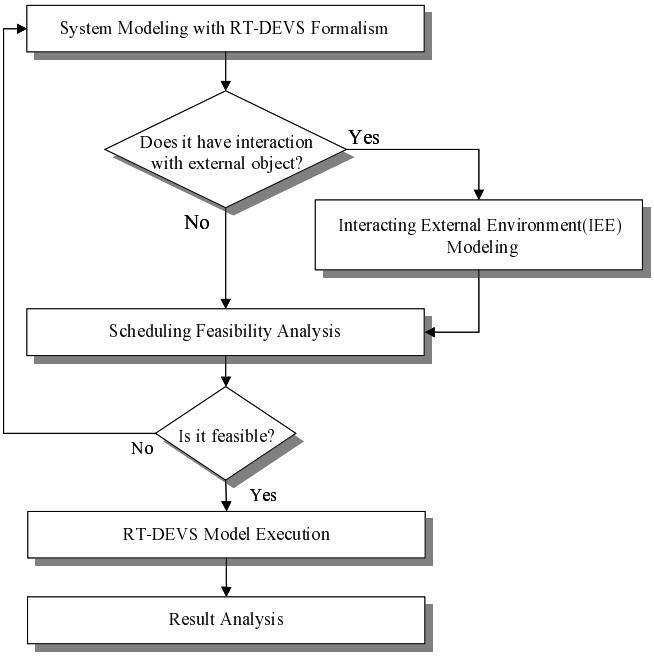


Figure 2. Proposed real time simulation methodology

IEE model is a model of MEI and interacting external environment

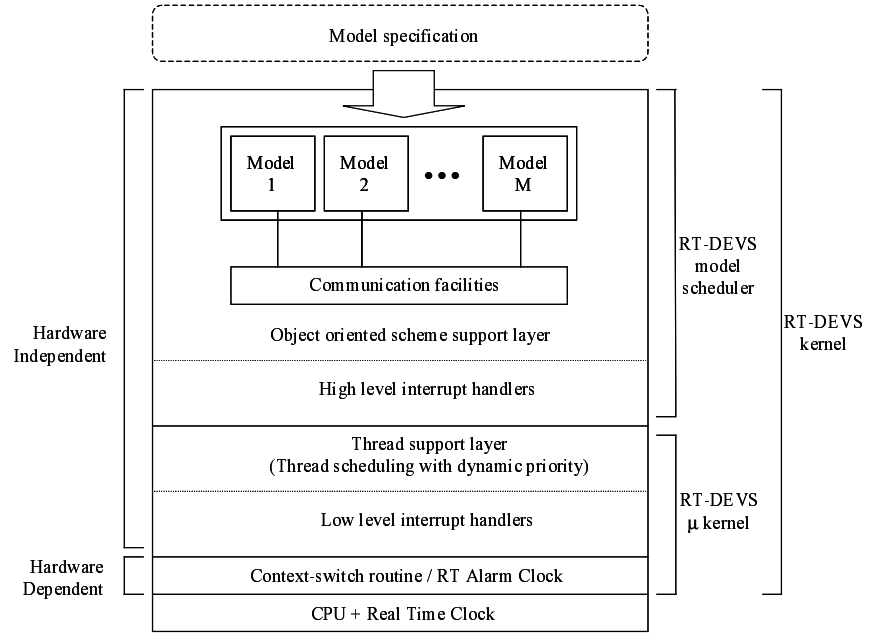
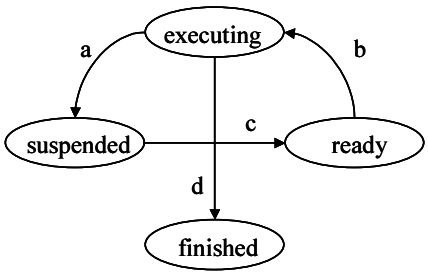


Figure 3. Architecture of RT-DEVS simulation environment



a : service request
b : scheduling
c : service completion
d : processing finished

