Roadmap on Real-Time Techniques in Control System Implementation

Control for Embedded Systems Cluster

EU/IST FP6 Artist2 NoE

Contents

Chapter 1

Introduction

An important result of the EU ARTIST FP5 project was four roadmaps on Hard Real-Time Development Environments, Component-Based Design and Implementation Platforms, Adaptive Real-Time Systems for Quality of Service Management, and Execution Platforms respectively, [Bouyssounouse and Sifakis, 2005]. The current roadmap written by the partners of the Control for Embedded Systems cluster within the EU/IST FP6 Network of Excellence ARTIST2 is focused on real-time implementation techniques for embedded and networked control systems. A special emphasis is put on the interactions between the control, computing, and communication subsystems, the influences these interactions have on control performance, as well as different ways to handle the interactions. In addition this roadmap describes the state of the art concerning techniques for implementing real-time control systems, analyzes the challenges posed by the new applications, and proposes a number of key developments to face these challenges.

Similar to the ARTIST roadmaps this roadmap is intended as a roadmap for *research* rather than an roadmap on industrial R & D in general. The roadmap is **not** a roadmap on control of computing systems. For this subject a separate complementary roadmap has been developed.

1.1 Background

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 generate 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.

Embedded control systems can be defined as *embedded systems* where control activities are relevant for their time constraints. Often, they also have resource constraints in terms of computing and communication resources. Embedded 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.

Traditionally, control algorithms are designed assuming unlimited computing and communication resources. In embedded systems, these resources are often shared between many applications and the environmental conditions are changing. Thus, the development of the real-time control algorithms should consider these constraints from the very beginning in the control design procedure.

In industry, most embedded control systems are based on micro-controllers and PLC's (Programmable Logic Controllers). Micro-controllers provide most of the basic features to implement basic control systems (processor, input/output, converters) but on a relatively low level. They often have no operating system, or a specialized embedded kernel. Small kernels with specific services are used for these kind of systems to develop several tasks. Their capacity for adaptation and reconfiguration is very limited and their use is constrained to well very known and simple control systems (printers, modems ...).

The need for higher computation power can be improved by the use of digital signal processors (DSPs). These microcomputers, whose hardware, software, and instruction sets are optimized for high-speed numeric processing applications, play an essential role for processing digital data in real time.

However, the general trend is that control systems are becoming increasingly complex from both the control and computer science perspectives. Today, even seemingly simple embedded control systems often contain a multitasking real-time kernel and support networking. The complexity and functionality level of the software is increasing steadily and it is becoming normal that it is the embedded software that dominates the development cost and schedule. The traditional way of developing software for each embedded project from scratch is giving way to the need to reuse software, and build on existing software wherever possible, using, e.g., component technology and object-orientation.

Many computer-controlled systems are also distributed, consisting of computer nodes and a communication network connecting the various systems. It is not uncommon 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. Examples of these embedded operating systems are: embedded Linux (several distributions), Windows CE, VxWorks, QNX, OS-9, etc. The diversity of operating environments and platforms poses a real challenge in deploying software across multiple platforms and configurations. The use of embedded operating systems providing POSIX interface facilitates the portability and reusability of the applications. The multitasking and the networking imply that the CPU time and the communication bandwidth can be viewed as shared resources for which the tasks compete.

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 of 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. This has led to a lack of mutual understanding between the fields. 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 arguable whether it is necessary to model them as hard-deadline tasks or not.

From an industrial point of view it can in many cases also be expensive or difficult to pursue a separation-based design approach. Guaranteeing hard deadlines and providing tight bounds on input output latency is costly. It may require the use of computational models which do not match the current state of practice. It requires good worst-case execution time estimates. It often implies that the resource utilization is quite low. Hence, in many industrial application, although the intention is to separate the concerns between control and computing, a complete separation will not be achieved. The effect

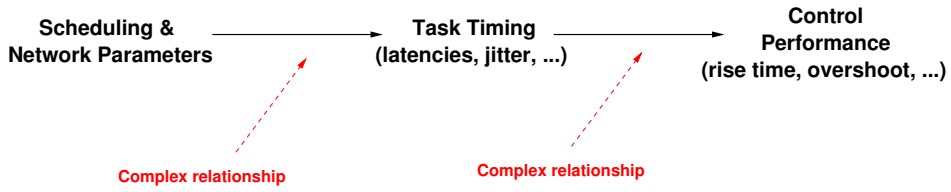

Figure 1.1: The relationship between computer system design parameters and control performance.

of this is undesired interactions between the computing system and control system, e.g., jitter and delays, having a negative effect on control performance.

