ACM SIGBED Review, Vol. 3, No. 3, July 2006.

Research Trends in Real-Time Computing for Embedded Systems

Giorgio Buttazzo
Scuola Superiore Sant'Anna
Pisa, Italy
giorgio.buttazzo@sssup.it

Abstract

Most of today's embedded systems are required to work in dynamic environments, where the characteristics of the computational load cannot always be predicted in advance. Still timely responses to events have to be provided within precise timing constraints in order to guarantee a desired level of performance. Hence, embedded systems are, by nature, inherently real-time. Moreover, most of embedded systems work under several resource constraints, due to space, weight, energy, and cost limitations imposed by the specific application. As a consequence, efficient resource management is a critical aspect in embedded systems, that must be considered at different architecture levels.

The objective of this document is to present the major research trends identified by the international community on real-time systems to make the next generation embedded systems more predictable and adaptive to environmental changes. After describing the characteristics of modern embedded applications, the paper presents the problems of the current approaches and discusses the new research trends in operating systems and scheduling emerging to overcome them.

1 Introduction

The use of computer controlled systems has increased dramatically in our daily life. Processors and microntrollers are embedded in most of the devices we use every day, such as mobile phones, PDAs, TVs, DVD players, cameras, cars, dishwashers, etc. This trend is expected to continue in the future. Several research projects on ambient intelligence, pervasive systems, home automation, and ubiquitous computing, aim at integrating computers in our environment even more in a way that they are hidden. Most of these devices share the following important properties:

- *Limited resources*. Several embedded devices are designed under space, weight, and energy constraints imposed by the specific application. 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. In order to make these devices cost-effective, it is mandatory to make a very efficient use of the computational resources.

- *Real-time constraints*. 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*. The complexity of embedded systems is constantly increasing and several applications consist of tens or hundreds of concurrent activities that interact with each other and compete for the use of shared resources. In addition, the behavior of some activities depends on sensory data inputs, which can hardly be predicted in advance. Finally, low-level architecture features, such as caching, pre-fetching, pipelining, DMA, and interrupts, although enhancing the average computer performance, introduce a non deterministic behavior on tasks execution, making the estimation of worst-case computation times very unpredictable. As a consequence, the overall workload of complex real-time applications is subject to significant variations that cannot be easily predicted in advance.

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 envi-

ronments, 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 mainly focuses on operating systems, presenting the major research trends in real-time scheduling and resource management identified by the international community on real-time systems to make the next generation embedded systems more predictable and adaptive to environmental changes. After analyzing the current state of practice in some real-time application domains, the paper presents the limitations of current solutions and the new research trends emerging to overcome them. A more complete discussion on other architectural aspects, including middleware, networks, languages, and design methodologies, can be found in the ARTIST Roadmap for embedded systems design [31].

In particular, Section 2 illustrates the characteristics of modern applications; Section 3 explains the problems of the current industrial practice; Section 4 reports the most innovative research areas in operating systems and scheduling to overcome these limitations; and Section 5 concludes the paper summarizing the major research challenges.

2 Novel applications requirements

The complexity of embedded systems is constantly increasing. While in the past embedded systems were synonym of 8-bit processors and small memory footprint, most of today's systems are developed on 32-bit processors with several megabytes of memory, and some of them include advanced multimedia features. For example, a modern mobile phone typically consists of several million lines of code with use-cases involving large number of concurrent activities. Utilizing available hardware and software resources in an optimal fashion is crucial both to save costs and to keep the competitive edge.

In the large domain of consumer electronics, most of the application software includes at least three types of activities, with different characteristics and requirements with respect to timing and resource sharing:

- Control software is typically implemented by periodic tasks and uses only a small fraction of the available resources. Sensory acquisition and control tasks are subject to hard timing requirements that must be guaranteed off line in all operating conditions.

- Media processing software is typically data or throughput driven, and is a major consumer of hardware resources (processor and network bandwidth). Audio/video processing and graphics applications are the main examples in this category. They are normally treated as soft real-time aperiodic tasks with quality-of-service (QoS) requirements. Due to the large resource consumption, achieving high resource utilization is of crucial importance.

- Interaction software is very complex and vastly increasing in size and complexity. Typical examples are electronic program guides, internet browsing, photo/music browsing, broadcast enhancement (for example, player info and statistics in sports games). In a high-end TV set, the total code size currently approaches 4 Mbytes. For this software, the timing requirements are interactive-response requirements.

A problem in handling such different types of activities is to manage the available resources so that each class of tasks can meet its specific quality constraints. Adding or removing features may cause a system to fail. For complex and dynamic systems, exhaustive design and testing of every possible use case is not tractable. Therefore, tools and metrics for expressing and handling resource requirements are essential in future system design.

