CONTROL AND EMBEDDED COMPUTING: SURVEY OF RESEARCH DIRECTIONS

Karl-Erik Årzén and Anton Cervin

Department of Automatic Control, Lund University,
Box 118, SE-221 00 Lund, Sweden

Abstract: This paper provides a survey of the role of feedback control in embedded real-time systems, presented in the context of ARTIST2, the EU/IST network of excellence on design of embedded systems. The survey highlights recent research efforts and future research directions in the areas of codesign of computer-based control systems, implementation-aware embedded control systems, and control of real-time computing systems.

Keywords: computer control, embedded systems, computer systems

1. INTRODUCTION

The current pervasive and ubiquitous computing trend has increased the emphasis on embedded and networked computing within the engineering community. Already today, embedded computers by far outnumber desktop computers. Embedded systems are often found in consumer products and are therefore subject to hard economic constraints. Some examples are automotive systems and mobile phones. The pervasive nature of these systems generates further constraints on physical size and power consumption. These product-level constraints give rise to resource constraints on the computing platform level, for example, limitations on computing speed, memory size, and communication bandwidth. Due to economic considerations, this is true in spite of the fast development of computing hardware. In many cases, it is not economically justified to add an additional CPU or to use a processor with more capacity than what is required by the application. Cost also favors general-purpose computing components over specially designed hardware and software solutions.

Although in many application areas embedded systems are becoming smaller and smaller, there are also other examples in which the requirements on functionality of embedded systems increases substantially. The resulting complexity, in particular on the software side, makes it necessary to employ implementation techniques normally not associated with embedded systems. Object-oriented programming languages such as Java and C# and distributed middleware such as CORBA are increasingly being considered for embedded applications. One example is the Golden Gate project where Real-Time Java in the form of RTSJ (Real-Time Specification for Java) will be used for the implementation of the next autonomous Mars rover (Dvorak *et al.*, 2004).

Control systems constitute an important subclass of embedded computing systems. So important that, for example, within automotive systems, computers commonly go under the name electronic control units (ECUs). A top-level modern car contains more than 50 ECUs of varying complexity. A majority of these implement different feedback control tasks, for instance, engine control, traction control, anti-lock braking, active stability control, cruise control, and climate control.

Even fairly simple computer-based control systems are becoming increasingly complex from both the control and computer science perspectives. Today, even small embedded control systems often contain a multitasking real-time kernel and support networking. Many computer-controlled systems are distributed, consisting of computer nodes and a communication network connecting the various systems. It is not un-

common for the sensor, actuator, and control calculations to reside on different nodes. This gives rise to networked control loops. Within individual nodes, controllers are often implemented as one or several tasks on a microprocessor with a real-time operating system. Often, the microprocessor also contains tasks for other functions, such as communication and user interfaces. The operating system typically uses multiprogramming to multiplex the execution of the various tasks. The CPU time and the communication bandwidth can hence be viewed as shared resources for which the tasks compete.

At the same as control systems is an important application class of embedded systems, feedback control is also an important basic technology that can be employed in the design of embedded real-time systems. Over time applications of real-time computing have gradually evolved from closed embedded systems to complex, distributed open heterogeneous platforms operating in unpredictable poorly modeled environments such as, e.g., the Internet and, in the case of sensor networks, the physical world. Hard guarantees are impractical on such platforms since load and resource capacities are very difficult to predict. Yet, many modern applications require some form of performance assurances, which may include guarantees on timeliness, bandwidth, data consistency, or jitter. Traditional approaches for providing these performances, such as resource pre-allocation and a priori knowledge of worst case execution conditions are no longer applicable. Feedback control is a well-established and mathematically well-founded theory that is ideal for handling uncertainties. Using control-based approaches for modeling, analysis, and design of real-time computing systems is currently receiving increased attention as a promising foundation for controlling the uncertainty in large and complex real-time systems. The approach is sometimes called feedback scheduling. Areas that currently are being investigated are dynamic models of real-time computing systems and control of timing behavior, e.g., delays.

The aim of this paper is provide a survey of the role of feedback control in embedded real-time systems. The background for the survey is the recently started EU/IST Network of Excellence ARTIST2 (see http://www.artist-embedded.org/FP6/Overview) on design of embedded systems in which control and embedded systems is one of the clusters. The aim of the survey is to highlight the relevant research directions in the areas of codesign of computer-based control systems, implementation-aware embedded control systems, and control of real-time computing systems.

Outline of the paper

The codesign problem is discussed in Section 2. Temporal determinism in control is discussed in Section 3. Section 4 discusses two alternative approaches to control system design, the hard real-time approach and the soft control approach. Codesign tools that allow the designer to analyze and simulate how control loop timing affects control performance are the topic of Section 5. Special emphasis is given to the TrueTime and Jitterbug tools. Section 6 discusses feedback scheduling with a special emphasis on feedback scheduling of control tasks. Finally, in Section 7 the most important research directions within the field are identified.

2. THE CODESIGN PROBLEM

By tradition, the design of computer-based control systems is based on the principle of *separation of concerns*. This separation is based on the assumption that feedback controllers can be modeled and implemented as periodic tasks that have a fixed period, T, a known worst-case bound on the execution time (WCET), C, and a *hard deadline*, D. The latter implies that it is imperative that the tasks always meet their deadlines, i.e., that the actual execution time (response time) is always less or equal to the deadline, for each invocation of the task. This is in contrast to a *soft deadline*, that may occasionally be violated. The fixed-period assumption of the simple task model has also been widely adopted by the control community and has resulted in the development of the sampled computer-control theory with its assumption on deterministic, equidistant sampling. The separation of concerns has allowed the control community to focus on the pure control design without having to worry about how the control system eventually is implemented. At the same time, it has allowed the real-time computing community to focus on development of scheduling theory and computational models that guarantee that hard deadlines are met, without any need to understand what impact scheduling has on the stability and performance of the plant under control.

Historically, the separated development of control and scheduling theories for computer-based control systems has produced many useful results and served its purpose well. However, the separation has also had negative effects. The two communities have partly become alienated. The assumptions of the simple model are also overly restrictive with respect to the characteristics of many control loops. Many control loops are not periodic, or they may switch between a number of different fixed sampling periods. Control loop deadlines are not always hard. On the contrary, many controllers are quite robust to variations in sampling period and response time. Hence, it is questionable whether it is necessary to model them as hard-deadline tasks.

