Embedded Systems Research: Missed Opportunities

(ISORC ’08 Panel Position Paper)
Bran Selic
Malina Software Corp.
selic@acm.org

With society’s increasing dependence on software, the issue of software quality is becoming more prominent. The term quality of service (QoS) has been used in conjunction with software to cover a broad range of characteristics, such as dependability, responsiveness, security, etc. Historically, the technical challenge of achieving a desired QoS in conditions of limited resources (e.g., finite computing power, memory capacity, communication bandwidth) was primarily relegated to the specialized domain of real-time and embedded software systems, that is, systems involved in continuous and timely interaction with the physical world. The “mainstream”1 view and dominant software design philosophy are still based on the flawed premise that software should be developed without any considerations given to the characteristics and limitations of the hardware and software platforms that give it life. The important and useful principle of platform independence is being tragically misinterpreted as implying platform ignorance.

The software world in general could learn much on how to address QoS concerns from the experience and knowledge accumulated in the real-time and embedded domain. But, this is not happening on any appreciable scale and, instead of taking advantage of this readily available shortcut, there is a process of slow and painful rediscovery.

To be fair, the mainstream software community is not solely to blame for this – much of the fault lies with embedded software researchers, who have isolated themselves by focusing on increasingly more specialized low-level infrastructure problems. For example, in the most recent edition of the IEEE Real-Time and Embedded technology and Applications Symposium (RTAS 2007) – a prestigious high-caliber technical conference – the great majority of papers focused on low-level issues such as scheduling, FPGA programming, and managing energy consumption, with only a smattering of papers dedicated to higher-level topics such as programming models [1].

This is not to say that infrastructure problems are unimportant (on the contrary), but that there is a dearth of research dealing with high-level architectural problems and processes, issues that are generally of greatest consequence when it comes to software design. There is very little research being done on how to bridge the gap between these two polar opposites. (One notable exception is the recent work on the UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [2]).

When we examine a typical modern real-time system, such as a wireless telecommunications network, the complexity encountered is truly staggering. Such systems comprise extremely sophisticated hardware/software complexes, developed by diverse teams with very different areas of specialization, working on different time scales and using very different processes. Yet, it all has to come together and work correctly and reliably and within the stringent QoS envelope set for the system. Unfortunately, we do not have a well-developed theory of how to successfully construct such complex systems. What architectural patterns apply here? How do we express system-level QoS requirements in such systems and how do we ensure that they are fulfilled as we work top down or bottom up across their numerous and heterogeneous layers? Which development and management processes should be used? (e.g., where and how can we use modern agile software development processes in such an environment?)

As more and more modern software systems fall into this category of complex real-time systems, we are in desperate need of systematic and comprehensive theories that would help us answer these difficult questions. It is an area of research that has barely been touched, yet, it seems critically important.

References


1 We use this rather crude term as a convenient catch-all to denote all domains other than real-time and embedded systems.