In more sophisticated devices (e.g., smart phones), these problems are often addressed by multiple CPU solutions, where one CPU is dedicated to real-time tasks, while another CPU runs the user applications on top of a non real-time kernel. Such a separation between real-time and non real-time functionality simplifies resource allocation and protects real-time tasks from a misbehaving user application. However, more CPUs consume more power, require special multiprocessor scheduling algorithms and there is still a need for managing resources on the individual processors and support the communication among tasks running on different processors.

A single CPU system with proper resource management would provide a viable alternative. For example, a system supporting "memory and temporal protection" would allow safely mixing real-time and non real-time applications with the benefit of achieving a more scalable platform. Adding and removing features would become predictable, allowing configuring the system without worrying about unpleasant surprises.

Finally, especially in multimedia systems, embedded applications exhibit a highly dynamic behavior, since task execution times are often depended on input data that are difficult to predict. For example, if a multimedia task manages compressed frames, the time for coding/decoding each frame can vary significantly; hence, the worst-case execution time (WCET) of the task can be much higher than its mean execution time.

3 Problems with the current approach

In spite of the increased systems complexity, real-time applications are mainly configured acting on task priorities, which usually express the importance of tasks. This is for many reasons inadequate when configuring complex dynamic systems, because there are other system constraints that cannot be mapped into a set of priority levels. As a consequence, today, systems require extensive testing and tuning to operate optimally.

Another problem with priorities is that activities often consist of several tasks, which may play different roles in different scenarios, making the priority assignment even more difficult. Any attempt to group tasks together fails since priority is a global property and will always break any type of encapsulation.

When dealing with dynamic applications with variable execution requirements, the use of dynamic priorities schemes would give higher flexibility to the system, allowing better adaptivity and full resource exploitation. However, most of today's industrial products with dynamic behavior are still based on fixed priority schemes and have very limited flexibility. The main reason is due to the fact that they are built on top of commercial components that do not offer the possibility of being reconfigured at runtime. For example, at the operating system level, most of the internal kernel mechanisms, such as scheduling, interrupt handling, synchronization, mutual exclusion, or communication, have a fixed behavior dictated by a specific policy that cannot be easily replaced or modified.

The typical approach used today at the operating system level to affect the execution behavior and achieve some level of adaptation is to modify task priorities. However, this method does not always succeed and it is not trivial to predict how the system performance will change as a function of priorities. For example, increasing the priority of a task with long execution time could lead to an overload condition that would degrade system performance. Even decreasing the priority of a task could create problems, since it would implicitly raise the relative priority of other tasks, so also leading to an overload situation. If tasks interact through shared resources, task priorities also affect the delays caused by blocking on critical sections. If the kernel uses a priority inheritance protocol for accessing shared resources, changing the priority of a task at the "wrong" time instant could interfere with the protocol and cause very undesirable effects, such as priority inversion [32]. This examples show that today's commercial operating systems are not suited for on-line adaptation because they do not provide explicit support for quality-of-service (QoS) management.

4 Challenges and work directions

The most important mechanism in the operating system affecting adaptiveness is scheduling. Unfortunately, however, the majority of today's commercial operating systems schedule tasks based on a single parameter, the priority. Recent research on flexible scheduling showed that a single parameter is not enough to express all the application requirements. In order to provide effective support to QoS management, modern operating systems should be:

- *Reflective*. That is, they should reflect the application characteristics into a set of parameters, which can be used by appropriate scheduling algorithms to optimize system performance. For example, typical parameters that may be useful for effective task management include deadlines, periodicity constraints, importance, QoS values, computation time, and so on.

- *Resource aware*. That is, they should give the possibility of partitioning the resources (e.g., the processor) among the existing activities based on their computational requirements. Such a partitioning would enforce a form of temporal protection that would prevent reciprocal interference among the tasks during overload conditions.

- *Informative*. That is, they should provide information on the current state of execution to allow the implementation of adaptive management schemes at different levels of the software architecture. Any difference between the expected and the actual behavior of a computation can be used to adjust system parameters and achieve a better control of the performance.

To achieve these general objectives, further research is needed in several areas. They are illustrated in the following sections.

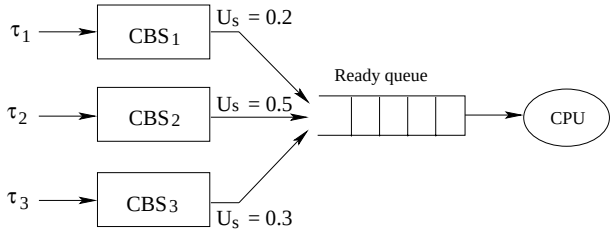

Figure 1. Example of three tasks running under a CPU reservation scheme implemented by a CBS.