The main drawbacks with the separations of concerns are that it does not always utilize the available computing resources in an optimal way, and that it sometimes gives rise to worse control performance than

what can be achieved if the design of the control and real-time computing parts are integrated. This is particularly important for embedded control applications with limited computing and communication resources, with demanding performance specifications and high requirements on flexibility. For these types of applications, better performance can be achieved if a codesign approach is adopted where the control system is designed taking the resource constraints into account and where the real-time computing and scheduling is designed with the control performance in mind. The resulting *implementation-aware control systems* are better suited to meet the requirements of embedded and networked applications.

The *control and scheduling codesign problem* can be informally stated as follows (in the uniprocessor case): "Given a set of processes to be controlled and a computer with limited computational resources, design a set of controllers and schedule them as real-time tasks such that the overall control performance is optimized." An alternative view of the same problem is to say that we should design and schedule a set of controllers such that the least expensive implementation platform can be used while still meeting the performance specifications. For distributed systems, the scheduling is extended to also include the network communication.

The nature and the degree of difficulty of the codesign problem for a given system depend on a number of factors:

- *The real-time operating system.* What scheduling algorithms are supported? How is I/O handled? Can the real-time kernel measure task execution times and detect execution overruns and missed deadlines?
- *The scheduling algorithm.* Is it time-driven or event-driven, priority-driven or deadline-driven? What analytical results regarding schedulability and response times are available? What scheduling parameters can be changed on-line? How are task overruns handled?
- *The controller synthesis method.* What design criteria are used? Are the controllers designed in the continuous-time domain and then discretized or is direct discrete design used? Are the controllers designed to be robust against timing variations? Should they actively compensate for timing variations?
- *The execution-time characteristics of the control algorithms.* Do the algorithms have predictable worst-case execution times? Are there large variations in execution time from sample to sample? Do the controllers switch between different internal modes with different execution-time profiles?
- *Off-line or on-line optimization.* What information is available for the off-line design and how accurate is it? What can be measured on-line? Should the system be able to handle the arrival of new tasks? Should the system be re-optimized when the workload changes? Should there be feedback from the control performance to the scheduling algorithm?
- *The network communication.* Which type of network protocol is used? Can the protocol provide worst-case guarantees on the network latency? How large is the probability of lost packets?

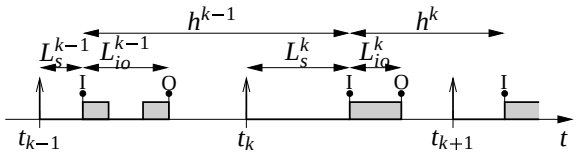

Fig. 1. Controller timing.

Codesign of control and computing systems is not a new topic. Control applications were one of the major driving forces in the early development of computers. Then, limited computer resources was a general problem, not only a problem for embedded controllers. For example, the issues of limited word length, fixed-point calculations, and the results that this has on resolution were something that was well-known among control engineers in the 1970s. However, as computing power has increased, these issues have received less attention. A nice survey of the area from the mid 1980s is (Hanselmann, 1987).

3. TEMPORAL DETERMINISM

Computer-based control theory normally assumes equidistant sampling and negligible, or constant, input-output latencies. However, this can seldom be achieved in practice or is too costly for a particular application. In a multi-threaded system, tasks interfere with each other due to preemption and blocking from task communication. Execution times may be data-dependent or vary due to the use of caches. In networked control loops, where the sensors, controllers, and actuators reside on different physical nodes, the communication gives rise to latencies that can be more or less deterministic, depending on the network protocols used. The result of all this is jitter in sampling intervals and non-negligible and varying latencies.

The basic timing parameters of a control task are shown in Fig. 1. It is assumed that the control task is *released* (i.e., inserted into the ready queue of the real-time operating system) periodically at times given by $t_k = hk$, where h is the *nominal sampling interval* of the controller. Due to preemption and blocking from other tasks in the system, the actual *start* of the task may be delayed for some time L_s. This is called the *sampling latency* of the controller. A dynamic scheduling policy will introduce variations in this interval. The *sampling jitter* can be quantified by the difference between the maximum and minimum sampling latencies in all task instances,

$$J_s \stackrel{\text{def}}{=} \max_k L_s^k - \min_k L_s^k. \quad (1)$$

Normally, it can be assumed that the minimum sampling latency of a task is zero, in which case we have $J_s = \max_k L_s^k$. Jitter in the sampling latency will also introduce jitter in the sampling interval h. The actual sampling interval in period k is given by

$$h^k = h - L_s^{k-1} + L_s^k. \quad (2)$$

After some computation time and possibly further preemption from other tasks, the controller will actuate the control signal. The delay from the sampling to the actuation is called the *input-output latency*, denoted L_{io}. Varying execution times or task scheduling will introduce variations in this interval. The *input-output jitter* can be quantified by

$$J_{io} \stackrel{\text{def}}{=} \max_k L_{io}^k - \min_k L_{io}^k. \quad (3)$$

Control loop timing issues are further discussed in (Wittenmark *et al.*, 1995; Törngren, 1998).

It is well known that a constant input-output latency decreases the phase margin of the control system, and that it introduces a fundamental limitation on the achievable closed-loop performance. The resulting sampled-data system is time-invariant and of finite order, which allows standard linear time-invariant (LTI) analysis to be used (see e.g., (Åström and Wittenmark, 1997)). For a given value of the latency, it is easy to predict the performance degradation due to the delay. Furthermore, it is straightforward to account for a constant latency in most control design methods. From this perspective, a constant input-output latency is preferable over a varying latency.

The scheduling-induced input-output latency of a single control task can be reduced by assigning it a higher priority (or, alternatively, under deadline-based scheduling, a shorter deadline). This approach will of course not work for the whole task set. Another option is to use non-preemptive scheduling. This will guarantee that, once the task has started its execution, it will continue uninterrupted until the end. The disadvantages of this approach are that the scheduling analysis for non-preemptive scheduling is quite complicated (e.g., (Klein *et al.*, 1993; Stankovic *et al.*, 1998)), and that the schedulability of the other tasks may be compromised. However, as computing speed increases, and, hence, $C \ll T$, it becomes increasingly interesting to execute tasks with hard deadlines non-preemptively.

A standard way to achieve a short input-output latency in a control task is to separate the algorithm calculations in two parts: *Calculate Output* and *Update State*. Calculate Output contains only the parts of the algorithm that make use of the current sample information. Update State contains the update of the controller states and precalculations for the next period. Update State can therefore be executed after the output signal transmission, hence, reducing the input-output latency. Further improvements can be obtained by scheduling the two parts as subtasks with different priorities, see (Cervin, 1999).

