Studying Basic Behavior of Real-Time Systems Using Object-Oriented Simulation

José M. Garrido
Department of Computer Science and Information Systems
Kennesaw State University
Kennesaw, GA 30144
jgarrido@kennesaw.edu

Abstract

A practical approach to object-oriented modeling and simulation is presented to help improve the study and understanding of the general behavior of real-time systems. The modeling approach is based on Communicating Real-Time Machines (CRSM). The object-oriented simulation approach is based on process interaction with discrete-event simulation using PsimJ2 (a set of Java classes).

This modeling and simulation approach models timing constraints, concurrency, and synchronization, in addition to more basic expected behavior. The Train-Gate System is presented as a case study that shows the modeling and the simulation aspects of this study. The results of the simulation display the reaction times, response times, throughput, and the deadlines missed.

1 Introduction

Real-time systems respond (or react) in a timely fashion to variety of events from its environment. A real-time system is usually one that maintains an ongoing interaction with its environment. In this interaction, a real-time system is bounded by the specified time windows for its output. In other words, the response from the system cannot occur too early, or too late, given a set of inputs from the environment.

A real-time system can be considered a control system, which has the goal of controlling part of the environment (sometimes called the controlled subsystem). In simpler cases, a real-time system is a supervisory system if it only takes measures (input signals and data) from the environment at specified time intervals and/or when specified periodic events occur. In this case, the response of the real-time system is limited to record such inputs from the environment.
An embedded system, can be considered a real-time system that is part of a larger system, it is very specialized to suit a specific purpose, and does not usually interact with human users or operators. Real-time systems normally include a combination of hardware and software. The range of systems span from small embedded systems to very large systems with wide area networks. Figure 1 illustrates the general architecture of a real-time system and its environment.

Real-time computing is very different from general-purpose computing. In the first, not only are there strict timing constraints but frequently a very limited set of low-capacity (hardware) resources such as: memory, type of processor, disk space, etc. Customized software must interface with a small number of hardware devices. The software is usually developed in a desktop or workstation and targeted to a specific set of processors and devices.

In general, real-time systems must be extremely reliable, robust, dependable, and fault tolerant. They are much more difficult to develop than non-real-time systems and the cost of development is much higher. The general principles of real-time systems are discussed in several texts, among others are [1, 2, 3].

2 General Characteristics of Real-Time Systems

The general characteristics of real-time systems are:

- Timeliness - the system must perform operations in timely manner;
- Reactiveness - the system continuously responds to (random) events;
Concurrency - multiple simultaneous activities are carried out; each one responds to a different set of events;

Distribution - real-time processes that cooperate are located in multiple computing sites.

Real-time systems are non-terminating because of the reactiveness characteristics. They are also non-deterministic since they interact with an environment whose behavior is of an unpredictable nature. The input events from the environment have no specific order of occurrence.

3 Modeling Of Real-Time Systems

The basic modeling approach used for real-time systems is object-oriented. A real-time system is modeled as a set of interacting concurrent real-time processes. Each process represents an active object, which is modeled as a Java thread. These processes interact among themselves and with the environment. The communication among real-time processes is normally defined by message passing, which can be synchronous or asynchronous. The resources used in the system are modeled as passive objects. The general model for real-time systems adopted here is based on Communicating Real-Time Machines (CRSM) presented in [4, 3]. In CRSM all communications are synchronous and via unidirectional channels that connect two processes.

Time is an important consideration in real-time systems. To improve the performance of these systems two time measures must be reduced:

- Service time - the period taken to compute a response to a given input;
- Latency - the period from the instance of occurrence of an input to the instance at which the system starts servicing the input.

The response time of the system for a given input is the sum of the two periods listed above. The response time must be shorter than the deadline specified for this type of input. The reaction time is the latency for a given input.

The real-time requirements of a system involve specifying the deadlines for each type of input. Because of concurrent real-time processes and variable delays, satisfying the specification is very complicated. For hard real-time systems, missing a single deadline is taken as incorrect (and unacceptable). For soft real-time systems, a few selected deadlines may be missed occasionally.

In CRSM, timing constraints are specified in the transitions between states. A timing constraint is a pair of time expressions with the following meaning:

1. The values of the expressions give the lower and upper bounds on the duration of an internal activity (a computation).
2. The values on the expressions give the earliest and latest times that an I/O activity (communication) can occur relative to the starting time of the current state.
The general form of a transition is:

\[ G \rightarrow act[tc] \]

\( G \) is the guard of the transition, \( act \) is the associated activity, and \( tc \) is the timing constraint.