The relationship between computer system design parameters and control performance is quite complex. The situation is shown in Fig. 1.1. Scheduling and networking related parameters such as thread periods, deadlines, priorities, protocols, etc., influence the controller task parameters (latencies, jitter, etc) in a complex way. Similarly the controller task parameters influence the control performance parameters (e.g., rise time, overshoot, signal variances, etc) in an equally complex way. Hence, also in applications where a separation of concerns-based design approach is followed, the need is large for analysis tools that help the designer to quantify the relationships above.

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.

Of special interest is temporal robustness in control systems, i.e., robustness towards implementation-level timing uncertainties and implementation-level functional robustness, i.e., tolerance towards implementation platform faults. Increased understanding of which types of temporal guarantees that really are required by a given control application in order to meet desired specifications is needed. Different computational models are more or less well suited for control system implementation. Software component technology and domain-specific languages for control systems are important ingredients in control systems implementation as well as model-based development tools.

The main motivation for this roadmap is to advance the state of the art in applying real-time system methodologies for control system implementations, in particular, embedded control system. There is a research need in this field, as it has started recently and there is not yet a formal body of theory behind it. But, on the other hand, there is a

great motivation driven by more demanding applications involving powerful embedded control systems.

1.2 Organization of this document

After this introductory section, the roadmap is organized as follows:

As real-time techniques in control system application is important in a large number of application fields, such as avionics, automotive, mobile robots and so on, the Section 2 is devoted to a review of the needs and current practices in selected industrial sectors. In order to frame the problems and challenges, Section 3 is devoted to analyze the main parameters defining the interplay between control systems and their real-time implementation. Different integrated real-time and control design techniques are reviewed in Section 4. Most embedded control systems operate within a network. Thus, the main issues related to implementing real-time control systems using typical wired and wireless networks are summarized in Section 5. Finally in Section 6, challenges and future research directions are outlined.

Chapter 2

Needs and Practices in Selected Industrial Sectors

Embedded control can be found in a large number of different industry branches. In this version of the roadmap the focus has been the automotive sector and the industrial automation and robotics sector although also the situation in the avionics sector and the medical section are briefly described.

2.1 Automotive industry

Embedded control has played a major role in automotive systems for a substantial time already and the role is expected to increase even further in the future as the requirements on and demands for, e.g., improved energy management and active collision avoidance grow. One sign of this is the fact that computers commonly go under the name electronic control units (ECUs) rather than CPUs. A top-level modern car contains some 50-80 ECUs of varying complexity. A majority of these implement different feedback control functions, for instance, engine control, traction control, anti-lock braking, active stability control, cruise control, and climate control.

The technology drivers includes the following topics, [EAST-EEA, 2002],: reductions of development cost, time to market, product cost, and maintenance efforts; improvement of reliability, safety for the persons in the car, outside the car, and for the system itself, improvement in vehicle security, emission and noise reductions, increased vehicle efficiency, and improvements in comfort, customer satisfaction, and mobility efficiency.

The needs of the automotive systems with respect to embedded control can be divided into two categories. The first category include items that are of a more general software nature and not so much specific to control. This includes the need for re-use in various ways, e.g., using component technologies and support for product variability, including,

e.g., product line approaches, improved requirements management and better early prediction models for software costs, quality and development time. The second category includes the things that are more directly related to the actual control. This will be the focus of this presentation.

A common phenomena in several industrial sectors and also in the automotive industry is a lack of integration and mutual understanding between control engineers and software engineers. Control engineers consider the algorithm design as the challenging part and the need of the algorithmic part of the software should steer the development. The software engineers considers the architecture design as the challenge and underestimate the complexity of the algorithms. The control engineering departments, e.g., engine control departments, rely heavily on automatic code generation tools, e.g., based on Simulink. In that way they partly can avoid having to consider the software engineering considerations. For example, in certain companies and for certain functions already today 100% of the algorithmic controller code is automatically generated. This code is then forwarded to the software engineers for integration and implementation. During this process, unfortunately, a lot of information can be lost, e.g., assumptions which are not made explicit in the code. The situation is made worse by the fact that often the integration of the code is performed by an external system supplier. In order to overcome this problem better integration is needed between the control design and the software design. The model-based control design where plant models are used needs to be integrated with UML-type software models. Control-motivated requirements must be made explicit and transfered properly to the software engineers