4.1 Resource Reservation

The problems caused by priorities, mentioned in Section 3, could be solved by using a programming model that enables the designer to explicitly control the resources assigned to a given activity at a given point in time. With reservation-based scheduling, a task or subsystem receives a real-time share of the system resources according to a given allocation policy [27].

Notice that, from a timing point of view, receiving a fraction of the processor is equivalent to executing on a private processor running at reduced speed. As a consequence, this approach isolates the execution of tasks running under resource reservations, thus protecting the other activities from possible overruns (temporal protection).

Resource reservation can be efficiently implemented using a Constant Bandwidth Server (CBS) [2], which is a service mechanism allocating a budget of Q_s units of time every period T_s. The ratio Q_s/T_s is denoted as the server bandwidth, defining the fraction of the CPU reserved to the task.

If a task is handled by a CBS with bandwidth U_s, it is guaranteed that it will never demand more than its reserved bandwidth, independently of its actual requests. This is achieved by assigning each task a suitable (dynamic) deadline, computed as a function of the reserved bandwidth and its actual requests. If a task requires to execute more than its expected computation time, its deadline is postponed by the server, so that its reserved bandwidth is never exceeded. As a consequence, overruns occurring in a served task will only delay that task, without stealing the bandwidth assigned to other tasks, which are then isolated and protected from reciprocal interference. In other words, the CBS acts as a filter that reshapes the served task in a more uniform way. Figure 1 illustrates an example in which three tasks are scheduled under a CPU reservation scheme.

Currently, reservation-based scheduling is focused on the CPU only. However, a system wide approach is demanded, including other system resources, including memory, disk, and network. Pertinent research directions include adding support for resource reservation schemes in embedded systems. Here, a major challenge is to contain computational overhead in the implementation. One of the few existing kernels for small embedded systems supporting resource reservation is Erika Enterprise [17], developed by Evidence S.r.l.

A further step in this area is providing a notion for real-time contracts that includes the rights and duties of the involved parties in a detailed way.

4.2 Efficient resource reclaiming

Although the resource reservation paradigm may solve many problems related to priority-based scheduling, its behavior heavily depends on a balanced allocation of the available resources to the application tasks. In fact, if the amount of resource reserved to a task is less than required, that task will slow down too much, decreasing system performance. Conversely, if the amount of resource reserved to a task is too high, resources are wasted and the system will run with low efficiency (increasing the overall cost of the application).

However, providing an exact estimation of the resources needed by each system activity is a non trivial job, which requires heavy execution tests and specific tools for code analysis. Nevertheless, the resulting estimations are usually affected by large errors (up to 20%). Therefore, an additional runtime reclaiming mechanism is required in the kernel to cope with wrong or imprecise resource reservations. The idea behind resource reclaiming is quite simple: whenever a task completes earlier than expected, so leaving part of a resource unused, this part is temporally kept in the system to be given to other tasks that may require more bandwidth.

It is worth observing, however, that resource reclaiming is an on-line mechanism that exploits early completions, hence it cannot always compensate for reservation errors. In the average, resource reclaiming is quite effective for compensating for small reservation errors, but it does not represent the solution for coping with large and systematic deviations in the execution behavior of computational activities. A few examples of reclaiming algorithms have been proposed in the literature [25, 9, 16, 23].

4.3 Integrating real-time and control

When application tasks have an extremely variable and unpredictable execution behavior, feedback control theory can be used to estimate the current workload conditions and perform proper parameter tuning [35]. Integration of real-time and control theory just begun to be studied and is a promising research area. The advantage of such an integration is twofold. From one hand, feedback control schemes can be used in the kernel to make the system more adap-

tive to unpredictable changes. On the other hand, real-time theory and schedulability analysis can be also considered during the design of control systems, to take into account jitter and delays introduced in control loops by resource contention and concurrent execution.

In the traditional approach to the analysis and design of computer control systems, controllers are assumed to execute in dedicated processors and these are assumed to be fast and predictable enough to meet all the application requirements. However, when resources such as processor time or network bandwidth are limited, the analysis and design of computer control systems is a challenging task: the resource limitations must be taken into account in the controller design stage, or the controlled system may exhibit unexpected behavior.

For example, the criteria for scheduling tasks on processors influences the timing of all tasks and can thus introduce timing variability (jitter) in the execution of control loops. These timing variations in the execution of control algorithms - which are allowed as long as the schedulability constraints are preserved - affect performance and possibly cause instability. This degradation appears because the controller execution violates the timing assumptions of classical discrete-time controller design theory, equidistant sampling and actuation.

