# The Modeling and Analysis of Mixed-Criticality Systems

## Extended Abstract

Sanjoy Baruah*

Department of Computer Science,
The University of North Carolina
Chapel Hill, NC 27599, USA
baruah@cs.unc.edu

**Abstract.** Methodologies that are currently widely used in the design and implementation of safety-critical real-time application systems are primarily focused on ensuring correctness. This, in conjunction with the trend towards implementing such systems using COTS components, may lead to very poor utilization of the implementation platform resources during run-time. Mixed-criticality implementations have been proposed as one means of achieving more efficient resource utilization upon such platforms. The real-time scheduling community has been developing a theory of mixed-criticality scheduling that seeks to solve resource allocation problems for mixed-criticality systems. There is a need for the formal methods and analysis community to work on developing methodologies for the design and analysis of mixed-criticality systems; such methodologies, in conjunction with the work on mixed-criticality scheduling currently being done in the real-time scheduling community, has the potential to significantly enhance our ability to design and implement large, complex, real-time systems in a manner that is both provably correct and resource-efficient.

### Correctness and Efficiency Considerations

The discipline of real-time computing has its origins in the application domains of defense, space, and aviation. These are all highly safety-critical domains that place a great premium on correctness, since the consequences of incorrect system behavior is potentially very severe. Furthermore, early systems in these application domains had limited computational capabilities upon which to implement the desired functionalities. Early real-time systems were therefore required to both be *correct*, and have *resource-efficient implementations*.

In the early years of the discipline, these twin goals of correctness and efficiency were achieved by keeping things very simple: safety-critical computer systems were restricted to being very simple, responsible for very simple, highly

* Supported in part by NSF grants CNS 1016954, CNS 1115284, and CNS 1218693; and ARO grant W911NF-09-1-0535.

A. Legay and M. Bozga (Eds.): FORMATS 2014, LNCS 8711, pp. 1–6, 2014.

repetitive, functionalities. They were typically implemented as carefully hand-crafted code executing upon very simple and predictable processors. Run-time behavior was therefore very predictable, and hence correctness could be demonstrated in a fairly straightforward manner.

### Choosing Correctness Over Efficiency

However, things soon became far more complicated. The requirements placed upon safety-critical computer systems increased significantly in size and complexity, and continue to increase at a very rapid pace. This meant that safety-critical systems could no longer be implemented upon simple and predictable processors; instead, advanced modern processors that offer far greater computational capabilities but exhibit less predictable run-time behavior increasingly came to be used in implementing even highly safety-critical real-time systems. As a consequence of this increase in the complexity and diversity of real-time application system requirements and implementation platforms, it soon became impossible for a single system developer, or a small group of developers, to keep all details in mind while reasoning about a system design and implementation. Instead, it became necessary to introduce *abstractions* that would highlight the relevant aspects of a system's behavior while concealing the less important aspects.

In devising these abstractions, the real-time systems research community was deeply influenced by the increasingly central role that computer systems are coming to play in the control of safety-critical devices and systems. The increasing criticality of these systems, coupled with their increasing complexity and diversity, meant that correctness was becoming both more important, and more difficult to achieve. In contrast, ensuring efficiency of implementation became less important as Moore's Law, compounded over decades, made it possible to provide larger amounts of computing capabilities at relatively low cost.

Given this state of affairs — correctness becoming increasingly both more important and more difficult to achieve, and efficiency mattering less — the real-time systems community made the rational decision to focus on abstractions that facilitate the correct construction of systems, letting efficiency considerations recede to the background. An important advance here was the introduction of the *synchrony assumption* [2], which separated functional and temporal correctness concerns by introducing the concept of logical time. Time considerations were re-integrated into a functionally correct design by the use of abstractions and mechanisms such as logical execution time, timed automata, etc. These abstractions, which are sometimes collectively referred to by the umbrella term *model-based design* for timed systems, have proved extremely popular; supported by powerful tools and rigorous proof and design formalisms, such model-based design methodologies are widely used today in developing systems in safety-critical industrial domains such as automotive, aviation, etc.