Another approach is to try to implement the control system in such a way that the control and scheduling subsystems are isolated, such that scheduling effects are not affecting control system performance. In the seminal Liu and Layland paper (Liu and Layland, 1973) on CPU scheduling, it is assumed that I/O is performed periodically by hardware functions, introducing a one-sample delay in all control loops closed over the computer. This scheme does provide a quite nice separation between scheduling and control design. From a scheduling perspective, the controller can be described by a periodic task with a period T, a computation time C, and a deadline $D = T$. From a control perspective, the controller will have a sampling period of T and a constant latency $L = T$. This allows the control design and the real-time design to be carried out in relative isolation. A similar separation has been suggested in the embedded systems programming model Giotto (Henzinger *et al.*, 2003). In Giotto, I/O and communication are time-triggered and assumed to take zero time , while the computations in between are assumed to be scheduled in real-time.

A drawback with the Giotto approach is that a minimum of one sample input-output latency is introduced in all control loops. Also if the one sample latency is compensated for in the control design, the achievable performance is generally worse than what is obtained with a shorter, albeit time-varying, latency. This problem is remedied in the Control Server (Cervin and Eker, 2003). A control server creates the abstraction of a control task with a specified period and a fixed input-output latency shorter than the period. Individual tasks can be combined into more complex components without loss of their individual guaranteed fixed latency properties. I/O occurs at fixed predefined points in time, at which inputs are read or controller outputs become visible. The single parameter linking the scheduling design and the controller design is the task utilization factor. The proposed server is an extension of the constant bandwidth server (Abeni and Buttazzo, 1998).

A control system with a time-varying input-output latency is quite difficult to analyze, since the standard tools for LTI systems cannot be used. If the statistical properties of the latency are known, then theory from jump linear systems can be used to evaluate the stability and performance of the system (in the mean sense), see (Nilsson *et al.*, 1998). Often, it is not possible to have exact knowledge of the input-output latency distribution. A simple, sufficient stability test for systems where only the range of the latency is known is given in (Kao and Lincoln, 2004). Assuming zero sampling jitter, the test can guarantee stability for *any* input-output latencies in a given interval (whether they are time-varying, dependent, etc).

4. DESIGN APPROACHES

The temporal non-determinism caused by the implementation platform can be approached in two different ways: the hard real-time approach or the soft control-based approach.

The hard real-time approach strives to maximize the temporal determinism by using special purpose hardware, software, and protocols. This includes techniques such as static cyclic scheduling, time-triggered computing and communication (Kopetz and Bauer, 2003), synchronous programming languages (Benveniste and Berry, 1991), and computing models such as Giotto. This approach has several advantages, specially for safety-critical applications. For example, it simplifies attempts at formal verification. The approach also has drawbacks. The approach has strong requirements on the availability of realistic worst-case bounds on resource utilization, something which is very difficult to obtain in practice. A result of this could be under-utilization and, possibly, poor control performance due to too long sampling intervals. The approach also makes it difficult to use general-purpose implementation platforms. This is particularly serious, since it is these systems that have the most advantageous price-performance development.

The soft, control-based approach instead views the temporal nondeterminism caused by the implementation platform as an uncertainty or disturbance acting on the control loop and handles it using control-based approaches. This can be done using a number of techniques. The simplest way is to rely on the inherent robustness of feedback. It is well-known that feedback increases the robustness towards plant variations. The same holds for variations caused by the implementation platform, i.e., *temporal robustness*. Another approach to deal with jitter in the control design is to explicitly design the controller to be robust, i.e., treat the delay as a parametric uncertainty. Many robust design methods are available, such as H_∞, quantitative feedback theory (QFT), and μ-design. The majority of these methods are developed for plant uncertainties. Although parts of the results carry over to temporal robustness it is likely that there is room for much more research here.

It is also possible to let the controller actively compensate for the delay in each sample. This can be compared to traditional gain-scheduling and feedforward from disturbances. An optimal, jitter-compensating controller was developed in (Nilsson *et al.*, 1998). The controller compensates for time-varying delays in a control loop, which is closed over a communication network. The setup is shown in Fig. 2. The sensor node samples the process periodically, sending the measurements over the network to the controller node. The controller node is event-driven and computes a new control signal as soon as a measurement arrives. The control signal is sent to the event-driven actuator

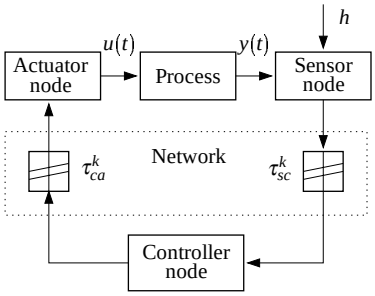

Fig. 2. Distributed digital control system with network communication delays τ_{sc}^k and τ_{ca}^k.

node, which outputs the signal to the process. The LQ (linear-quadratic) state feedback control law has the form

$$u(k) = -L(\tau_{sc}^k) \begin{bmatrix} x(k) \\ u(k-1) \end{bmatrix}, \qquad (4)$$

where the feedback gain L depends on the sensor-to-controller delay τ_{sc}^k in the current sample. The computation of the gain vector L is quite involved and requires that the probability distributions of τ_{sc} and τ_{ca} are known. The state feedback can be combined with an optimal state observer that takes the actual delays into account.

The above approach cannot be directly applied to scheduling-induced delays. The problem is that the delay in the current sample will not be known until the task has finished, and by then it is too late to compensate. A simple scheme that compensates for delay in the previous sample is presented in (Lincoln, 2002). The compensator has the same basic structure as the well-known Smith predictor, but allows for a time-varying delay.

Many other heuristic jitter compensation schemes have been suggested, e.g., (Hägglund, 1992; Albertos and Crespo, 1999; Marti *et al.*, 2001). The approaches have in common that they require language and/or operating system support for instrumenting an application with measurement code.

In order to fully apply these techniques it is necessary to increase the understanding of how temporal nondeterminism affects control performance. This requires new theory and tools that are now beginning to emerge. An important issue that still is lacking is theory that allows us to determine which level of temporal determinism that a given control loop really requires in order to meet given control objectives on stability and performance. Is it necessary to use a time-triggered approach or will an event-based approach perform satisfactorily? How large are the input-output latencies that can be tolerated? Is it OK to now and then skip a sample in order to maintain the schedulability of the task set? Ideally one would like to have an index that decides the required level of temporal determinism through a single quantitative measure. One possible name for such an index would be the schedulability margin. This measure would need to combine both a margin with respect to input-output latency and jitter

and a margin that decides how large sampling jitter the loop can tolerate. For constant input-output latencies the classical phase margin can be applied.