The model of real-time systems discussed here has been developed with the maximum parallelism assumption; i.e. every process executes in its own processor. No scheduling issues are discussed. A similar graphical notation for modeling real-time systems in UML appear in several recent books, such as [1].

4 Train Gate-Crossing System

This system consists of a one-directional railway track that crosses a road. There is a gate at the crossing, which is lowered and raised under computer control. A short distance before the crossing, there is a sensor that can detect a train is approaching the crossing area. Another sensor is located a short distance after the crossing point, which detects trains leaving the area.

Trains arrive with a mean inter-arrival period given by \( a \), trains take an average of \( b \) time units to cross the gate and leave. The gate takes a maximum of \( z \) time units to close or open. There are some delays imposed for communication. The system is specified to ensure that the gate closes before the train reaches the crossing. Any number of trains can be in the area at once. The physical safety requirement of the system is that the gate is closed whenever there are trains in the area. The physical liveness requirement is to keep the gate open if there are no trains in the area.

The response time (or period) is the time interval that elapses from the instance a train arrives to the instance the gate completes closing. The reaction time is the interval that elapses from the instance the train arrives (input event) until the gate starts to close.

The system diagram in Figure 2 shows all the active objects (processes) and the communication channels defined. The figure shows the main components (objects and channels) of the system. The following processes are defined:
Figure 3: The Entry sensor process.

Figure 4: The Exit sensor process.

Figure 5: The Monitor process.
Figure 6: The Controller process.

- The train entering sensor, $Se$.
- The train leaving sensor, $Sl$.
- The gate, $G$.
- The monitor, $M$, that keeps track of the number of trains in the area.
- The controller $C$ that controls the gate.

Process $Se$ sends 'train-in' signals via channel $trin$ according to some train inter-arrival time expression given by $a$, to the monitor process, $M$. $Se$ will also send signals via channel $tr$ to process $Sl$ so that it can expect a train exit in the future. After receiving a signal from $Se$, process $Sl$ is prepared to sense when a train leaves; when this occurs, it sends the signal $trout$ to process $M$.

Process $M$ monitors the state of the area in order to send appropriate 'open' signal via channel $Cont_{og}$ and 'close' signal via channel $Cont_{cg}$ to the controller process-a required behavior. The controller process $C$ sends its signals to the gate process $G$ for opening or closing. The controller process, $C$, controls the gate with the 'opengate' via channel $og$ and 'closegate' via channel $cg$ as commands to the gate process.

The controller process is the most complex one in the system; its functionality is described as follows: After receiving a signal from the monitor process, $M$, to close the gate, the controller process attempts to communicate with the gate process, $G$. There is a deadline imposed when attempting communication with the gate process. There is another deadline imposed on the gate process, when the interval for the gate to close exceeds $z$ time units.

In a similar manner, the controller process waits for the monitor process with a signal to open the gate, then it sends a signal to the gate for start to open. The main difference is that while the controller process is waiting for the opening of the gate, it may receive a signal from the monitor for immediate closing of the
gate. In this case, the controller process must interrupt the activity of the gate process, then signal the gate for closing.

The gate process, G, ignores $og$ commands when open and $cg$ commands when closed. A $cg$ command causes the gate to physically close; taking $z$ time units of time, and an $og$ command opens the gate. Opening also takes $z$ time units of time. Figures 3 through 7 show the CRSM diagrams the processes described. When the gate has completely closed (or opened), it sends the $gok$ signal to the controller process.

5 Timeouts in Communications

A process may take longer than the deadline set for carrying out a communication activity. In this circumstance, a timer process may be used to interrupt a process. A timer process can also be used to reactivate a process that is attempting to communicate with another process.

Once interrupted or reactivated, a process can typically proceed to a different activity or be terminated with the activation of various alarms.

The communication used in this modeling approach is synchronous, i.e., using CSP semantics. Two processes that communicate connect via a unidirectional channel, and both processes have to participate simultaneously in the communication interaction.

In the usual case, one of the processes, either the sender or the receiver, will have to wait until the other process is available. When the communication is not possible for a specified period, the communication timeout expires and an exception is enabled. Two cases are considered:

- The sender process cannot wait more than 'send_per' time units to communicate with the receiver process. If this occurs, a timer process interrupts the sender and carries out some special activity (an exception).
- The receiver process cannot wait more than ‘rec_per’ time units to communicate with the sender process. As in the previous case, a timer process interrupts the receiver process and carries out an exception.