But it is important to realize that the end-product of the process of designing a system using these model-based design methodologies is a *model* of the system, not its physical realization upon an implementation platform. We do

not currently know how to implement such models upon modern advanced platforms in a resource-efficient manner; instead, current practice is to use ad hoc techniques to obtain an implementation of a model that is developed using a model-based design process (and thereby rigorously proved to be correct), without worrying too much about efficiency, compensating for this lack of efficiency by an over-provisioning of computational and other resources upon the implementation platform.

### The Problem of Modern Platforms

As safety-critical systems became ever more complex and computationally demanding, it became necessary to implement them upon the most advanced computing platforms available. Due to cost and related reasons, such platforms are increasingly coming to be built using commercial off-the-shelf (COTS) processors and other components. COTS processors are generally developed with the objective of providing improved "typical" or average-case performance rather than better worst-case guarantees; they therefore incorporate advanced architectural features such as multi-level cache memories, deep pipelining, speculative out-of-order execution, etc., that do indeed significantly improve average performance but also lead to very large *variances* in run-time behavior. For example, it is known [1] that the simple operation of adding two integer variables and storing the result in a third may take from as few as 3 to as many as 321 cycles on the Motorola PowerPC-755 (by contrast, the earlier Motorola 68K processor, which was state-of-the art in the 1990's, always executes this operation in exactly 20 cycles). In order to predict the precise behavior that will be experienced by a particular process during run-time, extensive knowledge of the run-time situation –the inputs to the process at run-time; the states of the other processes that are executing concurrently; etc.– must be known; since such knowledge is not usually obtainable during system design time, the run-time behavior is *unpredictable* beforehand.

### The Move towards Mixed-Criticality Systems

Summarizing the points made above:

- Safety-critical system requirements have become vastly more complex, and more computation-intensive.
- They must therefore be implemented upon advanced modern computing platforms, which offer increased computing capabilities but are unpredictable and exhibit great variance between average-case and worst-case behavior.
- Given the extremely safety-critical nature of the applications, though, their correctness must nevertheless be validated to extremely high levels of assurance.

Since the systems are so complex and the implementation platforms so unpredictable, system correctness at the desired high levels of assurance is guaranteed

during the system design process by tremendous over-provisioning of computational and other platform resources during system design time. However, the very conservative assumptions that must be made during validation in order to ensure correctness at the desired levels of assurance are highly unlikely to occur during the typical run; hence, much of the over-provisioned resources are unlikely to actually be used during run-time. As a consequence, such system implementations will see extremely low resource utilization during run-time. **SWaP** concerns (the Size, and Weight of the implementation platform, and the Power, or rather, the energy, that is consumed by it) make such resource under-utilization increasingly unacceptable. One approach towards improving run-time resource usage is by moving towards *mixed-criticality* implementations, in which the highly safety-critical functionalities are implemented upon the same platform as less critical functionalities. This approach has proved very popular in safety-critical application domains, as is evident in industry-driven initiatives such as Integrated Modular Avionics (IMA) [4] in aerospace and AUTOSAR[1] in the automotive industry. Informally speaking, the idea is that the resources that are provisioned to highly critical functionalities during design time, but are likely to remain unused by these functionalities at run-time, can be "re-claimed" and used to make performance guarantees, albeit at lower levels of assurance, to the less critical functionalities.

### Mixed-Criticality Scheduling Theory

The recently emergent field of mixed-criticality scheduling theory (see, e.g., [3] for a current survey) is concerned with the study of resource-allocation, scheduling, and synchronization in such mixed-criticality systems. Two related but distinct approaches have been widely investigated: one focused primarily on run-time robustness, and the other on verification.

*Run-time robustness* is a form of fault tolerance that allows graceful degradation to occur in a manner that is mindful of criticality levels: informally speaking, in the event that all functionalities implemented upon a shared platform cannot be serviced satisfactorily the goal is to ensure that less critical functionalities are denied their requested levels of service before more critical functionalities are. Approaches in mixed-criticality scheduling theory that seek to ensure such run-time robustness are centered upon identifying, during run-time, when cumulative resource demand exceeds the available supply, and triggering a *mode change* [6] when this happens. Real-time scheduling theory has a rich history of results towards obtaining resource-efficient implementations of mode changes (see [5] for a survey); these techniques may be adapted to ensure run-time robust mixed-criticality systems.