An extension of the classical delay margin to time-varying delays is proposed in (Cervin *et al.*, 2004). The jitter margin $J_m(L)$ is defined as the largest input-output jitter for which closed-loop stability is guaranteed for any time-varying latency $\Delta \in [L, L + J_m(L)]$, where L is the constant part of the input-output latency. The jitter margin is based on the stability theorem defined in (Kao and Lincoln, 2004). The jitter margin can be used to derive hard deadlines that guarantee closed-loop stability, provided that the scheduling method employed can provide bounds on the worst-case and best-case response times of the controller tasks. What is still missing in order to be able to define a reasonable analytical concept for a schedulability margin is a simple sampling jitter criterion. The criterion should ideally tell how large variations around a nominal sampling interval that the process could tolerate and still be stable, alternatively maintain acceptable performance.

In addition to being temporally robust, it is also important for a control system to be robust towards faults. A large amount of theory and methods have been developed for fault detection, diagnosis, and fault-tolerance within the control community. However, the large majority of this work concerns faults that occur within the plant, sensors, or actuators. As most software engineers are sadly aware of, faults in the software system are far more common than in the plant under control. In spite of this, the amount of work that considers robustness against these types of faults, i.e., *functional robustness*, is very small. In (Gäfvert *et al.*, 2003) a method is presented that renders a control system more robust to computer-level faults leading to data errors. The method is based on the introduction of artificial signal limits in combination with an anti-windup scheme. Related to this, in (Askerdal *et al.*, 2003), a methodology is developed for analyzing the the impact that these types of data errors have on control system dependability.

5. CO-DESIGN TOOLS

In order for co-design of control, computing, and communication systems to become feasible, it is necessary to have software tools that allow the designers to analyze and simulate how timing affects control performance. Such tools have recently begun to emerge, both from the control and the computing communities. Most of the available tools are simulation tools. However, there are also tools that provide analysis of control performance subject to timing variations or integrate schedulability analysis with task allocation. Here, two such tools will be briefly described: Jitterbug [1] and TrueTime [2], together with an overview of other related tools.

5.1 *Jitterbug*

Jitterbug (Cervin *et al.*, 2003; Lincoln and Cervin, 2002) is a MATLAB-based toolbox that computes a quadratic performance criterion for a linear control system under various timing conditions. Using the toolbox, one can easily assert how sensitive a control system is to delay, jitter, lost samples, etc., without resorting to simulation. The tool is quite general and can also be used to investigate jitter-compensating controllers, aperiodic controllers, and multi-rate controllers. The main contribution of the toolbox, which is built on well-known theory (linear quadratic Gaussian (LQG) theory and jump linear systems), is to make it easy to apply this type of stochastic analysis to a wide range of problems.

Jitterbug offers a collection of MATLAB routines that allow the user to build and analyze simple timing models of computer-controlled systems. A control system is built by connecting a number of continuous- and discrete-time systems. For each subsystem, optional noise and cost specifications may be given. In the simplest case, the discrete-time systems are assumed to be updated in order during the control period. For each discrete system, a random delay (described by a discrete probability density function) can be specified that must elapse before the next system is updated. The total cost of the system (summed over all subsystems) is computed algebraically if the timing model system is periodic or iteratively if the timing model is aperiodic. A higher value of the cost function typically indicates that the closed-loop system is less stable (i.e., more oscillatory), and an infinite cost means that the control loop is unstable. The cost function can easily be evaluated for a large set of design parameters and can be used as a basis for the control and real-time design.

5.2 *TrueTime*

TrueTime (Cervin *et al.*, 2003; Henriksson *et al.*, 2002*a*) is a MATLAB/Simulink-based tool that facilitates simulation of the temporal behavior of a multitasking real-time kernel executing controller tasks. The tasks are controlling processes that are modeled as ordinary continuous-time Simulink blocks. TrueTime also makes it possible to simulate models of standard MAC layer network protocols, and their influence on networked control loops.

The main use of TrueTime is for simultaneous simulation of all aspects of distributed real-time control

[1] Available at http://www.control.lth.se/~lincoln/jitterbug.
[2] Available at http://www.control.lth.se/~dan/truetime.

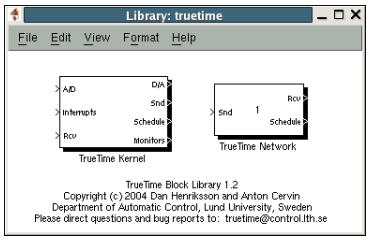

Fig. 3. The TrueTime block library.

applications. By co-simulation of continuous process dynamics, task execution in real-time kernels, and network communication, it is possible to evaluate the performance of control loops subject to the constraints of the target system.

In TrueTime, kernel and network Simulink blocks are introduced, the interfaces of which are shown in Figure 3. The kernel blocks are event-driven and execute code that models, e.g., I/O tasks, control algorithms, and network interfaces. The scheduling policy of the individual kernel blocks is arbitrary and decided by the user. Likewise, in the network, messages are sent and received according to the chosen network model.

The level of simulation detail is also chosen by the user—it is often neither necessary nor desirable to simulate code execution on instruction level or network transmissions on bit level. TrueTime allows the execution time of tasks and the transmission times of messages to be modeled as constant, random, or data-dependent. Furthermore, TrueTime allows simulation of context switching and task synchronization using events or monitors.

The kernel block is a MATLAB S-function that simulates a computer with a simple but flexible real-time kernel, A/D and D/A converters, a network interface, and external interrupt channels. The kernel executes user-defined tasks and interrupt handlers. An arbitrary number of periodic and aperiodic tasks can be created to run in the TrueTime kernel. Tasks may also be created dynamically as the simulation progresses.

Each task is characterized by a number of static (e.g., relative deadline, period, and priority) and dynamic (e.g., absolute deadline and release time) attributes. It is possible to attach two overrun handlers to each task: a deadline overrun handler (triggered if the task misses its deadline) and an execution time overrun handler (triggered if the task executes longer than its worst-case execution time).

Interrupts may be generated in two ways: externally (associated with the external interrupt channel of the kernel block) or internally (triggered by user-defined timers). When an external or internal interrupt occurs, a user-defined interrupt handler is scheduled to serve the interrupt.

