



E. Douglas Jensen's
Real-Time for the Real World

[Home](#) | [Search](#) | [Contact Me](#)

My personal manifesto about the widely misunderstood field of real-time computing...

Navigation

[Introduction](#)
[About Me](#)
[Real-Time](#)
[Distributed Real-Time](#)
[Distrib. Real-Time Java](#)
[Real-Time Java](#)
[Real-Time CORBA](#)
[Real-Time Resources](#)
[Our Documents](#)

Sequencing

Generally a real-time computer system has more than one thread that may execute time-constrained actions. In addition, constant time/utility functions may be used for non-time-constrained actions to facilitate coherent scheduling of both time-constrained and non-time-constrained actions.

A subset of these threads (along with other threads not currently having time/utility functions) may be ready to execute with real or virtual concurrency, and hence compete for access to sequentially shared resources. Figure 1 illustrates (using the [time/utility function model](#)) time constraints for actions of four threads that are ready to execute at the current time ("now"). In this example, two of these threads have already been released and two are released at the current time.

News

Real-Time:

[Real-Time Overview](#)

[Time Constraints](#)

[Deadlines](#)

[Time/Utility Functions](#)

[Time Constraints Scopes and Priorities](#)

[Sequencing](#)

[Sequencing Criteria](#)

[Timeliness Optimality](#)

[Predictability](#)

[Hard and Soft Real-Time](#)

[Sequencing Algorithms](#)

[Worked Examples](#)

[Coastal Air Defense](#)

Last updated: 02/29/2004 18:06:18

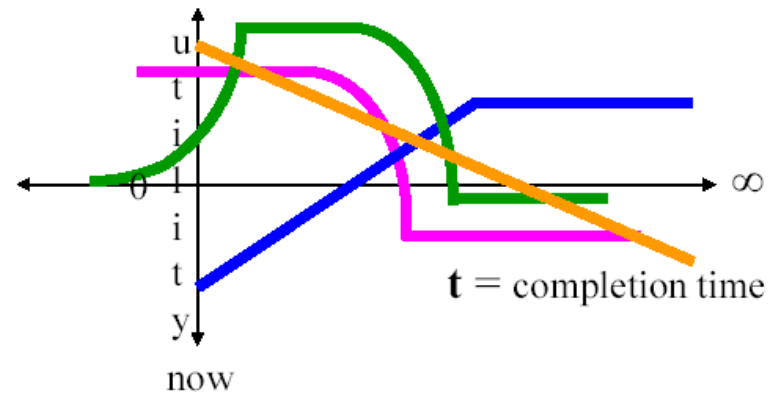
[AWACS Tracker](#)[History](#)

Figure 1: Multiple Concurrent Time Constraints

Thus it must be possible for application designers or users to express any of a variety of application- or even situation-specific policies for managing resources – e.g., resolving conflicts and dependencies, especially contention for sequentially shared resources, among the actions - so as to satisfy

- the actions' time constraints and other execution eligibility criteria
- resource constraints (e.g., mutual interference, power consumption, duty cycle)
- forward progress or fairness requirements for actions having constant time/utility functions

acceptably well.

The best-known example of an exclusive or sequentially shared resource is a processor, but there are numerous other physical and logical ones - e.g., networks and locks, respectively. For brevity and familiarity, here we will focus on thread access to a processor; but coherence of resolving contention for all shared resources is critical to minimizing the occurrence and duration of action timeliness anomalies

("priority inversion" being a simple special case).

Resolving contention by actions for sequentially shared resources is called *sequencing*. There are two forms of sequencing - scheduling and dispatching - as discussed below.

Satisfying the actions' time constraints and other execution eligibility criteria "acceptably well" is measured in two dimensions: optimality of thread sequencing, and predictability of optimality of thread sequencing, discussed on the [sequencing criteria](#) page. Satisfying the resource constraints "acceptably well" is introduced on the [constraint based sequencing](#) page.

Sequencing has a *horizon* - it applies to a time interval for execution of the threads that is delimited by the latest terminal time of the time-constrained ready threads (in general, not all time/utility functions for ready threads have the same range and terminal times).

Sequencing occurs due to *sequencing events* - e.g., a thread becomes ready, a thread completes execution (or is aborted or paused), a thread's time constraint is changed, etc.

Scheduling and Dispatching

Scheduling and dispatching are widely misunderstood in the real-time computing community.

Thread *scheduling* is deciding in what order all currently ready threads will access a resource (e.g., execute on a processor) - i.e., creating a *schedule*. Scheduling may be performed statically (off-line at design or configuration time) or dynamically (on-line at execution time) - in either case by manual or automated means. The basis for creating a schedule is called the *scheduling discipline* (or *policy*). A scheduling discipline is (supposed to be) chosen or devised to satisfy some sequence optimality criterion (sometimes called an *objective function*), as discussed subsequently on the

[sequencing criteria](#) page.