On the other hand, trying to reduce jitter for control tasks by over-constraining the control task specification (e.g. by very tight deadlines) reduces the degradation of the controlled systems, but at the expense of finding feasible scheduling solutions for the entire task set.

These kinds of problems can be addressed using a combination of control and real-time scheduling principles. Instead of separating the two aspects during design, control design and computer implementation have to be jointly considered early in the design. A number of algorithms to combine real-time and control have been presented in the literature in both areas of research, with different perspectives and objectives.

4.4 Flexible Scheduling

Novel applications combine various types of tasks and constraints within the same system. The requirements on tasks may also change dynamically. While off-line guarantees are still essential for meeting minimum performance levels, different types of requirements and runtime changes are included in the system analysis, such as demands on quality of service (QoS) or acceptance probabilities. Algorithms might even change during system's runtime to better adapt to environment variations. In such a new scenario, the basic assumptions made on the classical scheduling theory are no longer valid. New approaches are needed to handle these situations in a predictable fashion. They should enforce timing constraints with a certain degree of flexibility, aimed at achieving the desired trade-off between predictable performance and efficient use of resources [14].

Flexible scheduling is an underlying theme to most novel scheduling trends which go beyond the standard model of completely known tasks with timing constraints expressed as periods and deadline. Many applications areas, notably for control and media processing, have timing requirements which cannot be expressed only by deadlines and periods. As a consequence, scheduling algorithms should consider more flexible ways of expressing temporal constraints, to meet the demands of application level requirements rather than system models.

Issues addressed include probabilistic parameters [1, 12], handling of applications with only partially known properties [22], relaxed constraints, coexistence of activities with diverse properties and demands in a system, combinations of scheduling schemes, and adaptive scheduling schemes.

While individual algorithms have been proposed to adapt scheduling parameters to changes in application demands, a systematic approach is needed to identify changes in the application, distinguish them between temporary and structural variations, and adjust the scheduling parameters to determine a proper system response.

4.5 Hierarchical scheduling

Today's computers are powerful enough to execute multiple applications at the same time. This may require partitioning the processor into several "virtual" machines, each with a proper fraction of computation power and scheduling algorithm. When different scheduling schemes are demanded in the same computer, the analysis of the entire system becomes complex and more theoretical work is needed for providing guarantee tests of multiple concurrent applications.

Hierarchical scheduling means that there is not just one scheduling algorithm for a given resource, but a hierarchy of schedulers. The tasks in the system are hierarchically grouped. The root of the hierarchy is the complete system; the leaves of the hierarchy are the individual tasks. At each node in the hierarchy, a scheduling algorithm schedules the children of that node. The practical value of a two level-hierarchy is immediately obvious: intermediate nodes are applications that are independently developed and/or independently loaded. If the root scheduler provides guaranteed resource reservation and temporal isolation, the applications can (with some additional precautions) be viewed to be running on private processors. There are two very distinct real-time processing domains where some form of hierarchical scheduling is proposed: one is the area of soft real-time in personal computers; the other is the area of cer-

tified hard real-time systems. In the first domain, several frameworks [18, 19, 3, 30, 36] have been proposed for deterministic soft real-time scheduler composition. In the second domain, ARINC [5] proposes a root scheduler that provides time slots in a recalculated schedule. In other proposals, the root scheduler provides some form of guaranteed bandwidth allocation [24, 28].

4.6 Overload handling

Predictability in dynamic systems is strictly related to the capability of controlling the incoming workload to prevent overload conditions. In fact, when the computation exceeds the processor capabilities, breakdown phenomena may cause abrupt performance degradation. Computational workload can be controlled using different techniques, each requiring deeper investigation.

- *Selection of different QoS levels*. Some computations can be performed using different algorithms with different execution requirements, hence leading to results with different quality. In other cases, the precision (hence, the quality) of a result can be enhanced by increasing the number of steps of an iterative solution. Hence, the workload can be controlled by selecting the proper quality level for each system activity.
- *Adjustable timing constraints*. In a real-time system, the workload depends not only on the amount of computation arriving per each unit of time, but also on the timing constraints associated with the computations. Hence, another way to react to overloads is to relax the timing constraints of the application tasks in the presence of high computational requirements. For periodic tasks, the load can be reduced by enlarging the periods [13], but more work is needed to deal with generic task sets with arbitrary deadlines.
- *Admission control*. A third way to control the load is to filter the incoming requests of computation. This solution is the most drastic one, because it solves the overload by rejecting one or more tasks. However, additional research is needed to evaluate the effect of a rejection on the overall system performance.

