Course Instructor: Nida Munawar
Email Address: nida.munawar@nu.edu.pk

Chapter 6

Advanced Dynamic Modeling Illustrated by Real-Time Systems

Textbook: UML 2 toolkit

Real-Time System?

- There are two types of notion for time

1. **Quantitative** : include numbers
2. **Qualitative** : includes words like before , after

- **Real-Time:**
- When time is measured using physical clock

How to measure Correctness

- **Correctness of a system:**
- In terms of what system do we simply ignore how long it takes to do that task

- **Correctness of a Real-time system:**
- If the response in a real time system is too late the system is incorrect

Real-time system

- Real time systems must respond to the events within a certain time
- In the real time systems, the correctness depends both on the response to an input and the time taken to generate that output

Real-time system(Def)

- A Real-time system is a software system where the correct functioning of the system depends on the results produced by the system and the time at which these results are produced
- response should be guaranteed within a specified timing constraint or system should meet the specified deadline. For example: airbag control ,flight control system, real time monitors etc.

Types of Real-time system

1. Soft Real-time system(Noncritical)
2. Hard Real-time system(critical)

Hard RTS

- This type of system can never miss its deadline. Missing the deadline may have disastrous consequences.
- In a hard real-time system, a late (or incorrect) response is considered an unacceptable error that can result in loss of life.
- some examples of these are airbag controllers , earthquake alerts, military tactical training, kidnapping and rescue systems, etc.

Example of Hard RTS

- A good example of hard real-time is a fly-by-wire flight control system where a computer intervenes between the pilot and the engine and control surfaces. The control algorithm depends on precisely timed samples of airspeed, altitude, rate of climb or descent, and so on. If these samples are late, the algorithm can become unstable and the plane crashes.

Soft RTS

A **Soft RTS** is a system whose operation is degraded if results are not produced according to the specified timing requirements

Non-critical RTS are systems in which failure to comply with time restrictions implies a loss of functionality or performance of the system.

The multimedia systems (audio, images and video), are a typical example of these.

Missing the deadline have no disastrous consequences.

Example of Soft RTS

- for instance, in a digital telephone system: it may take a long time to connect a call, or the connection may fail; neither scenario is considered a serious or dangerous error, but they are situations the company wants to avoid.
- A good example of soft real-time is the network of automated teller machines. “Come on,” I hear you say, “the ATM network isn’t real-time.” Oh yes it is, because when you put your card in that machine, you expect a response within a few seconds. So the designers of the ATM network have set a goal of responding in, say, 5 seconds. But they are not going to get terribly upset if occasionally it takes a few seconds longer. After all, what’s the worst that can happen? You get annoyed.

Attributes of a Real-Time System

- **Timeliness is important:** The system performs its functions within specified time limits (“response time”). All specified time deadlines must be handled by the system.
- **It is reactive:**The system is continuously responding to events from the external environment that trigger the execution of the system
- **It contains concurrently executing control processes, where different parts of the software run in parallel.**
- **It has very high requirements in most of the non-function-related areas such as reliability, fault tolerance, and performance.**
- **It is not deterministic:**It is difficult to formally prove that the system will work in all situations under all conditions, due to the complexity of concurrency, the unpredictable events, and the hardware involved.

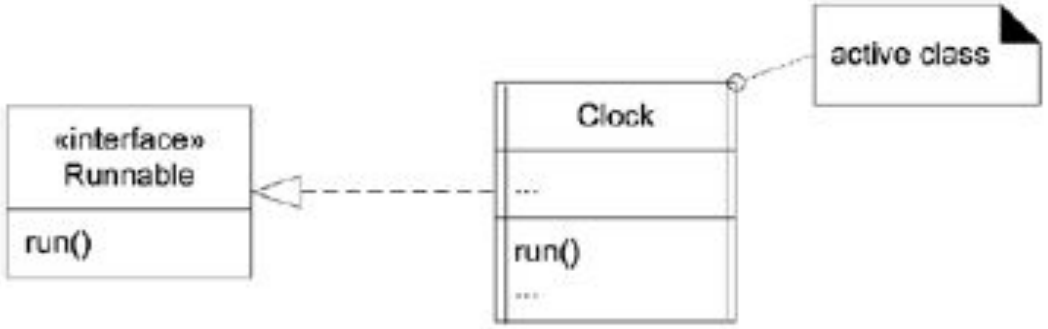
Active Class and Active Object

- The key structure in a real-time system is the **active class or object**. An active class owns an execution thread so that all instances of that class, or the active objects, can initiate control activity
- An active class usually is implemented as a process or a thread.
- The important difference between process and thread is that a process normally encapsulates and protects all its internal structure by executing in its own memory space, while a thread executes in a memory space shared with other threads; this memory space can contain shared information (such as other objects). A thread may require less resources, but this means the shared resources must be managed.

Active Class and Active Object

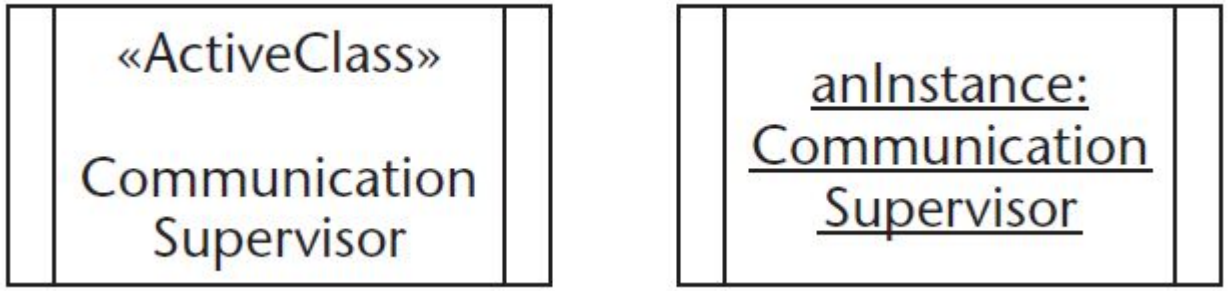
- Active classes are typically implemented through a class library with a superclass for the active class or an interface that an active class can implement.
- This superclass contains the mapping of process or thread operations

such as start, stop, suspend, resume, priority handling, and so on to the corresponding operating system calls that implement these functions.



Active Class and Active Object

- An Active Class indicates that, when instantiated, the Class controls its own execution. Rather than being invoked or activated by other objects, it can operate standalone and define its own thread of behavior.
- An **active object** runs on and controls its own thread of execution. Not surprisingly, the class of an active object is an **active class**. In the UML, it may be shown with double vertical lines on the left and right sides of the class box



Active Class and Active Object

An active object is an object that, as a direct consequence of its creation, commences to execute its classifier behavior, and does not cease until either the complete behavior is executed or the object is terminated by some external object. (This is sometimes referred to as "the object having its own thread of control.") The points at which an active object responds to communications from other objects is determined solely by the behavior of the active object and not by the invoking object. If the classifier behavior of an active object completes, the object is terminated.

Active Class and Active Object

- **Example**
- An object that performs some background task such as garbage collection periodically monitors the environment to see if the task needs to be performed, then performs it if necessary.
- **Example**
- A server object perpetually listens for client requests. When a request arrives, the server launches a request handler object that services the client, then resumes listening. Request handlers are also active objects.