There is currently a change within the automotive industry moving away from the so called federated design philosophy over to a more integrated design philosophy. In a federated design each function is implemented in a dedicated and discrete hardware device. The hardware devices are then interconnected using buses to in order to achieve the necessary coordination. The functions or subsystems typically comes special suppliers and can, from the car manufacturer's point of view, be viewed as black boxes. This is one reason that the number of ECUs has grown so much during recent years. For safety and fault-tolerance reasons each subsystem may also use replicated ECUs. In a federated system the number of ECUs is smaller, but the individual ECUs are more powerful. Each ECU hosts a number of functions, which simplifies the coordination of the functions. However, it also creates new challenges with respect to fault isolation and fault containment. The different functions within the same ECU needs to be protected both from a functional point of view, e.g., through memory protection schemes, and temporally. The role of the system suppliers also change, becoming more focused on providing software components that can be hosted in the car manufacturer's ECUs.

Several architecture standardization activities are currently being pursued, or have recently ended, within the automotive industry. One example is AUTOSAR (Automotive Open Systems Architecture) where the aim is to specify and standardize the central architectural elements across different functional domains (e.g., power-train, chassis, safety, man-machine interface, and telematics). Another example is the EU/IST FP6 integrated projects DECOS (Dependable Embedded Components and Systems) where the aim is to define a system architecture that combines the complexity management

advantages of the federated approach, but also realizes the functional integration and hardware benefits of an integrated system. Strong topics within DECOS is fault isolation, fault containment, and diagnosis. The overall DECOS architecture is described in [Kopetz *et al.*, 2004]. The basis for architecture is time-triggered computations and communication. Other examples of large projects are EAST-EEA project, where a large number of car manufacturers collaborate and the CARTRONICS project initiated by Bosch.

The timing constraints of the control loops within the automotive industry have so far not really been of a hard nature. Hence, event-driven computing and communication have been the common solutions. For example, CAN has been, and is still the dominating network protocol. Although schedulability theory for event-based systems exists which also cover CAN-bus communication, this is rarely used. Time-driven control loops are most common, except from the engine control where event-based sampling is used (sampling relative to cam shaft revolutions). However, new more safety-critical applications such as steer-by-wire and brake-by-wire have increased the interest for statically-scheduled time-triggered approaches. Protocols such as TTP and Flexray are based on TDMA and allow inclusion of event-based communication in different ways.

The need to reduce cost in vehicle systems has led to a desire to use as inexpensive hardware components as possible. The results has been that very simple micro-controllers without any hardware support for floating point calculations has been the standard. Due to this fixed point numerics has been used when implementing controllers. Today this is beginning to change. In larger ECUs used for, e.g., engine control functions, hardware floating-point units (FPUs) are beginning to be available. In some cases they are also used for the control algorithms. However, industrial engine control is based on the extensive used of large engine maps, i.e., large lookup tables containing nonlinear functions. Also, if the processor supports floating point, these engine maps are still often represented in fixed point, in order to reduce the memory requirements.

2.2 Avionics and aerospace

The avionics sector has many similarities with the automotive sector from an embedded control point of view. The embedded systems are currently undergoing a change from a federated structure to an integrated structure. The operational complexity of the systems is growing exponentially. The reason being requirements on new capabilities, e.g., autonomous UAV aircrafts and new capabilities, e.g., real-time mission re-planning. The systems also typically in highly dynamic, and sometimes unknown operating environments. Model-driven engineering and support for re-use though, e.g., software components are very important for the future. System integration is still considered as a major problem. The safety-critical nature of the systems makes verification and certification major issues.

By tradition embedded flight control systems are statically scheduled. This is still mostly the case. European manufacturers, e.g., Airbus, have started using synchronous ap-

proaches, e.g., SCADE, to some degree. However, at the same time there is also a strong need for support for more dynamic behaviors with a mixture of hard and soft real-time tasks, active resource management, mode changes, and dynamic scheduling. This does, however, mainly concern the levels above the direct flight control.

2.3 Industrial automation & robotics

The industrial automation arena is dominated by large companies such as ABB, Honeywell, Schneider, Emerson, Rockwell, Yokogawa, and Siemens. In addition to this there are several smaller companies, in particular PLC-developers (Programmable Logic Controller). The industry has recently gone through a technology shift, where the focus has changed from being primarily control hardware vendors to being software companies. ABB has coined the term Industrial IT to denote this type of industrial software engineering technology.