4.7 Energy-aware scheduling

In battery-powered devices, reducing energy consumption is crucial for increasing system lifetime. Modern processors can operate at variable voltage/frequency levels for trading performance vs. energy consumption. In real-time systems, however, decreasing voltage prolongs task execution and may cause deadline misses; hence, future scheduling algorithms must take voltage into account to meet timing constraints while minimizing energy consumption.

The majority of scheduling algorithms with energy considerations mainly concentrates on the CPU, integrating dynamic voltage scaling into the scheduling problem. However, the CPU is only one resource consuming power. Other components, like memory, disk, communication devices, and input/output peripherals, are not yet considered by the theory. Hence, new approaches for a system wide energy view are needed. In particular, the "energy overhead" of the scheduling algorithm with respect to CPU and memory should also be considered.

Applying dynamic voltage scaling techniques to control energy consumption causes also other problems in real-time systems. In fact, in the presence of timing and resource constraints, the performance of a real-time system does not always improve as the speed of the processor is increased. Similarly, when reducing the processor speed, the quality of the delivered service may not always degrade as expected [15]. To prevent these problems, new approaches need to be investigated to allow the development of real-time applications whose performance can be scaled in a controlled fashion as a function of the processor speed.

4.8 Portability

With the constant evolution of hardware, portability is also a very important issue, necessary to run applications developed for a particular platform into new hardware platforms. The use of standard programming interfaces opens the door to the possibility of having several operating system providers for a single application, and promotes competition among vendors, thus increasing quality and value.

Current operating system standards mostly specify portability at the source code level, requiring the application developer to recompile the application for every different platform. There are four main operating system standards available today:

- RT-POSIX, which is the main general-purpose operating system standard, with real-time extensions;
- OSEK, for the automotive industry;
- APEX, for avionics systems;
- μITRON, for small embedded systems.

However, extensions are needed in these standards to support application-defined scheduling services and facilitate the evolution from fixed-priority scheduling towards more flexible scheduling algorithms. This additional flexibility is necessary to provide better support to systems with quality of service requirements, even though it is expected that most of the services required by these systems will continue to be implemented in a specialized middleware layer.

4.9 Modeling

Modelling plays a central role in systems engineering. The existence of modelling techniques is a basis for rigorous design and should drastically simplify validation. In current industrial practice, models are essentially used only at the early phases of system design and at a high level of abstraction. Requirements and design constraints are spread out and they do not easily carry through the entire development lifecycle. Validation of large real-time applications is mainly done by experimentation and measurement on specific platforms, in order to adjust design parameters and, hopefully, achieve conformity with requirements. Thus, experimenting with different architectures and execution platforms becomes error-prone.

The use of models can profitably replace experimentation on actual systems with incomparable advantages, such as:

- Enhanced modifiability of the system parameters;
- Ease of construction by integration of models of heterogeneous components;
- Generality by using abstraction and behavioural nondeterminism;
- Predictability analysis by application of formal methods.

Modelling methodologies should be closely related to implementation methodologies for building correct real-time systems as a sequence of steps involving both the development of software components and their integration in an execution and communication platform. These methodologies should support end-to-end constraints at every step in the design process and provide means to automatically propagate them down to the implementation. To be useful in practice, they should be accompanied by the development of appropriate middleware, QoS management support, and validation tools.

Modelling systems in the large is an important research topic in both academia and industry [33]. Several trends can be identified in this area:

- One line of research consists of the so-called model-based approaches. This research groups the study of unified frameworks for integrating different models of computation [21], languages [20] and abstraction-based design methodologies [11].
- A key issue in a modelling methodology is the use of adequate operators to compose heterogeneous schedulers (e.g., synchronous, asynchronous, event-triggered, or time-triggered). For this reason, some researchers propose model-based theories for composing scheduling policies [4].
- Another challenge consists in adequately relating the functional and non-functional requirements of the application software with the underlying execution platform. There are two current approaches to this problem:
  1. One relies on architecture description languages that provide means to relate software and hardware components (e.g., MetaH [10]).
  2. The other is based on the formal verification of automata-based models automatically generated from software and appropriately annotated with timing constraints (e.g., Taxys [8, 34]).
- Nevertheless, building models that faithfully represent real-time systems is not a trivial problem and still requires a great amount of theoretical and experimental research.

4.10 Towards component-based RTOS

Another promising research area is to adopt a component-based design paradigm at the kernel level. The use of component-based operating systems would allow the designer to quickly configure the kernel for a specific application just by combining existing modules, thus speeding up the development process and optimizing efficiency for the required functionality.

