Understanding Basic Behavior of Real-Time Systems
Through Object-Oriented Modeling and Simulation

Jose M. Garrido
Department of Computer Science and Information Systems
Kennesaw State University

Abstract

Object-oriented modeling and simulation are powerful although basic conceptual tools for studying (or understanding) of real-time systems. The modeling approach is based on Communicating Real-Time Machines (CRSM). The modeling notation used is the set of extensions to UML (RT-UML) for real-time systems. The object-oriented simulation is the process interaction approach to discrete-event simulation with Psim (a portable set of C++ classes) and PsimJ2 (a set of Java classes). This simulation approach models timing constraints, concurrency and synchronization, in addition to more the more basic expected behavior.

The Train-Gate System is presented as a case study that shows the modeling aspect and the simulation aspect of this study. The results of the simulation display the reaction times, response times, throughput and the deadlines missed.

Introduction

A real-time system is one that usually maintains an on-going 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, from a given input from the environment.

Real-time systems react (or respond) to random events from the environment in a timely fashion. From this point of view, 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 special 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 certain events occur. In this case, there is no response from the system to the environment.

An embedded system is a real-time system that is part of a larger system, is very specialized to suit a specific purpose, and does not usually interact with human users or operators.

Real-time systems usually include a combination of hardware and software. The range of systems span from small embedded systems (microcontrollers) 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 also 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.

**Figure 1. A real-time system and its environment.**

### 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 tasks 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.

The general modeling approach used for real-time systems is object-oriented modeling. A real-time system is modeled as a set of interacting concurrent real-time tasks. Each task represents an active object, which is modeled as a thread. These tasks interact among
themselves and with the environment. Synchronization and communication are part of the mechanisms to model behavior. The communication among tasks 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 [5].

The notation used for modeling real-time systems with the object-oriented approach is the UML and its extensions (RT-UML). A very detailed treatment of UML for real-time systems appears in [4]. The object-oriented simulation, compatible with RT-UML is carried out by applying the process interaction approach to simulation with an object-oriented simulation language.

The Psim library of C++ classes, and the PsimL simulation language provide very flexible and convenient software tools. Simulation principles are explained in [1]. An introduction to Psim as applied with object-oriented simulation appears in [2]. The Java version of Psim and the corresponding model for the train-gate system are also available from the Psim Web page.

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 servicing starts.

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 tasks 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.

**Using Psim to Simulate Real-Time Systems**

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

Concurrence. The process interaction approach in Psim and PsimL implicitly includes facilities for handling concurrent processes. Each active entity is of class `process`.

Synchronous communication.

Timing constraints.

Communications deadlines. These deadlines are controlled by setting a deadline on each type of communication.
Synchronous Communication in Psim

Basically, there are two Psim classes necessary to handle synchronous communication and timing constraint for real-time systems:

- Class `waitq`, for representing unidirectional channels
- Class `condq`, for manipulating various timers according to the timing constraints imposed on the system.
- Class `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 `wait` operation of `waitq`, and the receiver process uses the `coopt` operation.

Timeouts in Communications

A process may take longer than its deadline set for carrying out an activity. In this case, 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.

Typically, once interrupted or reactivated a process can 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 not 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.

Communication Timers in Psim

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 Psim using a communication flag `comm_wait`, a boolean attribute of class `process`.

A sender or receiver process that needs a communication timer creates an object of class `comm_timer`. The arguments passed to the constructor of the `comm_timer` class are:
pointer to the calling process, the value of the time interval for the period to wait, a pointer to the communication channel, and the type of calling process (sender or receiver).

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 Psim by dequeuing the sender process from the master queue, or dequeuing the receiver process from the slave queue in the channel waitq object that represents the communication channel.

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.

Other Timing Considerations

A timeout can occur while a process attempts to communicate, or when a process is carrying out an activity like closing a gate. 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 is attempted by the process at some instant within the specified interval. From this point in time, the process waits for the other process to communicate. If the communication is accomplished in the specified time window, then there is no problem with deadlines for this communication. If the communication was not possible in the specified interval, then a timeout is flagged, and the process enters a different state. The names of channels are the names of the waitq objects defined.

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.