The focus of industrial automation is obviously control. However, there are large amount of other factors and functions that are of equal importance. In addition to performance, predictability, reliability and availability are factors that are crucial for process control. Support for reusable application solutions is very important. The system architecture must scale well from very small systems to very large systems, e.g., enterprise-wide automation systems. The systems must be secure against intrusion with security mechanisms that allow all operations to be access controller and audit trailed. Safety certification according to the SIL classification is important in order to become under consideration for safety-critical application types. Dynamic integration of new functionality on-line without any needs for changing or recompiling the existing software is important in applications with high requirements on availability. Integration of heterogeneous hardware and software components of different origin is another requirement.

From a control point of view a special characteristic of industrial automation system is the fact that the systems are programmable and have users which need to interact with the controllers. The industrial automation vendor provides a development environment in which their customers can implement their control logic applications, together with compilation, distribution, downloading, and run-time tools. The control logic is expressed in a domain-specific programming language rather than in a general programming language. The standard that is commonly used is IEC 61131-3. This standard integrates five different programming languages, of which three are graphical and two are text-based. The graphical languages are relay ladder logic (for logical control), sequential function charts (a state-transition formalism intended for sequential control problems), and function block diagrams (a block-oriented signal flow language). The textual languages are structured text (a Pascal or C type language) and instruction list (a stack-oriented PLC-type language). The IEC-61131-3 standard also specifies tasks that are associated with programs, have priorities, and can be periodic or event-driven.

When a IEC 61131-3 program is compiled for execution in some target machine then the programs are translated into native code and the tasks are mapped onto native threads.

Often, a conventional RTOS, e.g., VxWorks, are used in the implementation platform. However, it is not always that each IEC 61131-3 task is mapped to an individual thread. For example, in ABB's system it is only the highest priority task that is mapped to an individual thread. All the remaining IEC 61131-3 tasks are executed by a single native thread. Schedulability analysis is normally not supported. However, the system has support for adjusting the task periods in the case of overruns. Long and varying input output latencies is a problem that in certain cases. One reason for this is use of different types of field buses for input and output.

Several companies also provide special control systems for more advanced control functions, e.g., Model Predictive control. These systems are typically not programmable in the same sense as IEC 61131-3 based systems. Instead, quite conventional real-time programming implementation techniques are typically used.

In addition to IEC 61131-3 or as a technique for implementing the system level software several companies use internally developed component or module frameworks. For example, ABB supports modules in addition to IEC 61131-3 programs. A module gives a higher level of re-use and can to some extents be compared with an object. Honeywell has an internal component-based systems called URT (unified real-time).

The fact that industrial automation systems are programmable and have end users that interact with the controllers generate special requirements, for example concerning verification. Safety classification implies requirements on verification. However, in an automation it is only the development environment and run-time environment that is the responsibility of the automation company. Also, if this is verified correctly there is nothing that prevents the customer from writing an application that is incorrect. Hence, verification is needed at two levels. The verification of at the user programming level should ideally involve the entire closed loop application including the dynamics of the controlled plant. Hence, a hybrid approach is necessary. The fact that the systems that end-users, e.g., process operators, that potentially may interfere with the system also creates strong demands on user acceptance.

Industrial robotics lies somewhere in between the more traditional embedded control applications and the more programmable industrial automation systems. The basic positioning control of the robot arm is normally embedded into the robot system. Here, dynamic scheduling is the predominant solution, although time-triggered solutions may be used at the most time-critical level. Again, schedulability theory is normally not used. Special requirements on openness are caused by the need to integrate external sensors, e.g., force and vision sensors, with the basic position control. On the task level the systems are programmable, often using domain-specific robot programming languages.

2.4 Medical equipment

Medical equipment are examples of embedded systems that are very safety-critical. The following description is based on the inputs from the Maquet Critical Company AB

(MCC).

MCC develops and manufactures devices for intensive care and surgical workplaces. Example products include ventilation and anesthesia medical devices. The different product areas are today diversified considering technical solutions implemented. The main customers are hospitals where the end users are nurses and medical doctors.

Ventilation and anesthesia devices mix and supply gas under controlled conditions to patients for purposes such as to assist sick patients in persons breathing. The control applications thus has the overall purpose to control the mixture of gases, and the gas pressure and flow. Detection of hazardous situations and user interfaces including alarms are an important part of the devices.

MCC uses model-based development. Matlab/Simulink is used in the early analysis of the controller to understand the problem and to do the first closed loop. The controller is then implemented in C and tested in the target system. For the higher level of application SW, e.g. the patient monitoring system and user interface, C++ is used. No code generation tool is used but it is considered that this would be feasible for future development of the controller.

