Chapter 8
Emerging Real-Time Methodologies

Giorgio C. Buttazzo
Scuola Superiore Sant'Anna, Italy

ABSTRACT

The number of computer-controlled systems has increased dramatically in our daily life. Processors and microcontrollers are embedded in most of the devices we use every day, such as mobile phones, cameras, media players, navigators, washing machines, biomedical devices, and cars. The complexity of such systems is increasing exponentially, pushed by the demand of new products with extra functionality, higher performance requirements, and low energy consumption. To cope with such a complex scenario, many embedded systems are adopting more powerful and highly integrated hardware components, such as multi-core systems, network-on-chip architectures, inertial subsystems, and special purpose co-processors. However, developing, analyzing, and testing the application software on these architectures is not easy, and new methodologies are being investigated in the research community to guarantee high predictability and efficiency in next generation embedded devices. This chapter presents some recent approaches proposed within the real-time research community aimed at achieving predictability, high modularity, efficiency, and adaptability in modern embedded computing systems.

1. INTRODUCTION

Complex embedded systems, like cell phones and multimedia devices, share a number of features that determine a set of design requirements:

- **Limited resources:** Small portable devices are designed under space, weight, and energy constraints. Often they also have cost constraints related with mass production and strong industrial competition. As a consequence, embedded applications typically run on small processing units with limited memory and computational power. To make these devices cost-effective, it is mandatory to make a very efficient use of the computational resources, both at the application and the operating system level.
- **High concurrency and resource sharing:** Typically, several functions are simultane-

DOI: 10.4018/978-1-4666-3922-5.ch008

ously active, often sharing the same set of resources. For instance, in a cell phone, different tasks can be performed at the same time, such as browsing, downloading, playing music, and receiving a call. Most of them need the same resources to run, as processor, memory, display, and sound codec. As a consequence, tasks can experience variable blocking delays due to the interference generated by the synchronization mechanisms required to access shared resources. To reduce inter-task interference and make tasks execution more predictable, proper scheduling algorithms have to be adopted at the operating systems level to isolate the timing behavior of concurrent tasks.

- **Interaction with the environment:** Most embedded devices interact with the environment and have demanding quality specifications, whose satisfaction requires the system to timely react to external events and execute computational activities within precise timing constraints. The operating system is responsible for ensuring a predictable execution behavior of the application to allow an off-line guarantee of the required performance.
- **Dynamic behavior:** In embedded systems characterized by several concurrent tasks, the overall computational load is not constant, but can have high variations, depending on the tasks activated by the user and the resources needed by the activities, which in turn may depend on the actual data. If not properly handled, overload conditions may cause undesired effects, from a transient performance degradation to a complete system crash. A certain degree of adaptivity in the resource management policies is essential to reallocate resources during peak load situations.

The combination of real-time features in tasks with dynamic behavior, together with cost and resource constraints, creates new problems to be addressed in the design of such systems, at different architecture levels. The classical worst-case design approach, typically adopted in hard real-time systems to guarantee timely responses in all possible scenarios, is no longer acceptable in highly dynamic environments, because it would waste the resources and prohibitively increase the cost.

Instead of allocating resources for the worst case, smarter techniques are needed to sense the current state of the environment and react as a consequence. This means that, to cope with dynamic environments, a real-time system must be *adaptive*; that is, it must be able to adjust its internal strategies in response to a change in the environment to keep the system performance at a desired level or, if this is not possible, degrade it in a controlled fashion.

Implementing adaptive embedded systems requires specific support at different levels of the software architecture. The most important component affecting adaptivity is the kernel, but some flexibility can also be introduced above the operating system, in a software layer denoted as the *middleware*. Some adaptation can also be done at the application level; however, it potentially incurs in low efficiency due to the higher overhead normally introduced by the application level services. Normally, for efficiency reasons, adaptation should be handled at the lower layers of the system architecture, as close as possible to the system resources. For those embedded systems that are distributed among several computing nodes, special network methodologies are needed to achieve adaptive behavior and predictable response.

The rest of this document presents several techniques to make next generation embedded systems more predictable and adaptive to environmental changes. In particular, Section 1 summarizes the most relevant results on real-time scheduling; Sec-