The execution of tasks and interrupt handlers is defined by user-written code functions. These functions can be written either in C++ (for speed) or as MATLAB m-files (for ease of use). Control algorithms may also be defined graphically using ordinary discrete Simulink block diagrams. The execution may be preemptive or non-preemptive; this can be specified individually for each task and interrupt handler. At the interrupt level, interrupt handlers are scheduled according to fixed priorities. At the task level, dynamic-priority scheduling may be used. At each scheduling point, the priority of a task is given by a user-defined priority function, which is a function of the task attributes. This makes it easy to simulate different scheduling policies.

The network block is event-driven and executes when messages enter or leave the network. A message contains information about the sending and the receiving computer node, arbitrary user data (typically measurement signals or control signals), the length of the message, and optional real-time attributes such as a priority or a deadline.

The network block simulates medium access and packet transmission in a local area network. Six simple models of networks are currently supported: CSMA/CD (e.g. Ethernet), CSMA/AMP (e.g. CAN), Round Robin (e.g. Token Bus), FDMA, TDMA (e.g. TTP), and Switched Ethernet. The propagation delay is ignored, since it is typically very small in a local area network. Only packet-level simulation is supported, i.e., it is assumed that higher protocol levels in the kernel nodes have divided long messages into packets.

5.3 *Other Tools*

While numerous other tools exist that support either simulation of control systems (e.g., MATLAB Simulink) or scheduling algorithms (e.g., DRTSS (Storch and Liu, 1996) and RTSIM (Casile *et al.*, 1998)), very few tools have been developed that support co-simulation of control systems and real-time scheduling. The RTSIM simulator has been extended with a numerical module (based on the Octave library) that supports simulation of continuous dynamics (Palopoli *et al.*, 2000).

Ptolemy II is the third generation of software produced within the Ptolemy project (Hylands *et al.*, 2003). Ptolemy II supports *heterogeneous*, *hierarchical* modeling, simulation, and design of concurrent systems, especially embedded systems. The focus is on complex systems mixing various technologies and operations. Simulation models are constructed under *models of computation* that govern the interaction of the components in the model. Different models of computation are used for modeling different types and parts of systems. The recently developed *timed multitasking* (TM) domain (Liu and Lee, 2003) adds the possibility to model fixed-priority scheduling of tasks with fixed execution times.

Orccad (Simon and Girault, 2001) is a CAD system and approach aimed at the development of robotic systems from high-level specifications down to the implementation details. It deals with hybrid systems where continuous-time aspects relating to control laws, must be merged with discrete-time aspects. The Syndex tool supports rapid prototyping of reactive data-driven algorithms implemented on distributed heterogeneous hardware architectures (Pernet and Sorel, 2003). Syndex lets the user specify both the algorithm and the distributed hardware in a graphical environment, and then automates the mapping and scheduling of functions and communications on the processors and communication buses.

The Aida toolset (Redell *et al.*, 2004) is an environment for model-based codesign and analysis of real-time control systems that also supports timing analysis. The toolset consists of a modelling environment, *Aidasign*, which interfaces with MATLAB/Simulink, and a response time analysis tool, *Aidalyze*. The real-time system design starts with the translation of the Simulink model to a *data-flow diagram* (DFD) in Aidasign. The timing aspects of the controller, such as sampling periods and delays then constitute requirements on the real-time system design. Another fundamental model in Aida is the *hardware structure diagram* (HSD), where the hardware architecture, in terms of processors and their interconnections via communication links, is designed. In the HSD the functions and data flows in the associated data-flow diagram(s) are mapped to processors and communication links, respectively. Based on this, a real-time implementation is designed. XILO (El-khoury and Törngren, 2001) is a toolset that is related to AIDA and intended for similar usage. XILO also contains support for fault-injection in blocks and signals.

For networked control loops the communication aspects also must be included in the codesign. There exists a large number of network simulators today, but very few of them include any co-design aspects. One of the most well-known network simulators is NS-2 (*NS-2*, 2004), which is a discrete-event simulator for both wired and wireless networks. It also supports simple movement models for mobile applications. Another discrete-event computer network simulator is OMNeT++ (*OMNeT++*, 2004). There are also some network simulators geared towards the sensor network domain. TOSSIM (Levis *et al.*, 2003) compiles directly from TinyOS code and scales very well. Network in a box (NAB) (*NAB (Network in A Box)*, 2004) is another simulator for large-scale sensor networks.

Recently, support for wireless networks has been added to TrueTime making it possible to simulate control application using sensor/actuator networks or wireless mobile ad hoc networks. Another tool for joint network/control simulation is presented in (Branicky *et al.*, 2003). Ptolemy II has recently also been extended to support wireless sensor net applications, (Baldwin *et al.*, 2004)

6. CONTROL OF REAL-TIME COMPUTING SYSTEMS

Feedback control is a well-established and mathematically well-founded theory that is well suited for handling uncertainties. Traditionally, the uncertainties are associated with the physical plant that should be controlled. However, the theory and design principles can also be applied to arbitrary systems containing uncertainties, e.g., real-time computing and communication systems with uncertainties in, e.g. workload and resource utilization patterns. When feedback is used in combination with real-time resource scheduling, precise schedulability models are not needed. Instead, the systems adjusts its resource allocation dynamically to achieve the desired temporal behavior. Many aspects of the real-time performance of a computing system can be inferred from the behavior of resource queues (for example the ready queue, semaphore queues and communication queues). On a high-level, the queue can be modeled as an integrator of request flows. The actuators acting on a request queue can be divided into enqueue and dequeue actuators. The former type of actuator adjust the input flow to the queue. An admission controller is an example of this. The latter type of actuators adjust the output flow from the queue, by, e.g., changing the quality-of-service level of the currently served requests.

Control of real-time computing systems is an area that currently is receiving a lot of attention, in particular from the real-time community. A recent survey of the area with many further references can be found in (Sha *et al.*, 2004). Recently also textbooks have emerged (Hellerstein *et al.*, 2004). Control can in principle be applied to any resource allocation problem in real-time computing and communication. However, the majority of the work so far can be found in three areas: control of web-servers, control of the CPU utilization of periodic task sets, and the use of feedback control in network communication. Here we will focus on the second application area.

Here we consider the case where the computing system implements multiple digital control loops, with each controller being realized as a separate periodic task. The main resource of concern in these types of problems is the CPU time. The objective for the feedback scheduler is to dynamically adjust the CPU utilization of the controller tasks so that the task set remains schedulable and the stability and performance requirements of the individual controllers are met. The feedback scheduling problem can therefore be stated as an optimization problem where the objective is to maximize the global control performance of the physical system according to some criterion, subject to resource and schedulability constraints.