MCC has a single processor controller. The controllers are interrupt-driven and an OS is used for the application. Design margins are used to ensure that deadlines are met. The typical CPU utilization is around 80 %. No special techniques are needed to handle temporal non-determinism. Jitter and latencies are currently so small that they need no be accounted for.

2.5 General Remarks

Practices and development processes vary to a large extent across different application domains, also within a single industrial sector. While some domains are characterized by control-theoretic and model-based approaches supporting the development, other domains, such as automotive engine control, rely heavily on look-up tables and calibration of systems for control purposes i.e. there is no tradition of model-based control. Typically, domain specific tools are in use (some examples: Matlab/Simulink are dominating in general, SCADE is a tool used pre-dominantly in the European aerospace industry, railway control and industrial automation tend to use IEC-61131-3 compliant modeling environments).

Companies with more mature processes utilize tool chains typically starting from functional design (Matlab/Simulink/Stateflow + Modelica, or Scade) using rapid prototyping (through code generation and prototyping hardware), software in the loop simulation, production target code generation, and reuse of plant models in hardware in the loop simulation. Less advanced companies still tend to use e.g. Matlab/Simulink, but with no or little connection to the embedded systems implementation.

The maturity of model based development thus varies a lot but is in general more devel-

oped in Control for embedded systems compared to software development for embedded systems in general [Törngren and Larses, 2004]. The introduction of tool chains supporting model based control engineering is not unproblematic and is strongly related to and by affected by organizational, process and technology constraints. Introducing tool chains causes a reliance and dependence on particular tool vendors and requires training of personnel. For OEMs, transitioning from specification of control systems to actually implementing them is large step and has many implications including maintenance of in-vehicle software and possibly also a larger responsibility. On the other hand, it gives the OEMs better control of vital vehicle functionality.

Many OEMs only specify control systems which are in turn developed by subsystem suppliers (this is typical for the automotive industry for example; e.g. BMW top models have some 60 ECUs where most are developed by some 30 subsystem suppliers). Increasingly, models such as Simulink diagrams are used in the communication between organizations.

Some companies have organizations in which control engineers are co-located with implementation engineers, facilitating co-design. In other companies, control engineers are located in a function development departments which are separated from implementation engineers – in which co-design and understanding becomes a large problem (word documents as specs are passed over the boarders). Especially for the later category, code generation tools are seen as a solution improving the interactions between the groups.

Chapter 3

Real-time and control systems: Design aspects

3.1 Control Loop Timing

A control loop consists of three main parts: input data collection, control algorithm computation, and output signal transmission. In the simplest case the input and output operations consist of calls to an external I/O interface, e.g., A-D and D-A converters or a field-bus interface. In a more complex setting the input data may be received from other computational blocks, such as noise filters, and the output signal may be sent to other computational blocks, e.g., other control loops in the case of set-point control. The complexity of the control algorithm may range from a few lines of code implementing a PID (proportional-integral-derivative) controller to the iterative solution of a quadratic optimization problem in the case of model predictive control (MPC). In most cases the control is executed periodically with a sampling interval that is determined by the dynamics of the process that is controlled and the requirements on the closed-loop performance. A typical rule-of-thumb [Åström and Wittenmark, 1997] is that the sampling interval h should be selected such that

$$\omega_c h = 0.2\text{–}0.6, \tag{3.1}$$

where ω_c is the bandwidth of the closed-loop system. A real-time system where the product $\omega_c h$ is small for all control tasks will be less sensitive to scheduling-induced latencies and jitter, but it will also consume more CPU resources.

3.1.1 Timing Parameters

The basic timing parameters of a control task are shown in Figure 3.1.

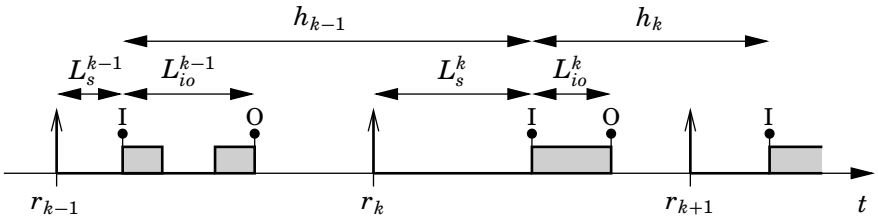

Figure 3.1: Controller timing.