Figure 4. State transition of model thread

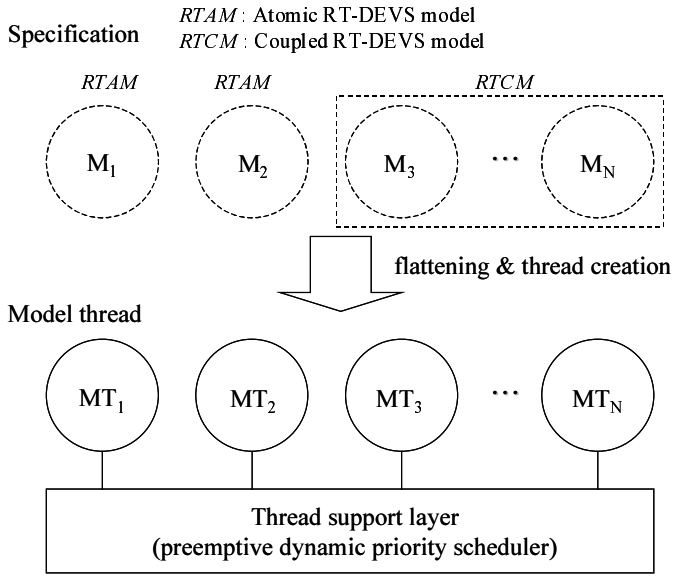


Figure 5. Model specification to model thread translation

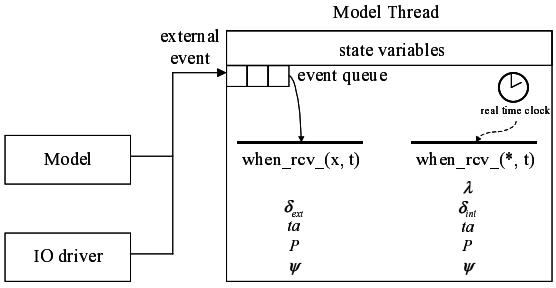


Figure 6. Event handling within model thread

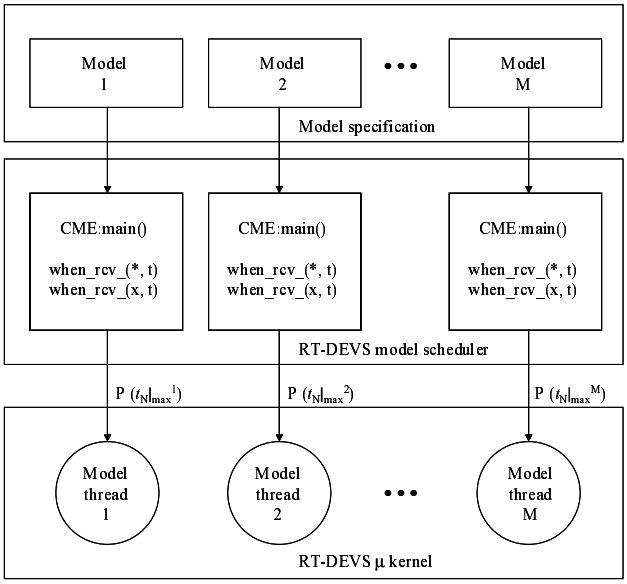


Figure 7. Priority mapping $P(t_N|_{max})$ from scheduler to micro kernel

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

(a)

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. $t_N := [t_L + ta(s)|_{min}, t_L + ta(s)|_{max}]$;
7. $P(t_N|_{max})$
8. $\psi(s)$
9. **else**
10. error;
11. **end if**

(b)

```
1. CME:main():
2.   s = s0                          /* initialize */
3.   tN := [ta(s0)|min, ta(s0)|max];
4.   concur_forever for each RT-DEVS model
                                     /* main loop */
5.        wait for an event
6.        if an external event then
7.             when_rcv_(x, t);
8.        else if an internal time out event then
9.             when_rcv_(*, t);
10.       end if
11.  end concur_forever
```

(c)

Figure 8. Concurrent model execution(CME) algorithms
(a) CME: when_rcv(*, t)
(b) CME: when_rcv(x, t)
(c) CME: main()