Today, software modularity allows the same system to be assembled in several incremental configurations, with different features and functionality. For example, the POSIX standard, specifies four real-time profiles [29]:

- **Minimal Real-Time System profile (PSE51)**. This profile is intended for small embedded systems and can be implemented with a few thousand lines of code and with memory footprints in the tens of kilobytes range. Processes are not supported and there is not a complete file system (input/output is possible through predefined device files).
- **Real-Time Controller profile (PSE52)**. It is similar to the PSE51 profile, with the addition of a file system in which regular files can be created, read, or written. It is intended for systems like a robot controller, which may need support for a simplified file system.
- **Dedicated Real-Time System profile (PSE53)**. It is intended for large embedded systems (e.g., avionics) and extends the PSE52 profile with the support for multiple processes that operate with protection boundaries.
- **Multi-purpose Real-Time System profile (PSE54)**. It is intended for general-purpose computing systems

running applications with real-time and non-real-time requirements. It requires all of the POSIX functionality for general purpose systems and, in addition, most of the real-time services.

However, this level of modularity does not allow the user to replace an internal kernel mechanism with another one with the same interface, nor replacing a mechanism without changing the application. This happens because all kernel mechanisms have strong inter-dependencies and are usually developed on the basis of other internal features. For example, typical implementations of the Priority Inheritance Protocol [32] strongly rely on a fixed priority scheduler and cannot be used under deadline-based scheduling algorithms.

A true component-based approach should separate mechanisms from policies in order to replace a scheduling algorithm or a resource management protocol without affecting the applications and the others components. In addition, it should allow a safe combination of different scheduling disciplines to support the development of hierarchical software architectures.

There would be several benefits in adopting a component-based approach at the operating system level. First of all, it would be possible to enhance the functionality of the kernel by adding new blocks, depending of the application requirements, so tailoring the kernel to the specific system to be developed. Secondly, it would facilitate and speed up the integration of novel research results, which could increase efficiency and/or predicability. Finally, it would simplify the process of porting the kernel on different platforms, so reducing the time to market and the development costs on upgrades (since only small parts should be developed). However, there are several practical and theoretical problems to be solved, since most of the mechanisms implemented in a kernel (like scheduling, resource protocols, interrupt handling, aperiodic servers, synchronization and communication) heavily interact with each other and have a high degree of inter-dependencies.

Specific research in this area should provide methods for decoupling scheduling algorithms from applications, separating scheduling mechanisms from scheduling policies, separating scheduling algorithms from resource management protocols, and, finally, guaranteeing a safe integration of resource reservation with resource management protocols.

5 Conclusions

In the last 30 years, embedded systems experienced an exponential growth in many application domains, both in terms of number and complexity. Surprisingly, however, such a growth in complexity was not followed by a corresponding evolution of the control software used to manage the computational resources, which is substantially similar to that adopted in the early 70s. In fact, application activities are still handled by cyclic executives or, in the best case, by fixed priority kernels. The problem is not due to a lack of alternatives, but more that nobody has been able to make a convincing case for a transition. Every attempt to raise the level of abstraction has included unacceptable penalties in terms of memory and speed. Also from an industrial perspective, the support for legacy code has often been weak.

The possibilities for embedded systems to evolve and become more reliable, while yet more complex, to some extent depend on what the next generation real-time operating systems and implementation tools have to offer. The challenge is how to implement applications that can execute efficiently on limited resources, to meet non-functional requirements, such as timeliness, robustness, dependability, and performance.

Proper resource and quality-of-service management would enable the implementation of embedded systems that are more flexible, yet more deterministic, than it is possible today. Since such systems would be better specified, their properties would also be verified more easily. By supporting explicit resource allocation and quality-of-service functionality, the system designers would regain control over the system they are set to design.

To effectively assign system resources among applications and achieve predictability and flexibility, a number of issues should be further investigated. At the higher abstraction level, protocols for managing quality levels and suitable architectures should be used to obtain flexible systems. At a lower level, further work on resource management algorithms, new task models, admission control, monitoring, and adaptation algorithms should be done.

Also, a promising research area consists in developing hybrid methods, which integrate two complementary types of adaptation strategies: one embedded in the application and the other performed by a QoS manager. Such an integration can be done by controlling the CPU bandwidth reserved to a task, but allowing each task to change its QoS requirements if the amount of reserved resources is not sufficient to accomplish the goal within a desired deadline. Using such an integrated approach, the QoS adaptation is performed in a task-specific fashion: each task can react to overloads in a different way and use different techniques to scale down its resource requirements. On the other hand, if a task does not react adequately to a lack of resources, the scheduler will slow it down in order not to influence the other tasks.

As a conclusion, the following recommendations for research emerge in the area of real-time operating systems for embedded applications:

- Flexible scheduling services. The complexity of modern applications requires more flexibility in schedul-

ing than just fixed priorities. Defining APIs that could make the scheduler a pluggable and interchangeable object seems the most promising research direction.

