

**Response to NITRD RFI**  
**White Paper to Inform the Five-Year Strategic Plan**  
**Soft Real-time Computing in Cyber-Physical Systems**

Gary Nutt  
Computer Science  
University of Colorado  
August 2008

It is common for cyber-physical systems to be designed to support real-time processing, i.e., the system is required to respond to external conditions within a bounded amount of time. Historically such systems are designed to incorporate strict *hard real-time* (HRT) scheduling policies that *guarantee* the system will respond to each external event before a prescribed deadline elapses. In essence, failure to meet the guarantee means that the system has failed, usually resulting in the system being restarted (all ongoing tasks managed by the cyber-physical system are halted, then restarted). In certain cases, (e.g., mission-critical task processing) HRT policies are required to avoid loss of human life or failure of a large, expensive system (e.g., an unmanned space mission). A slightly relaxed variant uses the same admission tests and technology as strict HRT systems, but simply kills a job that overruns its deadline, and then continues operation (without restarting the system).<sup>1</sup> It is quite difficult to design pure HRT systems, since systems every possible execution path of the code must have bounded latency; the presence of multiple tasks can cause unexpected delays due to secondary activities of other tasks (e.g., I/O) or synchronization.

By 1990, researchers realized that many real-time applications would be acceptable even if they occasionally failed to meet the traditional HRT assurances – people began to build real-time system using *soft real-time* (SRT) policies. There are many approaches to implementing SRT policies, e.g., many depend on the way that a job is managed if it misses its deadline, i.e., there is no generally accepted policy for what the system should do in this case. SRT is widely used for tasks such as audio/video streaming tasks (including application domains such as security or observation video and digital voice communication over a network), reading sensor data (such as temperature monitoring), controlling actuators (such as redirecting a tracking telescope). Depending on the exact nature of the application software, SRT systems can support high confidence cyber-physical systems at far less cost; in some cases satisfactory SRT systems can be built when it is infeasible to build a pure HRT system. In SRT systems, different facets of HRT execution are relaxed; for example: jobs in a task may be permitted to overrun their deadline by a fixed amount of time; pre specified percentage of jobs in a task may be permitted to miss their deadlines, such that the system minimizes the amount of time that the collection of software tasks misses their respective deadlines; job may be scheduled so that on *average*, a software tasks does not exceed a bound on the resources it uses (although in any given short period of time it may exceed that bound); jobs may be delayed for an entire task phase; etc.

SRT methodology is greatly influenced by HRT methodology, typically sharing the same assumptions, early principles, and proven approaches for HRT systems. For example, early HRT schedulers often used rate monotonic (RM) scheduling policies since they satisfy well-known bounds for admission and scheduling that were published in 1973, and since RM was widely used in HRT system in the next two decades. The earliest deadline first (EDF) scheduling approach was an early competitor with RM, and has certain desirable properties in SRT systems (e.g., it can

---

<sup>1</sup> Real-time workload is typically specified as a collection of *tasks*, each of which is executed as a series of *jobs*. Thus a task is often periodic, with one job executing in each period of the task. Each job is specified with an a priori service time and deadline.

have better processor utilization than RM); however in early kernel scheduler technology, it was more difficult to implement than RM. EDF schedulers have been shown to have certain advantages in SRT system, yet it is sometimes difficult to publish SRT research, or to obtain funding for a grant proposal, if it depends on EDF scheduling.

Along the same lines, SRT system applications are typically designed under the assumption that all software tasks are implemented as well-informed, well-intentioned HRT style programs that, e.g., would not attempt to make one statement of resource need to the system, but then either naively or covertly *far* exceed that stated need.<sup>2</sup> For example, in SRT systems that use slack time scheduling techniques, the scheduler is likely to ignore the possibility that a task could effectively launch a denial of service attack on the real-time scheduler simply by greatly understating the service time (earning high admission and scheduling priority, but using much greater amounts of resources if they are available). This greatly influences the breadth, complexity, and availability of application software for SRT systems; each SRT system typically uses its own specialized set of applications rather than drawing from an open pool of application programs.

The essence of this input statement is that SRT systems are an engineering reality for most cyber-physical systems, but that SRT systems and software technology could provide far more useful support to practical cyber-physical systems than is currently possible using the existing hybrid theoretic foundations derived from classic HRT systems.

I imagine a design environment in which SRT software is required to state its resource requirements *a priori*, but in which cyber-physical system applications can be built using modern software techniques and constraints. Designers should be able to easily choose a SRT (or hybrid HRT/SRT) system policy based on pure SRT criteria rather than as softening of HRT; if the particular cyber-physical system requires strict HRT management, then it should be built using the classic HRT principles. But if the cyber-physical system can be built with favorable cost-reliability tradeoffs or alternatively infrastructure (such as EDF scheduling), the design environment should support a spectrum of policies that enable the designer to relax certain policies while maintaining other, and have a clear understanding of the implications of such policy changes.

I imagine a real-time based design environment in which the cyber-physical system designer can incorporate a much more diverse set of applications from diverse software providers while still be able to make the selected assurances about the real-time behavior of the system. Success in this area would greatly increase the amount and quality of software that could be used in cyber-physical systems.

It is very difficult to “reset” an entire discipline, such as soft real-time computing, by reexamining basic principles, since many results in the area rely on assumptions built into the logical design environment. However, modern cyber-physical systems are being held back by the aging set of assumptions that were developed for hard real-time systems, and by the creation of sound SRT methodology through the “sprawl” of HRT methodology. In other cases, the technology is limited by principles that are simply accepted as “the way to do things,” even though technology has evolved (or could evolve) to accommodate better approaches. An effort to reexamine and update the base assumptions cannot possibly be done by any single researcher, or even a single funding agency; it is a movement that can only be by an agency such as NITRD.

---

<sup>2</sup> HRT systems rely on each job’s service time being specified as its worst case execution time, which causes the scheduler to operate using the most conservative admission and scheduling approach. SRT systems typically depend on the applications to make more aggressive service estimates, but ones that approximate the WCET, or perhaps the average job execution time for all jobs in a task.