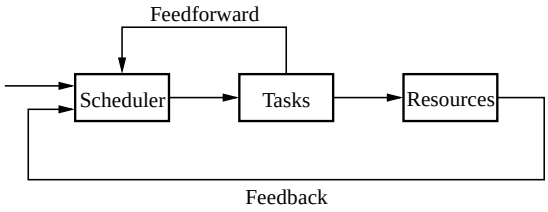

Fig. 4. A general feedback scheduling structure.

One reason why feedback is needed for the scheduler is the uncertainty associated with the WCET estimation. In some control applications, the computational workload can change dramatically over time as different control algorithms are switched in and out. Others use anytime algorithms, which may execute a varying number of iterations.

Different structures are possible in feedback scheduling (Cervin *et al.*, 2002). A pure feedback scheme is reactive in the sense that the feedback scheduler will only remove a utilization error once it is already present. By combining the feedback with feedforward a pro-active scheme is obtained. The feedforward path could be use to allow controller task to inform the scheduler that they are changing their desired amount of resources, e.g., changing nominal sampling periods, and to give the scheduler the possibility to compensate for this before any overload has occurred. The feedforward path can be also be used for dynamic task admission. A block diagram of the feedback-feedforward structure is shown in Fig. 4.

It is also possible to consider a layered or cascaded control structure. The outer layer would consist of a feedback scheduler that, based on a desired set-point for the overall utilization, generates as outputs the desired utilization for each controller task. Associated with each controller task is then a local feedback scheduler that is responsible for adjusting the timing parameters of the task in order to fulfill the desired utilization. The utilization assigned to each controller task can be viewed as its share of the total resource, e.g., the total CPU capacity. This approach can be combined with reservation-based scheduling in order to provide temporal protection for the individual tasks (Mercer *et al.*, 1993). Each task can then be seen as if it is executing on its own virtual CPU. One example of a reservation-based scheduling scheme is the constant-bandwidth server (CBS) (Abeni and Buttazzo, 1998). Analysis of a reservation-based feedback scheduler is presented in (Abeni *et al.*, 2002).

The actuators in the feedback scheduler loop are means with which the scheduler can modify the CPU utilization of individual controllers. For a controller task the task period is a natural actuation knob. Changing the task period dynamically may be more or less difficult depending on how the controller is implemented. For a controller implemented on input-output form it is generally more difficult to change the sampling period than for a controller that is realized on state-space form. In certain cases it may be necessary to use a Kalman filter to estimate the values of the state at the new sampling instants.

An alternative actuation knob is the execution time demands of the controller. This can be achieved using a multiple-versions approach or using an anytime approach. One example where the latter approach is applicable is model-predictive controllers (MPC) in which an quadratic optimization problem is solved iteratively in every sample, see (Henriksson *et al.*, 2002*b*).

The sensor in this type of feedback scheduler is a measurement of the actual CPU utilization. This assumes that the processor and RTOS are equipped with the means to perform such measurements. In order to avoid control actions caused by spurious measurement outliers (noise) a low-pass filter may be included in the sensor. The low-pass filter can also be used to calculate an average of the utilization over a certain time period (e.g., the sampling period of the feedback scheduler). Such filters are an important source of loop dynamics.

Stating the feedback scheduling problem as an optimization problem, a suitable optimization metric must be used. For off-line optimization, the sum of a set of quadratic cost functions has been suggested (Seto *et al.*, 1996). The performance of each control loop is described by a function $J_i(h) = \int (x^T Q_1 x + u^T Q_2 u) dt$ where x is the plant state and u is the control signal. Given a set of n controllers with execution times C_i, the optimization problem is stated as finding $\min_{h_1,\ldots h_n} \sum J_i(h)$ subject to schedulability constraints. Using a tool such as Jitterbug, it is possible to evaluate such cost functions analytically.

The approach above does not consider input-output latency in the control loops. An approach to joint optimization of sampling period and input-output latency subject to performance specifications and schedulability constraints is presented in (Ryu *et al.*, 1997). The control performance is specified in terms of steady state error, overshoot, rise time, and settling time. These performance parameters are expressed as functions of the sampling period and the input-output latency. The use of robustness measures in off-line optimization has been suggested (Palopoli *et al.*, 2002).

The optimization problem can also be solved online via feedback to the scheduler. For this purpose, a model of the scheduler is needed from a control perspective. The dynamics involved in feedback scheduling are often of low order or even purely static. The reason for this is obvious. If a task is given more or less CPU time the total utilization will change as soon as the next job of the task is started. Often the dynamics in the feedback loop comes from the filtering in the sensor. A consequence of this is that it is often enough with very simple control strategies in the feedback scheduler.

In (Eker *et al.*, 2000; Cervin *et al.*, 2002) it was shown that a simple linear proportional rescaling of the nominal task periods in order to meet the utilization setpoint is optimal with respect to the overall control performance under certain assumptions. It holds if the control cost functions are quadratic, i.e., $J_i(h_i) = \alpha_i + \beta_i h_i^2$, or if they are linear, $J_i(h_i) = \alpha_i + \gamma_i h_i$, and if the objective of the feedback scheduler is to minimize the sum of the control cost functions or a weighted sum of the control cost functions. The result is a simple and fast calculation that can be applied on-line.

7. RESEARCH DIRECTIONS

Implementation-aware control design, control and computation codesign, and feedback scheduling are challenging research areas where a lot or progress have been made in recent years. However, a number of research issues are still open and candidates for future research. In the following some of these are discussed.

Temporal Robustness of Control Loops. The question of how much temporal nondeterminism a given control loop can handle and still meet stability and performance requirements is still not completely solved. If the distributions of the sampling jitter and latency jitter are known, then it is possible to calculate an optimal controller that minimizes a given quadratic performance criterion. This problem was solved long time ago, see, e.g. (Davidson, 1973). However, what is more interesting is the robustness of a controller that only has partial information about the distributions and/or has been designed assuming perfect sampling and constant latency.

Event-driven Control. A consequence of the soft control-based approach to control design is an increased emphasis on aperiodically sampled control systems. Analysis of systems with event based sampling is related to general work on discontinuous systems, (Utkin, 1987), and to work on impulse control, see (Bensoussan and J.-L., 1984). Much work on systems of this type was done in the period 1960–1980. Analysis of event-based sampled systems is considerably harder than for time-based sampled systems. This is due to the fact that sampling is no longer a linear operation. There are several papers that treat special system setups, such as observers for linear system with quantized outputs, e.g., (Delchamps, 1989), many of which use classical ideas from Kalman observer design. In (Åström and Bernhardsson, 1999) it is shown that event-based sampling can be more efficient than equidistant sampling. However, we are still very far from a general theory for aperiodic event-triggered sampled systems.