A timeout can also occur while a process is carrying out an internal activity like closing a gate; this is internal activity or computation. These activities have a specified pair of bounds for the duration of the activity. If the duration of the activity takes longer than the upper bound, a timeout results. If the duration of the activity takes less than the lower bound, a timeout also results.

6 Using Psimj2 to Simulate Real-Time Systems

One good source for a general treatment of simulation is [5]. The Psimj2 library of Java classes can be considered a very flexible and convenient software tool. Object-oriented simulation principles and an introduction to Psimj2 are explained in [6]. Discussions of these concepts applied to modeling and simulation of operating systems and case studies appear in [7]. The Java version of Psim and a set of simulation models in Java appear in [8]. The following Web page provides the Java and the C++ versions of the Psim simulation library, and several simulation models:


The simulation of real-time systems involves simulation of several characteristics of these systems:

- Concurrency. The process interaction approach in Psimj2 implicitly includes facilities for handling concurrent processes. Each process is an active entity in the simulation model and implemented as a Java thread in PsimJ2. Every user class for active objects need to inherit class `Process`.

- Synchronous communication. This direct form of communication is implemented with the `Waitq` class, which includes mechanisms for synchronous cooperation of processes.

- Timing constraints. This is implemented comparing delays using the time domain as a variable of type double.

- Communications deadlines. These deadlines are controlled by setting a deadline on each type of communication. This is implemented using class `Comm_timer`, which automatically checks if a process that is attempting to communicate has exceeded its maximum wait period.

7 Synchronous Communication in Psimj2

Basically, there are three Psimj2 classes necessary to handle synchronous communication and timing constraint for real-time systems:
• Class \textit{Waitq}, for representing unidirectional channels;

• Class \textit{Condq}, for manipulating various timers according to the timing constraints imposed on the system;

• Class \textit{Comm\_timer}, for handling communication timeouts for each type of communication.

When two processes communicate, they are required to participate simultaneously in the communication interaction. The two processes will carry out the joint activity for a finite period. An object of class Waitq is used for every communication channel declared.

The receiver process is considered the master process and the sender is considered the slave process. The sender process uses the qwait operation in class Waitq, and the receiver process uses the coopt operation.

\section{Communication Timers in Psimj2}

A process attempting communication will wait for some specified interval, and then it will check if the sender (or receiver) process is still waiting to communicate. This is implemented in Psimj2 using a communication flag \textit{comm\_wait}, a boolean attribute of class \textit{Process}.

When the timer process checks that the sender (or receiver) process is still waiting after the intervals expired, it will interrupt the sender (or receiver) process. This is implemented in Psimj2 by dequeuing the sender process from the master queue, or dequeuing the receiver process from the slave queue in the channel \textit{Waitq} object that represents the communication channel [8].

When the sender (or receiver) process is reactivated, it checks its interrupt level for the value that corresponds to the communication channel. If the interrupt level corresponds, the process carries out a specified activity like, triggering an alarm, turning off a switch, etc.

To simulate the handling of the earliest and latest times that a communication in a specific process, a normal distribution is used to generate a random time to communicate. The random time generated is used as the time of the communication.

The communication of an event is just a signal; no duration period is involved. A random time is generated that defines the instance at which the communication is initially attempted by a process.
A receiver process may receive more than one type of message or event, through one or more channels (respectively). In this case, the receiver process is suspended waiting for an incoming event or message; the sender process needs to reactivate the receiver process. After restarting, the receiver process will detect which channel has the incoming communication.

When timing is involved in the communication, the master process checks the earliest and latest times of the sender’s message/event, and then it determines if the communication is possible.

A sender or receiver process that needs a communication timer creates an object of class `Comm_timer`. The arguments passed to the constructor of this class are:

1. a reference to the calling process
2. the value of the time interval for the period to wait
3. a reference to the communication channel
4. the communication type of the calling process (sender or receiver).

The simulation model for the Train-Gate system consists of five Java files one for each of the processes described: `Train.java`, `Sensore.java`, `Sensorl.java`, `Monitor.java`, `Controller.java`, and `Gate.java`. The output of a simulation run using PsimJ2 and JVM under Windows is shown in the following listing:

The Java source files for the simulation model of the Train-Gate system are available from the Psim Web page.

9 Conclusion

The basic dynamic behavior of real-time systems can be understood more clearly by modeling with CRSM and then by constructing an object-oriented simulation model. The results of the simulation show the complete sequence of events and the various performance metrics.