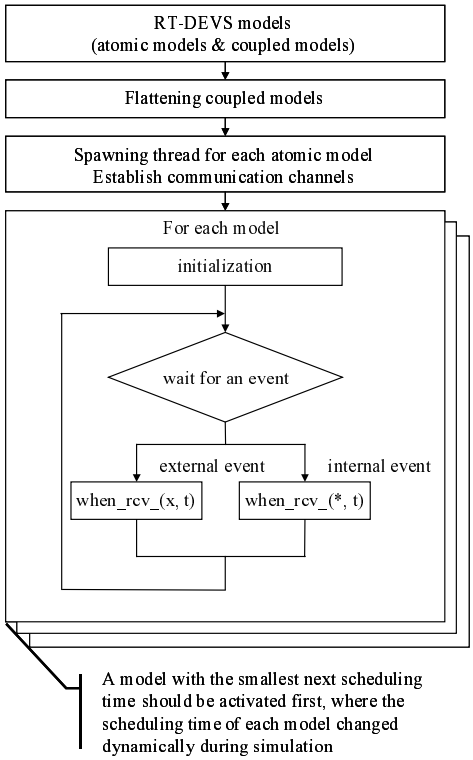


Figure 9. Execution flow of RT-DEVS models simulation

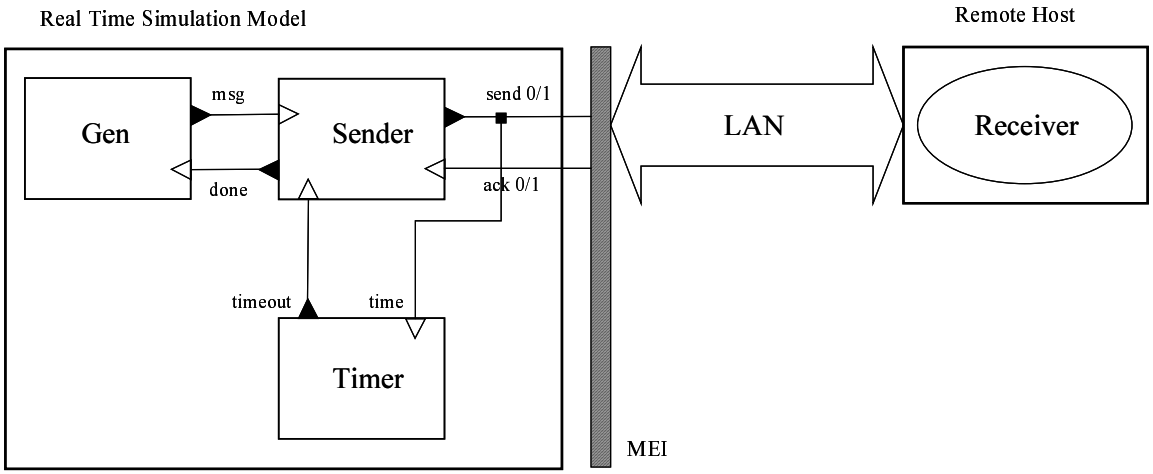


Figure 10. ABP system consisting of RT-DEVS models and a remote software process

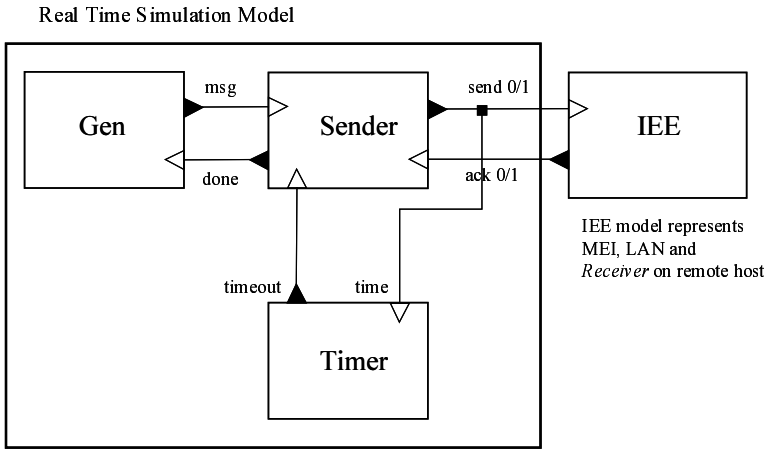


Figure 11. RT-DEVS model for scheduling feasibility analysis

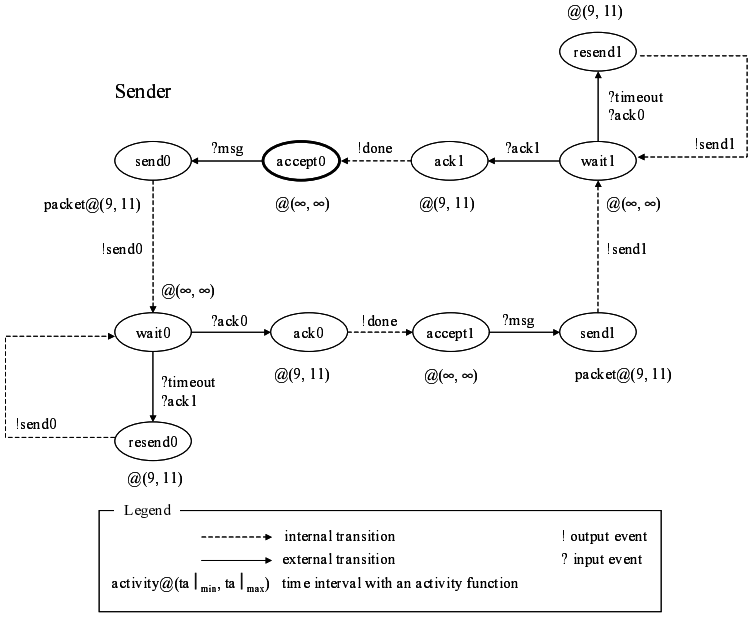


Figure 12. Sender model

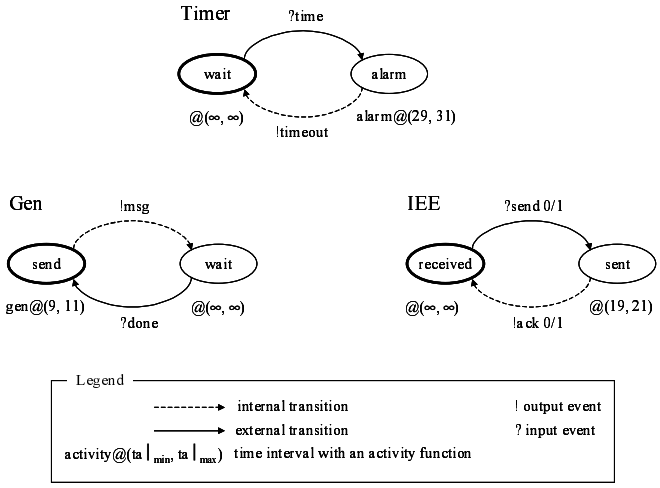


Figure 13. Timer/Gen/IEE model

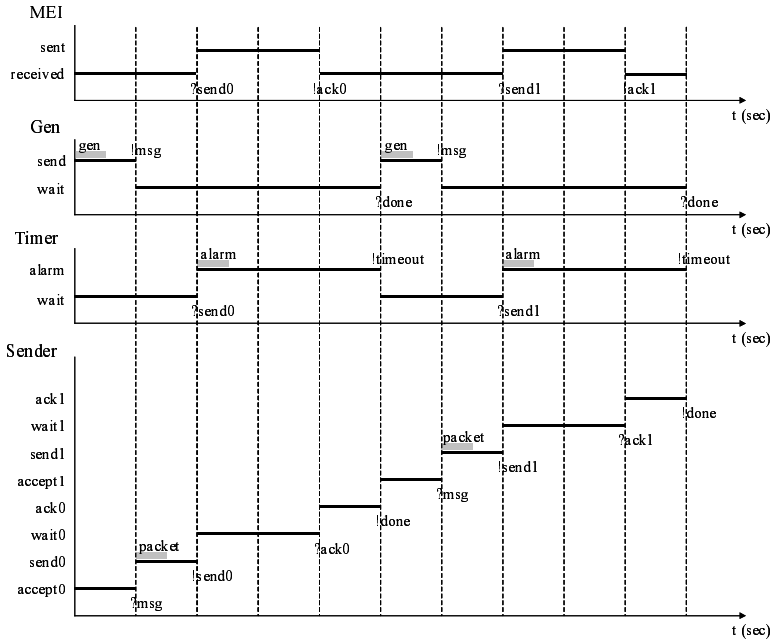


Figure 14. State transition diagrams of three RT-DEVS models and MEI

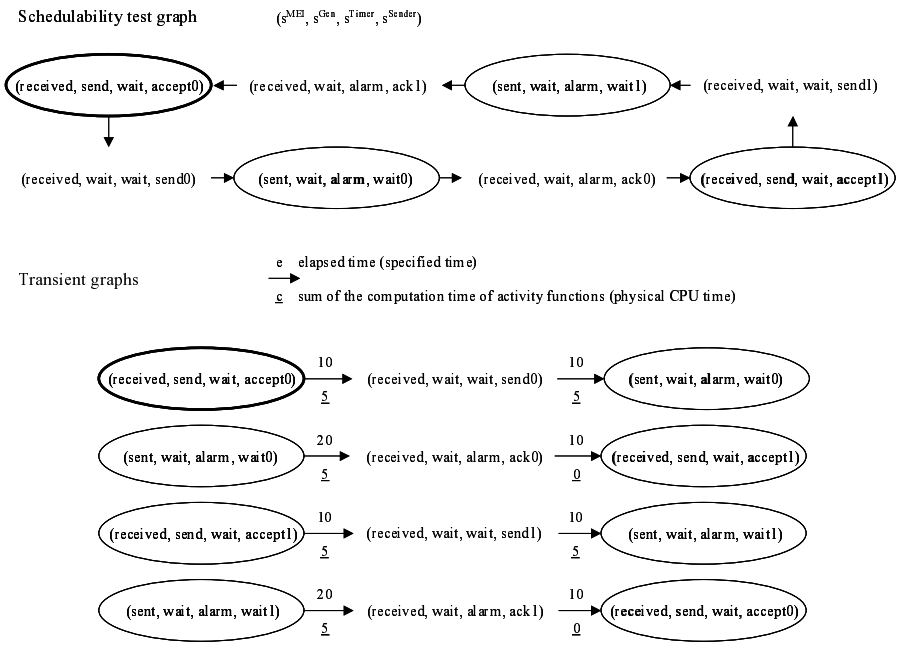