Event-driven Control of Computing Systems. Computing systems are generally event-driven. Although they can be approximated by continuous-time models and it is possible to apply time-driven control, it is plausible that more can be gained with event-driven control. Here the control action is generated each time a request enters or leaves a request queue, or when a controller tasks exceeds its deadline. An example of work in this direction is (Henriksson *et al.*, 2004).

Dynamic Models of Computing Systems. Computers are engineering artifacts whose behavior generally does not obey any first principles. Hence, the models used in control design are often generated from input-output measurements. Research is needed on what types of models that are most appropriate for these types of systems.

Feedback Scheduling Control Structures and Architectures. Feedback scheduling systems can be structured in several ways, both with respect to which controller structures that should be used and how the controller structures should be combined. An interesting approach is to view the feedback scheduler as a resource broker (e.g. CPU time broker and communication bandwidth broker) that uses quality of service based approaches (quality of control) to negotiate with the resource users, e.g., controllers. This approach is discussed in (Sanfridson, 2004).

Constrained Resource Control. Resource control problems often contain a combination of local constraints of minimum type and global constraints of maximum type. Each resource user or client must at least be given a certain amount of base resources while at the same time the global amount of resources that may be handed out is limited. This situation gives rise to interesting control problems especially for distributed systems with communication delays and limited communication channels. An example of this is distributed power control in third generation cellular mobile phone systems.

8. CONCLUSION

Control and embedded real-time systems have several connections. Control systems are an important application area for embedded systems with special requirements and possibilities. Control is also a fundamental technology which can be used in the design of embedded real-time computing systems. Implementation-aware control is especially important for embedded applications with limited computing resources.

This survey has been performed as a part of the EU/IST FP6 NoE ARTIST2.

9. REFERENCES

Abeni, Luca and Giorgio Buttazzo (1998). Integrating multimedia applications in hard real-time systems. In: *Proc. 19th IEEE Real-Time Systems Symposium*. Madrid, Spain.

Abeni, Luca, Luigi Palopoli, Giuseppe Lipari and Jonathan Walpole (2002). Analysis of a reservation-based feedback scheduler. In: *Proc. 23rd IEEE Real-Time Systems Symposium*.

Albertos, Pedro and Alfons Crespo (1999). Real-time control of non-uniformly sampled systems. *Control Engineering Practice* **7**, 445–458.

Askerdal, Örjan, Magnus Gäfvert, Martin Hiller and Neeraj Suri (2003). Analyzing the impact of data errors in safety-critical control systems. *IEICE Transactions on Information and Systems*. Special Issue on Dependable Computing.

Åström, K.J. and B. Bernhardsson (1999). Comparison of periodic and event based sampling for first-order stochastic systems. In: *Proceedings of the 14th IFAC World Congress, Beijing, P.R. China*.

Åström, K.J. and B. Wittenmark (1997). *Computer-Controlled Systems*. Prentice Hall.

Baldwin, Philip, Sanjeev Kohli, Edward A. Lee, Xiaojun Liu and Yang Zhao (2004). Modeling of sensor nets in ptolemy II. In: *IPSN'04: Proceedings of the third international symposium on Information processing in sensor networks*. ACM Press. pp. 359–368.

Bensoussan, A. and Lions J.-L. (1984). *Impulse control and quasi-variational inequalities*. Gauthier-Villars, Paris.

Benveniste, A. and G. Berry (1991). The synchronous approach to real-time programming. *Proceedings of the IEEE* **79**, 1270–1282.

Branicky, Michael, Vincenzo Liberatore and Stephen M. Phillips (2003). Networked control system co-simulation for co-design. In: *Proc. American Control Conference*.

Casile, A., G. Buttazzo, G. Lamastra and G. Lipari (1998). Simulation and tracing of hybrid task sets on distributed systems. In: *Proc. 5th International Conference on Real-Time Computing Systems and Applications*.

Cervin, A., B. Lincoln, J. Eker, K.-E. Årzén and G. Buttazzo (2004). The jitter margin and its application in the design of real-time control systems. In: *Proc. 10th International Conference on Real-Time and Embedded Computing Systems and Applications*. Göteborg, Sweden.

Cervin, Anton (1999). Improved scheduling of control tasks. In: *Proceedings of the 11th Euromicro Conference on Real-Time Systems*. York, UK. pp. 4–10.

Cervin, Anton and Johan Eker (2003). The Control Server: A computational model for real-time control tasks. In: *Proceedings of the 15th Euromicro Conference on Real-Time Systems*. Porto, Portugal. pp. 113–120. Best paper award.

Cervin, Anton, Dan Henriksson, Bo Lincoln, Johan Eker and Karl-Erik Årzén (2003). How does control timing affect performance?. *IEEE Control Systems Magazine* **23**(3), 16–30.

Cervin, Anton, Johan Eker, Bo Bernhardsson and Karl-Erik Årzén (2002). Feedback-feedforward scheduling of control tasks. *Real-Time Systems* **23**(1–2), 25–53.

Davidson, Charles (1973). Random sampling and random delays in optimal control systems. PhD thesis. Department of Optimization and Systems Theory, Royal Institute of Technology (KTH), Sweden.

Delchamps, D. (1989). Extracting state information from a quantized output record. *Systems and Control Letters* **13**, 365–372.

Dvorak, D., G. Bollella, T. Canham, V. Carson, C. Champlin, B. Giovannoni, M. Indictor, K. Meyer, A. Murray and K. Reinholtz (2004). Project golden gate: Towards real-time java in space missions. In: *Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'04)*.

Eker, Johan, Per Hagander and Karl-Erik Årzén (2000). A feedback scheduler for real-time control tasks. *Control Engineering Practice* **8**(12), 1369–1378.

El-khoury, Jad and M. Törngren (2001). Towards a toolset for architectural design of distributed real-time control systems. In: *Proceedings of the 22nd IEEE Real-Time Systems Symposium*. London, England.

Gäfvert, Magnus, Björn Wittenmark and Örjan Askerdal (2003). On the effect of transient data-errors in controller implementations. In: *Proceedings of the American Control Conference*. Denver, Colorado. pp. 3411–3416.