More on Simulation Modeling of Large and Complex Systems

As mentioned above, a real-time system is a very large and complex software component in a computer system. To study the dynamic behavior and the performance of such systems, simulation modeling is recommended as part of the study. In [3] various types of discrete-event simulation models are presented to study different aspects of operating systems. Most of the concurrency and synchronization simulation constructs are illustrated in C++, except timing constraints.
The Psim simulation package for object-oriented modeling and simulation is also explained and used to implement all the models of operating systems presented in the book mentioned above. Psim is basically a set of C++ classes that include all the facilities for object-oriented simulation. A Java version of Psim and the corresponding models are available in Java. The most recent version of the Psim software and the simulation models are available from the web site: http://ksuweb.kennesaw.edu/~jgarrido/psim.html.

---

**Train Crossing Gate**

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

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), the time interval that elapses from the instance a train arrives to the instance the gate opens, is \( \text{resp\_gate} \) time units.

The system view diagram shows all the active objects (processes) and the communication channels defined. The figure below shows the main components (objects and channels) of the system. The following objects are defined:

- The train entering sensor, \( S_e \).
- The train leaving sensor, \( S_l \).
The gate, G.

The monitor, M, that keeps track of the number of trains in the area.

The controller C that controls the gate.

The controller C controls the gate with the open gate (og) and close gate (cg) commands. The Se object will output train-in signals $tr_i$ according to some train interarrival time expression given by $a$, to the monitor object, M. Se will also signal to Sl so that it can expect a train exit in the future. After receiving a signal from Se, object Sl is prepared to sense when a train exits and sends the signal $tr_{out}$ to M.

The gate object 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.

Object M monitors the state of the gate area in order to send appropriate open and close messages to the controller—a required behavior. The controller C sends its signals to the gate.

The output of a simulation run using the Windows/DOS version of the train (train.exe) model is shown in following listing:

```
PsimJ2 model: Simple Gate-Train system
Simulation date: 8/27/2015 time: 11:40
------------------------- 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
0093.826 Entry sensor detected train, to comm Monitor
0093.826 Entry sensor to send signal to Monitor
0093.826 Monitor received signal from eSensor
0093.826 (Monitor) Number of trains now: 1
0093.826 Entry sensor sent signal to Monitor
0093.826 Entry sensor ready to send signal to xsensor stat: 1
0093.826 Controller received close signal from Monitor
0093.826 Exit Sensor waits signal from esensor
0093.826 Exit Sensor received signal from Entry Sensor
0093.826 Monitor sent close signal to Controller
0093.826 Entry Sensor sent signal to Exit Sensor
0093.826 Exit sensor waits until 163.47068345688444 for depart train
0093.826 Entry Sensor waits for train arrival
0097.120 Controller sending close signal to Gate
0097.120 Gate waiting for close from controller
0097.120 Gate received close signal
0097.120 Gate comm delay: 3.7070257605610255
0100.827 Gate ready to receive close signal
...
2590.302 Monitor checks channel from esensor
2595.999 Controller sending open signal to Gate
2595.999 Gate comm delay: 4.513461671969261
2600.512 Gate to rec open from Controller
```
2600.512 Gate received open signal from controller
2600.512 Controller sent open signal to Gate
2600.512 Gate starts to open
2600.512 Duration for opening gate: 26.197271192723804
2626.710 Gate is completely open
2626.710 Gate interrupting controller
2626.710 Gate to send ok open signal
2626.710 Controller interrupted by Gate (open)
2626.710 Controller receives gok open signal from Gate
2626.710 Gate sent ok open signal
2626.710 Gate waiting for close from controller

PsimJ2 model: Simple Gate-Train system
Simulation date: date: 8/27/2015 time: 11:40
Controller timeout: 10.5
Gate timeout: 40.25
----------------------STATISTICS REPORT----------------------

End Simulation of Simple Gate-Train system date: 8/27/2015 time: 11:40
Elapsed computer time: 78 msec

Total trains that arrived: 16
Average system reaction time: 5.274
Average system response time: 9.337
Worst reaction time: 15.360
Worst response time: 34.319
Number of deadlines missed in closing Gate: 0
Number of deadlines missed when Controller comm with Gate: 0

The executable simulation model for the train-gate system (train.exe) is available from the Psim Web page.
Conclusion

The basic dynamic behavior of real-time systems can be understood by modeling with UML and then by constructing an object-oriented simulation model. The results of the simulation show the complete sequence of events and the summary statistics, i.e., the performance metrics. The supporting Psim software simulation package is available in C++ and Java; it can freely be downloaded for educational purposes.

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 UML.

More detailed information on the train-gate modeling with UML and the simulation model is available from the Psim Web page.

References