- Protection. One way of managing the complexity of applications is by providing appropriate levels of protection, both in space (memory) and time. The time protection mechanisms specified in some standards, like OSEK or ARINC, are somehow too rigid, and there are needs to make this protection more flexible but still effective.

- Dynamicity. The complexity of applications requires moving from statically designed applications to a more dynamic environment where the application components can be changed on-line. Research is needed on methods and new APIs are needed for effective on-line admission tests and dynamic resource reservation.

- Quality of service. There is a need for middleware that allows the application to define quality of service requirements, using some contract mechanism that lets the application specify its minimum and desired requirements, so if the implementation accepts the contract it can guarantee the minimum requirements and try to provide the desired ones. To implement this middleware, there is a need to develop techniques and APIs at the operating system level to perform load adaptation, monitoring and budgeting of the system resources.

- Multiprocessor support. Predictability of the timing behavior in multiprocessor systems is still a research issue. Most multiprocessor real-time systems today require static allocation of threads to processors.

- Drivers. Portability of drivers for real-time applications is an open issue. There is a need for extending current APIs for portable drivers to support real-time requirements.

- Networks. There are few real-time networks and protocols, and the support for them in operating systems is very limited. There is a need to develop protocol-independent APIs that let a distributed application define its timing requirements for the network and the remote services.

- Modeling. There is a need to develop precise models of the timing behavior of the operating system services, which could be used in timing analysis tools. It would be useful to have automatic procedures to obtain the timing model of any operating system on a given platform.

References

[1] L. Abeni and G. Buttazzo. “Stochastic Analysis of a Reservation Based System,” Proceedings of the 9th International Workshop on Parallel and Distributed Real-Time Systems, San Francisco, April 2001.

[2] L. Abeni and G. Buttazzo, “Resource Reservation in Dynamic Real-Time Systems,” *Real-Time Systems*, Vol. 27, No. 2, pp. 123-167, July 2004.

[3] M. Aldea-Rivas and M. Gonzalez-Harbour, “POSIX-compatible application-defined scheduling in MARTE OS,” Proceedings of the 14th Euromicro Conference on Real-Time Systems (ECRTS02), Vienna, Austria, 2002.

[4] K. Altisen, G. Goessler, and J. Sifakis. “Scheduler modeling based on the controller synthesis paradigm,” Journal of Real-Time Systems, special issue on Control Approaches to Real-Time Computing, 23:55–84, 2002.

[5] “ARINC 651: Design Guidance for Integrated Modular Avionics,” pub. by Airlines Electronic Engineering Committee (AEEC), November 1991.

[6] A. K. Atlas and A. Bestavros. “Statistical Rate Monotonic Scheduling,” Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS 1998), Madrid, Spain, Dec. 1998.

[7] H. Aydin, R. Melhem, D.Moss and Pedro Mejia Alvarez “Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics,” IEEE Proceedings of the 13th Euromicro Conference on Real-Time Systems (ECRTS 2001), Delft, The Netherlands, June 2001.

[8] V. Bertin, E. Closse, M. Poize, J. Pulou, J. Sifakis, P. Venier, D. Weil, and S. Yovine. “Taxys = Esterel + Kronos: a tool for verifying real-time properties of embedded systems,” Proceedings of the Conference on Decision and Control (CDC 2001), Orlando, December 2001.

[9] G. Bernat and A. Burns, “Multiple Servers and Capacity Sharing for Implementing Flexible Scheduling,” Real-Time Systems Journal, Special Issue on Flexible Scheduling. Vol 22, pp. 49–75, 2002.

[10] P. Binns and S. Vestal. “Formalizing software architectures for embedded systems,” Proceedings of the First Int. Conference on Embedded Software (EMSOFT 2001), Tahoe City, California, Springer, LNCS 2211, October 2001.

[11] J.R. Burch, R. Passeronne, and A. Sangiovanni-Vincentelli. “Using multiple levels of abstractions in embedded software design,” Proceedings of the First Int. Conference on Embedded Software (EMSOFT 2001), Tahoe City, California, Springer, LNCS 2211, October 2001.

[12] A. Burns, G. Bernat, I. Broster, “A Probabilistic Framework for Schedulability Analysis,” Proceedings of the Third International Conference on Embedded Software (EMSOFT 2003), Philadelphia, Pennsylvania, USA, October 2003.

[13] G. C. Buttazzo, G. Lipari, M. Caccamo, L. Abeni, “Elastic Scheduling for Flexible Workload Management,” *IEEE Transactions on Computers*, Vol. 51, No. 3, pp. 289-302, March 2002.

[14] G. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo, Soft Real-Time Systems: Predictability vs. Efficiency, Springer, 2005.