Hägglund, Tore (1992). A predictive PI controller for processes with long dead times. *IEEE Control Systems Magazine* **12**(1), 57–60.

Hanselmann, H. (1987). Implementation of digital controllers—a survey. *Automatica* **23**(1), 7–32.

Hellerstein, Joseph L., Yixin Diao, Sujay Parekh and Dawn M. Tilbury (2004). *Feedback Control of Computing Systems*. Wiley-IEEE Press.

Henriksson, Dan, Anton Cervin and Karl-Erik Årzén (2002*a*). TrueTime: Simulation of control loops under shared computer resources. In: *Proceedings of the 15th IFAC World Congress on Automatic Control*. Barcelona, Spain.

Henriksson, Dan, Anton Cervin, Johan Åkesson and Karl-Erik Årzén (2002*b*). On dynamic real-time scheduling of model predictive controllers. In: *Proceedings of the 41st IEEE Conference on Decision and Control*. Las Vegas, NV.

Henriksson, Dan, Ying Lu and Tarek Abdelzaher (2004). Improved prediction for web server delay control. In: *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS 04)*. Catania, Sicily, Italy.

Henzinger, T., B. Horowitz and C. Kirsch (2003). Giotto: a time-triggered language for embedded programming. *Proceedings of the IEEE* **91**(1), 84–99.

Hylands, C., E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao and H. Zheng (2003). Overview of the Ptolemy project. Technical Report UCB/ERL M03/25. Department of Electrical Engineering and Computer Science, University of California Berkeley, CA.

Kao, Chung-Yao and Bo Lincoln (2004). Simple stability criteria for systems with time-varying delays. *Automatica*. To be published.

Klein, M. H., T. Ralya, B. Pollak, R. Obenza and M. Gonzalez Härbour (1993). *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publisher.

Kopetz, H. and G. Bauer (2003). The time-triggered architecture. *Proceedings of the IEEE* **91**(1), 112–126.

Levis, Philip, Nelson Lee, Matt Welsh and David Culler (2003). TOSSIM: accurate and scalable simulation of entire TinyOS applications. In: *Proceedings of the 1st international conference on Embedded networked sensor systems*. Los Angeles, CA, USA. pp. 126–137.

Lincoln, Bo (2002). Jitter compensation in digital control systems. In: *Proceedings of the 2002 American Control Conference*.

Lincoln, Bo and Anton Cervin (2002). Jitterbug: A tool for analysis of real-time control performance. In: *Proceedings of the 41st IEEE Conference on Decision and Control*. Las Vegas, NV.

Liu, C. L. and J. W. Layland (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM* **20**(1), 40–61.

Liu, Jie and Edward Lee (2003). Timed multitasking for real-time embedded software. *IEEE Control Systems Magazine*.

Marti, Pau, Gerhard Fohler, Krithi Ramamritham and Josep M. Fuertes (2001). Jitter compensation for real-time control systems. In: *Proceedings of the 22nd IEEE Real-Time Systems Symposium*.

Mercer, Clifford W., Stefan Savage and Hideyuki Tokuda (1993). Processor capacity reserves for multimedia operating systems. In: *Proc. Fourth Workshop on Workstation Operating Systems*.

NAB (Network in A Box) (2004). Home page: http://nab.epfl.ch/.

Nilsson, Johan, Bo Bernhardsson and Björn Wittenmark (1998). Stochastic analysis and control of real-time systems with random time delays. *Automatica* **34**(1), 57–64.

NS-2 (2004). Home page: http://www.isi.edu/nsnam/ns/index.html.

OMNeT++ (2004). Home page: http://www.omnetpp.org.

Palopoli, L., L. Abeni and G. Buttazzo (2000). Real-time control system analysis: An integrated approach. In: *Proc. 21st IEEE Real-Time Systems Symposium*.

Palopoli, Luigi, Claudio Pinello, Alberto Sangiovanni-Vincentelli, Laurent El-Ghaoui and Antonio Bicchi (2002). Synthesis of robust control systems under resource constraints. In: *Proc. Workshop on Hybrid Systems: Computation and Control*.

Pernet, N. and Y. Sorel (2003). Optimized implementation of distributed real-time embedded systems mixing control and data processing. In: *Proceedings of the ISCA 16th International Conference: Computer Applications in Industry and Engineering (CAINE-2003)*. Las Vegas, USA.

Redell, O., J. El-Khoury and M. Törngren (2004). The AIDA tool-set for design and implementation analysis of distributed real-time control systems. *Journal of Microprocessors and Microsystems* **28**(4), 163–182.

Ryu, M., S. Hong and M. Saksena (1997). Streamlining real-time controller design: From performance specifications to end-to-end timing constraints. In: *Proc. 3rd IEEE Real-Time Technology and Applications Symposium*. pp. 91–99.

Sanfridson, Martin (2004). Quality of control and real-time scheduling. PhD thesis. Department of Machine Design, Royal Institute of Technology (KTH), Sweden.

Seto, D., J. P. Lehoczky, L. Sha and K. G. Shin (1996). On task schedulability in real-time control systems. In: *Proc. 17th IEEE Real-Time Systems Symposium*. Washington, DC. pp. 13–21.

Sha, L., T. Abdelzaher, K.-E. Årzén, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, A. Cervin, J. Lehoczky and A. K. Mok (2004). Real time scheduling theory: A historical perspective. *Journal of Real-time Systems*. Invited paper to appear.

Simon, D. and A. Girault (2001). Synchronous programming of automatic control applications using Orccad and Esterel. In: *Proceedings of the 40th IEEE Conference on Decision and Control, CDC'01*. Orlando, USA.

Stankovic, J., M. Spuri, K. Ramamritham and G. Buttazzo (1998). *Deadline Scheduling for Real-Time Systems—EDF and Related Algorithms*. Kluwer Academic Publishers.

Storch, M. F. and J. W.-S. Liu (1996). DRTSS: A simulation framework for complex real-time systems. In: *Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium*. pp. 160–169.

Törngren, Martin (1998). Fundamentals of implementing real-time control applications in distributed computer systems. *Real-Time Systems* **14**(3), 219–250.

Utkin, V. I. (1987). Discontinuous control systems: State of the art in theory and applications. In: *Preprints 10th IFAC World Congress*. Munich, Germany.

Wittenmark, Björn, Johan Nilsson and Martin Törngren (1995). Timing problems in real-time control systems. In: *Proceedings of the 1995 American Control Conference*. Seattle, Washington.