Figure 15. Schedulability test graph of Figure 14 and transient graphs

Sender = < X, S, Y, δ_{ext}, δ_{int}, λ, ta, ψ, A >

X = { msg, ack0, ack1, timeout }
Y = { done, send0, send1 }
S = { ACCEPT0, SEND0, WAIT0, RESEND0, ACK0, ACCEPT1, SEND1, WAIT1, RESEND1, ACK1 }

$\delta_{ext} : Q \times X \to S$

ACCEPT0 × e* × msg → SEND0	δ_{ext}-(1)
ACCEPT1 × e* × msg → SEND1	δ_{ext}-(2)
WAIT0 × e* × ack0 → ACK0	δ_{ext}-(3)
WAIT1 × e* × ack0 → RESEND1	δ_{ext}-(4)
WAIT0 × e* × ack1 → RESEND0	δ_{ext}-(5)
WAIT1 × e* × ack1 → ACK1	δ_{ext}-(6)

where, e* = { e | $0 \le e \le ta(s)|_{\max}$ }

$\delta_{int} : S \to S$

SEND0 → WAIT0	δ_{int}-(1)
RESEND0 → WAIT0	δ_{int}-(2)
ACK0 → ACCEPT1	δ_{int}-(3)
SEND1 → WAIT1	δ_{int}-(4)
RESEND1 → WAIT1	δ_{int}-(5)
ACK1 → ACCEPT0	δ_{int}-(6)

$\lambda : S \to Y$

SEND0 → send0	λ-(1)
SEND1 → send1	λ-(2)
RESEND0 → send0	λ-(3)
RESEND1 → send1	λ-(4)
ACK0→ done	λ-(5)
ACK1→ done	λ-(6)

$\psi : S \to A$

SEND0 → packet0	ψ-(1)
SEND1 → packet1	ψ-(2)

where, A = { packet0, packet1 }

$ta : S \to I^{+}_{0,\infty} \times I^{+}_{0,\infty}$

SEND0 → (9, 11)	ta-(1)
SEND1 → (9, 11)	ta-(2)
RESEND0 → (9, 11)	ta-(3)
RESEND1 → (9, 11)	ta-(4)
ACK0→ (9, 11)	ta-(5)
ACK1→ (9, 11)	ta-(6)
ACCEPT0 → (∞, ∞)	ta-(7)
ACCEPT1 → (∞, ∞)	ta-(8)
WAIT0 → (∞, ∞)	ta-(9)
WAIT1 → (∞, ∞)	ta-(10)