It is assumed that the control task is *released* (e.g., inserted into the ready queue of the real-time operating system) periodically at times given by $r_k = hk$, where h is the *nominal sampling interval* of the controller. Due to preemption 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* is quantified by the difference between the maximum and minimum sampling latencies in all task instances,

$$J_s \stackrel{\text{def}}{=} L_s^{max} - L_s^{min}. \tag{3.2}$$

Normally, it can be assumed that the minimum sampling latency of a task is zero, in which case we have $J_s = L_s^{max}$. Jitter in the sampling latency will of course also introduce jitter in the sampling interval h. From the figure, it is seen that the actual sampling interval in period k is given by

$$h_k = h - L_s^{k-1} + L_s^k. \tag{3.3}$$

The *sampling interval jitter* is quantified by

$$J_h \stackrel{\text{def}}{=} h^{max} - h^{min}. \tag{3.4}$$

We can see that the sampling interval jitter is upper bounded by

$$J_h \leq 2J_s. \tag{3.5}$$

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 the *input-output latency*, denoted L_{io}. Varying execution times or task scheduling will introduce variations in this interval. The *input-output jitter* is quantified by

$$J_{io} \stackrel{\text{def}}{=} L_{io}^{max} - L_{io}^{min}. \tag{3.6}$$

Computer-based control theory normally assumes deterministic, in most cases periodic sampling and negligible, or constant, input-output latencies. If this is true sampled control theory can be applied and the input-output latency can, if it is small, be ignored, or, if it is constant, be included in the control design problem and, hence, compensated for.

In embedded and networked control systems these assumptions do no longer necessarily apply. The execution, task scheduling, and communication can cause temporal nondeterminism in the form of sampling jitter, considerable input-output latencies with potentially large jitter, and lost samples (for example caused by lost communication packets). These implementation effects may influence the control performance negatively, or even, cause instability.

3.1.2 Control Performance

The performance of a control loop can be measured in several ways. In servo control problems the main task is to follow a desired setpoint trajectory as closely as possible. Here measures associated with the time-domain response to a step change in the setpoint are commonly used, e.g., rise time, overshoot, and settling time. Alternatively, one can use frequency domain measures such as cross-over frequency, closed-loop bandwidth, amplitude margin, or phase margin.

In regulatory control problems the main task is to keep the controlled variable at a constant value in the presence of disturbances. For these types of control problems variance-type measures are natural. This is often expressed in terms of a quadratic cost function, i.e.

$$J = \lim_{T\to\infty} \int_0^T x^T(t)Qx(t) + u^T(t)Ru(t)dt$$

where $x(t)$ is the state of the system, $u(t)$ the control signal, and Q and R quadratic matrices. Alternatively one can use measures based on the integral of the absolute error (IAE).

The above performance measures mostly concern single control loops. In industrial applications it is more natural to consider application-specific performance measures. These typically concerns the performance of the overall system, involving multiple cooperating controllers implemented as one or several tasks. Hence, the performance is related to the goal or mission of the system rather than the performance of the individual loops. For example, in an industrial robotics control loop the main concern is the position and movement of the robot hand holding some type of tool. The performance of this is a complex function of the performance of the link controllers for each of the robot's individual links.

3.1.3 Effects of temporal nondeterminism

The effects of temporal nondeterminism on control performance are not always simple to analyze. To start with, already the effects of different scheduling policies and communication protocols on the loop timing parameters are difficult to analyze. Secondly, the effect of the nondeterminism in these parameters on the control loop performance is equally difficult to analyze in general. From a control perspective, sampling jitter and latency jitter can be interpreted as disturbances acting on the control system. The

input-output latency decreases the stability margin and limits the performance of the system. If the jitter and the latency are small, they could be ignored. Otherwise, they should be accounted for in the control design or, if possible, compensated for at run-time.

The input-output latency caused by task execution and communication latency can be modelled as a time delay or dead-time at the input or output of the controlled system. Time-delay systems belongs to the class of *functional differential equations* (FDEs) which are infinite-dimensional, as opposed to ordinary differential equations. Time-delay systems have been an active research area for the last forty years, see [Richard, 2003] for a survey. 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. However, there are always exceptions. For systems that are conditionally stable with respect to the phase shift, [Albertos, 2002], a time delay can have a stabilizing effect. However, in that case this delay should already be accounted for in the controller and the additional delay caused by the implementation will only have a detrimental effect. Also, most examples in which delay can help in stabilizing closed-loop dynamics can also be easily stabilized using rational controllers [Albertos *et al.*, 2000]. It is also in most cases so that a shorter but varying latency is better with respect to control performance than a longer, but constant, latency, also if the latter latency is compensated for. However, also here there are counterexamples, although these example are in most cases quite unrealistic.

