

## Real-time and Embedded Systems Seminar Spring and Fall Quarters 2003

### More refined notes for Lecture 1: [Liu]

“**Job**” – a unit of work that is scheduled and executed by the system – e.g. the computation of a Fast Fourier Transform (FFT) on some sensor data, transmission of a data packet, a control-law computation.

“**Task**” – a set of related jobs which jointly provide some system function.

“A job executes, or is executed by the (operating) system. Every job executes on some resource – e.g. a CPU, a network, or a disk. These resources are called servers in queuing theory literature, and sometimes active resources in real-time systems literature.” (p 26. Jane W.S. Liu’s Real-Time Systems – Prentice Hall)

The **release time** of a job is the instant of time at which the job becomes available for execution. The job can be scheduled and executed at any time at or after its release time whenever its data and control dependency conditions are met. “We say that jobs have no release time if all the jobs are released when the system begins execution.”

The **deadline** of a job is the instant of time by which its execution is required to be completed. For example we may find a system where each control-law computation job must complete by the release time of the subsequent job. {Use the model railroad example.} A job has no deadline if its deadline is at infinity.

The **response time** is the length of time from the release time of a job to the instant when it completes. We call the maximum allowable response time of a job its **relative deadline**. The deadline of a job, sometimes called its absolute deadline, is equal to its release time plus its relative deadline.

In general, we call a constraint imposed on the timing behavior of a job a **timing constraint**.

Hard and Soft Timing Constraints – these are based on the functional criticality of jobs, usefulness of late results, and deterministic or probabilistic nature of the constraints.

A **Hard** timing constraint or deadline is such that the failure to meet it is considered to be a fatal fault. A hard deadline is imposed on a job because a late result produced by the job after the deadline may have disastrous consequences – e.g. failing to stop a train before an impending collision, releasing an anti-missile missile too late to defend a civilian population.

A **Soft** deadline is imposed on a job if occasional failure to meet it causes undesirable effects that are not seriously harmful. (We will omit the philosophical arguments of what is and what is not “seriously harmful”.)

The **tardiness** of a job measures how late it completes relative to its deadline. The tardiness is zero if a job completes before or at its deadline; otherwise the job is late, and its tardiness is

equal to the difference between its actual completion time and its deadline. The usefulness of a result produced by a soft real-time job generally decreases gradually as the tardiness of the job increases. The usefulness of a result produced by a hard real-time job abruptly decreases to zero (and may even become negative) the instant it becomes tardy. (An example of negative usefulness might be the tardy release of an anti-missile missile eventually striking a nearby friendly population center that was supposed to be defended by the tardy missile.) The deadline of a job is softer if the usefulness of its result decreases at a slower rate. One can define a spectrum of hard/soft timing constraints; the quantitative measure of hardness and softness of deadlines is sometimes useful. (We will not get into the moral or philosophical arguments of prioritizing/quantifying the relative hardness of softness of various jobs or the absolute usefulness of various tardy results.)

Jane W.S. Liu's operational definition: "The timing constraint of a job is hard, and the job is a hard real-time job, if the user requires the validation that the system always meet the timing constraint. By validation, we mean a demonstration by a provably correct, efficient procedure or by exhaustive simulation and testing. . . On the other hand, if no validation is required, or only a demonstration that the job meet some statistical constraint (i.e. a timing constraint specified in terms of statistical averages) suffices, then the timing constraint of the job is soft."

(Clark, R. K., "Scheduling dependent real-time activities", Ph.D. thesis, CMU-CS-90-155 August 1990, and Locke, C.D., "Best-effort decision making for real-time scheduling," Ph.D. Thesis, CMU-CS-86-134, May 1986) proposed the distinction between guaranteed and best-effort services as similar to "hard" real-time and "soft" real-time responses.

Liu calls an application (task\_ with hard timing constraints) a hard real-time application and a system containing mostly hard real-time applications a hard real-time system.