Figure 16. RT-DEVS specification of *Sender*

```
1. void
2. Sender::extTransfn(Time e, XPortSet& X)
3. {
4.     if (X["msg"])
5.     {
6.         msg = (String&)X.receive();
7.         if (phase == Sender::ACCEPT0)            // δext-(1) of Figure 16.
8.             phase = Sender::SEND0;
9.         else if (phase == Sender::ACCEPT1)       // δext-(2) of Figure 16.
10.            phase = Sender::SEND1;
11.    }
12. // omitted
```

13. }

Figure 17. C++ implementation of δ_{ext} of *Sender*

```
void
Sender::intTransfn(void)
{
    switch (phase)
    {
       Sender::SEND0:      // δint-(1) of Figure 16.
       Sender::RESEND0: // δint-(2) of Figure 16.
          phase = Sender::WAIT0;
          return;
 // omitted
    }
}

TimeInterval
Sender::timeIntervalfn(void)
{
    switch (phase)
    {
       case Sender::SEND0:      // ta-(1) of Figure 16.
       case Sender::SEND1:      // ta-(2) of Figure 16.
       case Sender::RESEND0: // ta-(2) of Figure 16.
       case Sender::RESEND1: // ta-(2) of Figure 16.
          return TimeInterval(9, 11);
   // omitted
   }
}
```

```
void
Sender::outputfn(YPortSet& Y)
{
   switch (phase)
   {
      case Sender::SEND0: // λ-(1) of Figure 16.
      case Sender::SEND1: // λ-(2) of Figure 16.
         Y["send"] << msg;
         break;
// omitted
   }
}

void
Sender::activityfn(void)
{
   switch (phase)
   {
      case Sender::SEND0: // ψ-(1) of Figure 16.
         ::packet('0', msg);
         break;
      case Sender::SEND1: // ψ-(2) of Figure 16.
         ::packet('1', msg);
         break;
   }
}
```

Figure 18. C++ implementation of δ_{int}. λ, *ta*, ψ of *Sender*

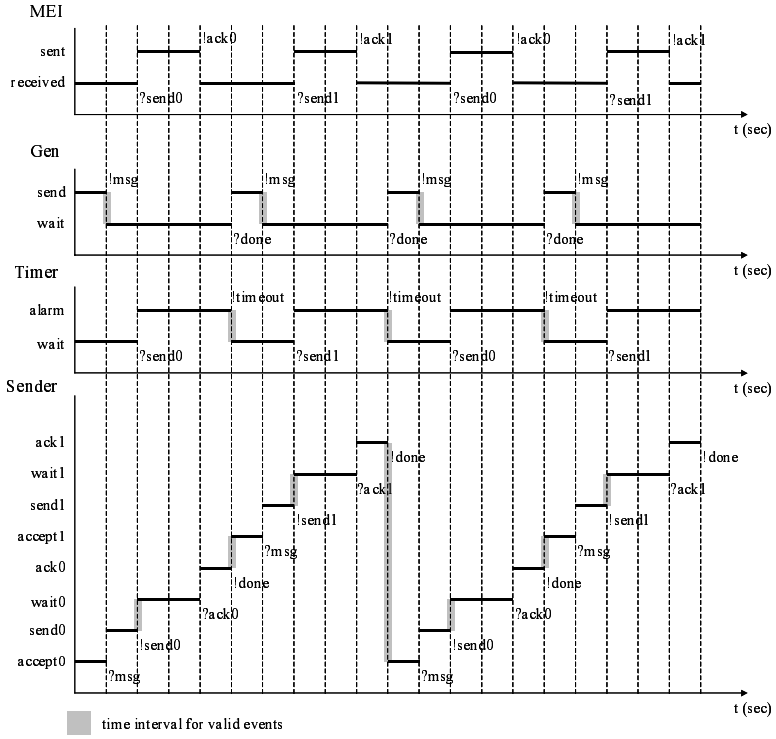


Figure 19. Real time simulation results