Incorporating simulation enhances not only the understanding of the dynamic behavior of real-time systems, but also reinforces other important basic concepts such as concurrency and synchronization; it also helps value modeling in CRSM or in UML. Graphics with animations can further enhance the mentioned advantages of simulation. Finally, simulation helps in the critical testing of the software components of real-time systems; otherwise testing can become very costly and even dangerous.

The supporting Psimj2 software simulation package is available in Java; it can freely be downloaded for educational purposes. More detailed information on the train-gate modeling with UML and the simulation model is available from the Psim Web page.

When using UML diagrams, the commands and events are represented by method invocation of the appropriate object. The state diagrams are similar to the ones shown here. These have to be complemented by other UML diagrams like the sequence diagrams.
Listing for a Simulation Run of the Train-Gate System

PsimJ2 model: Simple Gate-Train system
Simulation date: 8/24/2015 time: 16:16
------------------------- TRACE ---------------------------------
0000.000 Monitor checks channel from esensor
0000.000 Gate waiting for close from controller
0000.000 Entry Sensor waits for train arrival
0000.000 (Exit Sensor) status of Entry Sensor: 1
0000.000 Exit Sensor waits signal from esensor
0425.033 Entry sensor detected train, to comm Monitor
0425.033 Entry sensor to send signal to Monitor
0425.033 Monitor received signal from eSensor
0425.033 (Monitor) Number of trains now: 1
0425.033 Entry sensor sent signal to Monitor
0425.033 Entry sensor ready to send signal to xsensor stat: 1
0425.033 Controller received close signal from Monitor
0425.033 Exit Sensor waits signal from esensor
0425.033 Exit Sensor received signal from Entry Sensor
0425.033 Monitor sent close signal to Controller
0425.033 Entry Sensor sent signal to Exit Sensor
0425.033 Exit sensor waits until 533.142956886139 for depart train
0425.033 Entry Sensor waits for train arrival
0435.432 Controller sending close signal to Gate
0435.432 Gate waiting for close from controller
0435.432 Gate received close signal
0435.432 Gate comm delay: 4.4636995127023384
0439.895 Gate ready to receive close signal
0439.895 Controller in state 3
0439.895 Gate received close signal
0439.895 Controller sent close signal to Gate
0439.895 Gate starts to close
0439.895 Duration for closing gate: 24.104284561343768
0464.000 Gate closed
0464.000 Gate interrupting Controller in 4
0464.000 Gate sending close gok signal
0464.000 Controller interrupted by Gate
0464.000 Controller received gok closed from Gate
0464.000 Gate sent gok closed
0495.663 Entry sensor detected train, to comm Monitor
0495.663 Entry sensor to send signal to Monitor
0495.663 Monitor checks channel from esensor
0495.663 Monitor received signal from eSensor
0495.663 (Monitor) Number of trains now: 2
0495.663 Monitor checks channel from esensor
0495.663 Entry sensor sent signal to Monitor
0495.663 Entry sensor ready to send signal to xsensor stat: 5

2320.305 (Exit Sensor) status of Entry Sensor: 1
2320.305 Exit Sensor waits signal from esensor
2324.330 Monitor delaying to wait for Controller
2328.355 Monitor delaying to wait for Controller
2328.966 Gate closed
2328.966 Gate interrupting Controller in 4
2328.966 Gate sending close gok signal
2328.966 Controller interrupted by Gate
2328.966 Controller received gok closed from Gate
2328.966 Gate sent gok closed
2332.380 Monitor delaying to wait for Controller
2332.380 Monitor sending open signal to Controller
2332.380 Controller delay comm with Gate: 8.691000808288385
2332.380 Monitor received open signal from Controller
2332.380 Monitor checks channel from esensor
2341.071 Controller sending open signal to Gate
2341.071 Gate comm delay: 9.618918670809252
2350.690 Gate to rec open from Controller
2350.690 Controller sent open signal to Gate
2350.690 Gate starts to open
2350.690 Duration for opening gate: 25.795751158900458
2376.486 Gate is completely open
2376.486 Gate interrupting controller
2376.486 Gate to send ok open signal
2376.486 Controller interrupted by Gate (open)
2376.486 Controller receives gok open signal from Gate
2376.486 Gate sent ok open signal
2376.486 Gate waiting for close from controller

End Simulation of Simple Gate-Train system date: 8/21/2015 time: 12:40
Elapsed computer time: 125 msec

Total trains that arrived: 17
Average system reaction time: 5.121
Average system response time: 6.358
Worst reaction time: 17.280
Worst response time: 46.300
Number of deadlines missed in closing Gate: 0
Number of deadlines missed when Controller comm with Gate: 0

References