Rate Monotonic scheduling [] is a well known example of a static scheduling discipline in the real-time computing field. Earliest Deadline First (EDF) [] is a well known example of a dynamic scheduling discipline (but not in the practice of real-time computing, as is discussed at the end of this page). Both disciplines satisfy the "hard real-time" sequencing optimality criterion of meeting all deadlines if all deadlines can be met (given certain conditions discussed on the [sequencing policies](#) page). EDF also satisfies the "soft real-time" sequencing optimality criterion of minimizing the maximum lateness (among others) - an example of the fact that a particular sequencing algorithm may satisfy different sequencing optimality criteria under different circumstances (often to the unwelcome surprise of real-time computing practitioners), as discussed on the [sequencing policies](#) page.

Thread *dispatching* is granting resource access to the most eligible thread, according to a *dispatching rule*. When scheduling is employed, dispatching is vestigial - threads are simply dispatched in schedule order. Scheduling is not always necessary or computationally feasible (as will be discussed below). When scheduling is not employed (frequently the case in real-time computing systems), dispatching establishes a sequence one thread at a time - i.e., it determines only the most eligible ready thread and grants that thread access to the resource.

Priority dispatching (i.e., highest priority first) is the most frequently used form of sequencing in real-time computing systems, but it is often erroneously confused with, and referred to as, priority scheduling. Most operating systems for both real-time and non-real-time computer systems do support actual priority scheduling. In the absence of any timeliness semantics for "priority" (e.g., by mapping rates or deadlines to priorities), priority scheduling and dispatching do not perform real-time (meaning time constraint driven) sequencing.

EDF is also widely used as a dispatching rule (primarily in real-time systems outside the computing field) when scheduling is not employed. As when it is used for scheduling (given certain conditions), EDF satisfies the "hard real-time" sequencing optimality criterion of meeting all deadlines if all deadlines can be met, and the "soft real-time" criterion of minimizing the maximum lateness.

In real-time computing system practice, almost no commercial operating system (or other run-time software) products explicitly support any real-time - i.e., application time constraint based - scheduling disciplines or dispatching rules, such as EDF. Users whose applications have (say) deadlines are typically expected to map the deadlines onto operating system priorities, and then manipulate those priorities. Such mappings are complicated in all the but simplest, smallest, and least dynamic systems - for example:

- priority assignments are not modular - they require global knowledge of all other priority assignments (whereas time constraints do not);
- the granularity of time constraints is typically much finer than that of priority ranges;
- semantics are associated with priorities by the users, the system and application software, and the hardware - not always (or even usually) entirely coherently.

These priority mapping complications are exacerbated in [distributed real-time computing systems](#).

There are several reasons why EDF in particular, and other time constraint based disciplines/rules in general, are rarely commercially supported.

- Real-time computing practitioners have often chosen a scheduling discipline or a dispatching rule before they explicitly established a sequencing objective function that the discipline or rule was to optimize, instead of afterwards. In particular, many have intuitively thought

EDF to be optimal without understanding its principles and thus its behavior (both of which are non-obvious), and have suffered the consequences. For example, when it unexpectedly happens that not all deadlines can be met, EDF produces schedules the users may not have anticipated and desired - e.g., that minimize the maximum lateness, but do not necessarily predict which deadlines will be met. (The hard real-time computing community often refers to this property of EDF as "instability," which may be a reasonable characterization if properly and explicitly constrained to the hard real-time context.) If overloads are expected, handling them appropriately requires more sophisticated sequencing criteria and techniques selected for the specifics of the system, application, or circumstances (as discussed on the [sequencing policies](#) page).

- As mentioned above, EDF meets all deadlines if all deadlines can be met, given certain conditions, such as the unlikely case that all eligible threads have the same relative importance; when any of these conditions do not hold, EDF not only fails to satisfy the criterion but the sequencing problem inevitably becomes NP-complete.
- Implementing EDF (even in the special case when it is presumed to be certain that all deadlines can be met) is complicated by the fact that not every action in a system (especially in system software, such as the operating system, and the middleware of a distributed real-time system) can be given a deadline or can even be scheduled at all (e.g., interrupt routines, network routines, the scheduler). Again, this requires more sophisticated and knowledge-based sequencing criteria and techniques (or significantly unconventional architecture and design of the operating system).

Thus, the entire sequencing problem is normally left to the application programmers of real-time computing systems -

without even providing substantively helpful mechanisms in the system software (e.g., operating system, middleware) to support sequencing by higher level software. The Real-Time CORBA 2 specification, and perhaps the forthcoming Distributed Real-Time Specification for Java, are conspicuous exceptions (as discussed later on the [Multi-Node Behavior](#) page).

References:

Next: [Sequencing Criteria](#)

Back to: [Time Constraints](#)



[Add to Favorites](#)



[Print Page](#)



[Download a PDF copy of this page](#)

[View Site Changes](#) | [XML](#) | [Site Updated 02/29/2004](#) | [Legal](#)