Sampling jitter and sampling jitter have in general a negative effect on performance, although again there are counterexamples. As a rule of thumb, relative variations of sampling intervals that are smaller than ten percent of the nominal sampling interval need not be compensated for. The sensitivity to sampling period variations is larger with systems that use slow sampling and for systems with small phase margins; for such systems small variations in sampling period can lead to instability.

3.2 Real-Time Task Models for Control

Several real-time task models have been proposed for, or motivated by, control applications. The most commonly used of these is the hard real-time model. However, also adaptive task model, subtask models, and weakly hard models are to some extent relevant in this context.

3.2.1 Hard real-time model

The hard real time model is the classical task model among these. The traditional view in the hard real-time scheduling community is that a digital controller can be modeled as a task that has

- a fixed period,

- a hard deadline, and
- a known worst-case execution time (WCET).

While this simple model is appropriate for some control applications, there are also a number of cases where it is either too simplistic or too rigid.

First, the model is only concerned with the scheduling of the pure *control computations*. However, in a control task also the input data collection and the output signal transmission are important. Since the input and output operations are neglected in the basic task model, there are no means of controlling the latency and the jitter in the scheduling design. However, a large number of scheduling algorithms have been proposed with the aim to minimize jitter, see, e.g., [Audsley *et al.*, 1993; Baruah *et al.*, 1997; Lin and Herkert, 1996; Baruah *et al.*, 1999a].

Second, the model assumes that worst-case estimates of the controller execution times are available. In practice, it is very difficult to obtain a tight bound on the WCET. Few analytical tools are available, and the ones that exist cannot handle all the features of modern computing hardware, such as multi-level caches, pipelines, and speculative branching. The net result is that WCET estimates tend to be very pessimistic. Furthermore, some control algorithms have varying execution-time demands. One example is hybrid controllers that switch between different internal modes. Another example is model predictive controllers (MPCs) that solve an optimization problem in each sample. Basing the analysis on the WCETs of these controllers may lead to very low average utilization of the processor.

Third, the model assumes that all controller deadlines are hard. This is only a simplifying assumption, however. The sampling rate of a controller is typically chosen to be several times faster than what is dictated by Shannon's sampling theorem and pure stability concerns. A single missed deadline only means that the current sampling interval becomes somewhat longer. The net effect can be interpreted as a small disturbance acting on the process. Only in the case of longer CPU overloads will the stability of the plant be endangered. Hence, the hard real-time model is seldom something that is imposed by the process under control. A time-driven feedback control loop can occasionally miss a deadline without any serious consequences. If that not should be the case then the controller has been incorrectly designed from the beginning, e.g., the sampling period has been chosen too large. Rather, the hard real-time model is something that is imposed by the designers. By adopting the hard real time model the control designer can utilize the result of sampled control theory. The increased level of determinism is also advantageous if formal verification methods are applied. It is also a way of introducing a safety margin into the system.

In discrete-event based control, though, truly hard deadlines are more common. These controllers have to respond to external events within certain deadlines by providing the correct output actions.

3.2.2 Adaptive real-time model

So, if time-driven feedback control is not really hard in nature it is on the hand not really soft either. Even if missing an occasional deadline might not be a problem, missing too many in a row, or missing a deadline with too much, may have serious effects. It is also so that missing a deadline at some times is more serious than at other times. For example during transitions, e.g., setpoint changes, a missed deadline may be more noticeable than in stationarity. An alternative task model that has several advantages is the *adaptive task model*. In this model it is assumed that missing a deadline does not jeopardize system behaviour, but causes a decrease in the control performance. Using this approach the control performance or Quality of Control (QoC) is viewed as a Quality of Service (QoS) parameter. However, again the relationship between deadline overruns and performance is non-trivial.

3.2.3 Subtask models

Sub-task scheduling models are well suited for control. A control algorithm can naturally be divided either into two or four parts. In the first case the algorithm is structured into one *CalculateOutput* part and one *UpdateState* part. CalculateOutput should only contain the code statements that are absolutely necessary to calculate the control signal based on the current measurement signal. In UpdateState the internal controller state is updated and precalculations are performed for the next invocation of CalculateOutput in the next sample. The sampling of the controlled variable is performed at the beginning of CalculateOutput and the output of the control signal is performed at the end of CalculateOutput, before the execution of UpdateState. In this way the input-output latency caused by task execution will be minimized. Alternatively, the sampling and the output are also considered as separate subtasks, that possibly are scheduled separately.