[15] G. Buttazzo, "Achieving Scalability in Real-Time Systems," IEEE Computer, Vol. 39, No. 5, pp. 54–59, May 2006.

[16] M. Caccamo, G. Buttazzo, and D. C. Thomas, "Efficient Reclaiming in Reservation-Based Real-Time Systems with Variable Execution Times," IEEE Transactions on Computers, Vol. 54, No. 2, pp. 198–213, February 2005.

[17] Evidence Srl, "ERIKA Enterprise RTOS", URL: http://www.evidence.eu.com.

[18] P. Gai, L. Abeni, M. Giorgi, and G. Buttazzo, "A New Kernel Approach for Modular Real-Time Systems Development," IEEE Proceedings of the 13th Euromicro Conference on Real-Time Systems (ECRTS 2001), Delft, The Netherlands, June 2001.

[19] P. Goyal, X. Guo, and H. M. Vin, "A hierarchical CPU scheduler for multimedia operating systems," Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation, Seattle, Washington, October 1996.

[20] T.A. Henzinger, B. Horowitz, and C. Meyer Kirsch, "Giotto: A time-triggered language for embedded programming," Proceedings of the First Int. Conference on Embedded Software (EMSOFT 2001), Tahoe City, California, Springer, LNCS 2211, October 2001.

[21] E. Lee and A. Sangiovanni-Vincentelli. "A unified framework for comparing models of computation," IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 17(12):1217–1229, December 1998.

[22] A. Leulseged and N. Nissanke., "Probabilistic Analysis of Multi-processor Scheduling of Tasks with Uncertain Parameters," Proceedings of the 9th Int. Conference on Real-time and Embedded Computing Systems and Applications (RTCSA 2003), Tainan, Taiwan, February 2003.

[23] C. Lin and S. A. Brandt, "Improving Soft Real-Time Performance Through Better Slack Management," Proceedings of the IEEE Real-Time Systems Symposium (RTSS 2005), pp. 314, Miami, Florida, December 58, 2005.

[24] G. Lipari and S. K. Baruah "Efficient Scheduling of Multi-Task Applications in Open Systems," IEEE Proceedings of the 6th Real-Time Systems and Applications Symposium (RTAS 2000), Washington DC, June 2000.

[25] G. Lipari and S. K. Baruah "Greedy reclaimation of unused bandwidth in constant bandwidth servers," IEEE Proceedings of the 12th Euromicro Conference on Real-Time Systems (ECRTS 2000), Stokholm, Sweden, June 2000.

[26] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment," Journal of the ACM, No. 1, Vol. 20, pp. 40-61, 1973.

[27] C. W. Mercer, S. Savage, and H. Tokuda, "Processor Capacity Reserves: Operating System Support for Multimedia Applications," Proceedings of IEEE International Conference on Multimedia Computing and Systems, Boston (MA), USA, May 1994.

[28] A. Mok, X. Feng, and D. Chen, "Resource Partition for Real-Time Systems," Proceedings of the 7th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2001), Taipei, Taiwan, June 2001.

[29] IEEE Standard 1003.13-2003, Standard for Information Technology -Standardized Application Environment Profile-POSIX Realtime and Embedded Application Support (AEP). The Institute of Electrical and Electronics Engineers, 2003.

[30] J. Regehr and J. Stankovic, "HLS: a framework for composing soft real-time systems," Proceedings of the Real-Time Systems Symposium (RTSS 2001), London, December 2001.

[31] B. Bouyssounouse and J. Sifakis (Editors), Embedded Systems Design, The ARTIST Roadmap for Research and Development, Lecture Notes in Computer Science, Vol. 3436, Springer, 2005.

[32] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, 39(9), pp. 1175-1185, 1990.

[33] J. Sifakis. "Modeling real-time systems – challenges and work directions," Proceedings of the Int. Conference on Embedded Software (EMSOFT 2001), Springer, LNCS 2211, 2001.

[34] J. Sifakis, S. Tripakis, S. Yovine. "Building models of real-time systems from application software," Proceedings of the IEEE, Special issue on modeling and design of embedded, 91(1):100-111, January 2003.

[35] J. A. Stankovic, C. Lu, S. H. Son, and G. Tao, "The Case for Feedback Control Real-Time Scheduling," Proceedings of the 11th EuroMicro Conference on Real-Time Systems (ECRTS 1999), York, UK, June 1999.

[36] S. Wang, K.-J. Lin, and Y. Wang, "Hierarchical budget management in the RED-linux scheduling framework," Proceedings of the 14th Euromicro Conference on Real-Time Systems (ECRTS 2002), pp. 76-83, Vienna, Austria, 2002.