It is sometimes advantageous, or even essential, to keep "jitters" in the response times of a stream of jobs small. In this case we do not want to complete the jobs too early or too late. {again, refer to model railroad example}.

See Liu, page 31 for examples of probabilistic constraints.

[Liu] "System design becomes especially difficult when one combines the problems of concurrency with those related to time."

[Shaw]:

Real-time software features:

- a. The dominant role of timing constraints - Not only must a program produce the correct answer or output, but it must also compute the answer "on time".
- b. "Concurrency. Real-time systems must also deal with the inherent physical concurrency that is part of the external world to which they are connected. Signals from the environment can arrive simultaneously; physically disjoint and parallel activities may be monitored and controlled by a single computer systems; it is often necessary to know the real time at which signals are received; and output

signals may need to be emitted at approximately the same time due to timing constraints.”

- c. Reliability and fault tolerance. Reliability is a measure of how often a system will fail. Alternatively, to paraphrase Leveson95 p172, it is the probability that a system will perform correctly over a given period of time.
- d. Criticality – a measure of failure cost. e.g. a computer game may have hard real-time response constraints but low criticality because if the game does not meet the timing constraint on some occasions, there is little chance of any permanent damage to any one or any thing.
- e. “Many real-time systems have one or more humans who interactively control and monitor the system behavior. The human-machine interfaces need to be designed especially carefully in order to prevent human errors and confusion.”
- f. “Testing and certification. Because of the high costs associated with failures, it is usually not feasible to test and debug systems with their actual complete environments. Instead, one must rely upon simulations, testing of subsystems, careful specifications, comprehensive analysis of designs, and extensive run-time e procedures for fault detection and handling.”

[Shaw]: “Despite many successes, the earlier real-time systems suffered from many problems and errors, mainly because the relevant software techniques and methodologies did not exist. Software engineering technology for requirements, design, implementation, testing and debugging, and maintenance was poorly understood in the real-time area. This was especially the case as systems became larger and required more systematic design, modeling, analysis, and organization. In all kinds of systems, but particularly in real-time ones, software complexity can be overwhelming without some underlying methodological help.”

“ . . . the standard hardware technique for improving reliability and recovery from faults is to replicate components; this idea just doesn’t work for software. For many of these older systems, requirements and designs were not specified prior to coding. Concurrency and timing errors appeared frequently, were often not reproducible, and were extremely hard to find and correct.”

A little more recently, “the availability of inexpensive microprocessors made it feasible to monitor and control both simple and complex systems much more efficiently, precisely, and reliably.”

“All of this attention has resulted in some substantial advances. Ideas from operating systems and programming languages have been adapted and extended to real-time software. Computer science modeling and analysis methods, employing techniques in discrete mathematics, were developed for real-time problems. Formal notations specifying requirements and designs were invented or modified. Our knowledge and toolbag have grown impressively, but many challenging problems and issues remain.”

A generic real-time hardware configuration is quite similar to a conventional one, often having several nodes connected through a communications network. However there is usually a larger variety of sensors, actuators, displays, and a number of more precise clocks and timers present in a real-time hardware configuration.

Performance predictability is paramount for real-time systems. Common computer system performance-enhancing features such as instruction and data pipelining and branch prediction, make timing prediction difficult. Also sharing a memory bus between processor and I/O devices can lead to unpredictable performance. “It is more difficult to guarantee deterministic timing over a network, depending on its structure and communications method. If there exist several paths from one node to another, or more than one transmission medium, or if the network is shared, then message transmission times can be unpredictable.”

Standard software “life cycle”:

- Requirements - what the software must do
- design – how the software meets its requirements
- implementation – involves the actual construction of software
- testing (and debugging)
- maintenance.

Many real-time systems have a major safety constraint. Fail-soft is by design.

References:

[Liu] Real-Time Systems by Jane W.S. Liu – Prentice Hall 2000

[Shaw] Real-Time Systems and Software by Alan C. Shaw – John Wiley & Sons, Inc. 2001