*Static verification* of mixed-criticality systems is closely related to the problem of *certification* in safety-critical application domains. The accelerating trend in safety-critical application domains such as automotive and avionics systems towards computerized control of an ever-increasing range of functionalities, both

[1] AUTomotive Open System ARchitecture — see `www.autosar.org`

safety-critical and non-critical, means that even in highly safety-critical systems, typically only a relatively small fraction of the overall system is actually of critical functionality and needs to be certified. In order to certify a system as being correct, the certification authority (CA) may mandate that certain assumptions be made about the worst-case behavior of the system during run-time. CA's tend to be very conservative, and hence it is often the case that the assumptions required by the CA are far more pessimistic than those the system designer would typically use during the system design process if certification was not required. However, while the CA is only concerned with the correctness of the safety-critical part of the system the system designer wishes to ensure that the entire system is correct, including the non-critical parts. Vestal [7] first identified the challenge of obtaining certification for integrated system implementations in which different functionalities need to have their correctness validated to different levels of assurance, while simultaneously ensuring efficient resource-utilization. The real-time scheduling community has since produced a vast amount of work that builds upon Vestal's seminal idea; see [3] for a survey.

## A Need for Participation by the FORMATS Community

These advances in mixed-criticality scheduling theory point to a promising approach towards reintegrating efficiency considerations into the design and implementation of provably correct safety-critical real-time application systems. However, much of this work is based upon relatively low-level and simple workload models, such as collections of independent jobs, or systems represented as a finite collection of recurrent (e.g., periodic and sporadic) tasks. Prior experience has shown that such simple models are inadequate for building truly complex systems – more powerful models, at higher levels of abstraction and possessing greater expressive power, are needed. The development of such models, along with accompanying design methodologies, proof formalisms, and tool support, is one of the prime strengths of the formal methods and modeling community. There is therefore a pressing need for the formal methods and modeling community to take a close look at mixed-criticality systems, to develop more powerful models for representing such systems, and to extend the mixed-criticality scheduling theories to become applicable to these more advanced models. It is my belief that such work is best conducted in co-ordinated, cooperative efforts between the formal methods and the real-time scheduling communities.

**Acknowledgements.** The ideas discussed in this extended abstract are based upon discussions with a number of colleagues and research collaborators, Alan Burns in particular. Others include (in alphabetical order) Jim Anderson, Saddek Bensalem, Vincenzo Bonifaci, Pontus Ekberg, Gerhard Fohler, Laurent George, Nan Guan, Alberto Marchetti-Spaccamela, Joseph Sifakis, Leen Stougie, Lothar Thiele, Steve Vestal, and Wang Yi.

## References

1. Slide-show: Introduction to aiT, http://www.absint.com/ait/slides/4.htm (accessed on June 23, 2014)
2. Benveniste, A., Berry, G.: The synchronous approach to reactive and real-time systems. Proceedings of the IEEE 79(9), 1270–1282 (1991)
3. Burns, A., Davis, R.: Mixed-criticality systems: A review (2013), http://www-users.cs.york.ac.uk/~burns/review.pdf
4. Prisaznuk, P.J.: Integrated modular avionics. In: Proceedings of the IEEE 1992 National Aerospace and Electronics Conference (NAECON 1992), vol. 1, pp. 39–45 (May 1992)
5. Real, J., Crespo, A.: Mode change protocols for real-time systems: A survey and a new proposal. Real-Time Syst. 26(2), 161–197 (2004)
6. Sha, L., Rajkumar, R., Lehoczky, J., Ramamritham, K.: Mode change protocols for priority-driven preemptive scheduling. Real-Time Systems 1, 243–264 (1988)
7. Vestal, S.: Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In: Proceedings of the Real-Time Systems Symposium, pp. 239–243. IEEE Computer Society Press, Tucson (2007)