Several sub-task scheduling models and offset-based scheduling models have been proposed in the scheduling community. These include the multi-frame model [Baruah *et al.*, 1999b] and the serially executed subtask model [Harbour *et al.*, 1994]. Subtask scheduling of (for instance) controllers is considered in [Gerber and Hong, 1993; Gerber and Hong, 1997]. There, the objective is not to achieve better control performance but to enhance the schedulability of the task set under fixed-priority (FP) scheduling. An automatic program slicing procedure is suggested where tasks are decomposed into subtasks based on program dependence graphs.

A schedulability motive is also given in [Burns *et al.*, 1994], which deals with FP scheduling of tasks that consists of two parts: one that produces an output, and one that does not. Only the first part needs to complete before the deadline. The analysis is not applicable to controllers, however, since the Update State part of a control algorithm does produce an output (i.e., the new state).

In [Crespo *et al.*, 1999; Albertos *et al.*, 2000b; Balbastre *et al.*, 2000], subtask scheduling of control tasks is considered. Each task is decomposed into three parts with different

priorities: the input operation (medium priority), the control computation (low priority), and the output operation (high priority). The goal of the scheduling design is to minimize the input-output jitter (here termed "the control action interval"). As a side effect, the average input-output latency increases (despite the somewhat misleading title of [Balbastre *et al.*, 2000]). In [Cervin, 1999] a subtask scheduling approach for control tasks is proposed that uses a heuristic method for assigning deadlines, and hence priorities, to the different sub tasks

3.2.4 Other task models for control

Task models based on imprecise calculations [Liu *et al.*, 1987; Chung *et al.*, 1990], [Liu *et al.*, 1994], have also been motivated by the needs of control. Milestone-based methods use monotone algorithms, or anytime algorithms, that gradually refine the result, and where each intermediate result can be returned as an imprecise result. Sieve function methods can skip certain computation steps to tradeoff computation quality for time. Multiple version methods have different implementations with different cost and precision for a task.

Scheduling of monotone imprecise tasks is treated in [Chung *et al.*, 1990]. A task that may be terminated any time after it has produced an acceptable result is logically decomposed into two parts, a mandatory part that must be completed before deadline, and an optional part that further refines the result generated by the mandatory part. The refinement becomes better and better the longer the optional part is allowed to execute.

Imprecise task models only apply to control in particular cases. One example is model-predictive control (MPC). In an MPC, the control signal is determined by on-line optimization of a cost function in every sample. The optimization problem is solved iteratively, with highly varying execution time depending on a number of factors: the state of the plant, the current and future reference values, the disturbances acting on the plant, the number of active constraints on control signals and outputs, etc. An MPC can be viewed as an anytime algorithms of the milestone type. In each sample, once a feasible solution to the optimization problems has been found (the mandatory part), the quality of the control signal is gradually refined for each iteration in the optimization algorithm (the optional part), up to a certain bound. This makes it possible to abort the optimization before it has reached the optimum, and still obtain an acceptable control signal. The approach is reported in [Henriksson *et al.*, 2002b; Henriksson *et al.*, 2002a].

The *weakly hard task model* [Bernat *et al.*, 2001] is one example of a task model with associated schedulability theory that is able to guarantee that a task will meet n deadlines within a window of m consecutive task invocations. One of the motivating examples used is automatic control systems. The problems with this is that it is extremely difficult to assign values to n and m using control-theory arguments. In [Liu *et al.*, 2003] a related window-based metric is introduced based on a Markov chain process. The Markov chain is used to describe the desired stochastic job overrun behavior. In the paper it is claimed that this model can be directly related to the control system's performance. Statistical real-time scheduling in general, is an area where much work has been performed. It is

clear that information such as average response times or deadline overrun probabilities can be of large use in control system design in particular in stochastic control.

3.3 Embedded Control System Issues

The basic characteristics that can be found in embedded systems can be summarized as: small size, autonomous operation, reconfigurability, safety, fault-tolerance and capable of operation under missing data operation.

The above characteristics highly condition their use in control applications, where many control loops are competing for the resources and a number of timing and task scheduling problems arise. The general lay-out of an embedded control system is shown in Fig. 3.2, where different sensors, actuators and their interaction with the environment are illustrated.

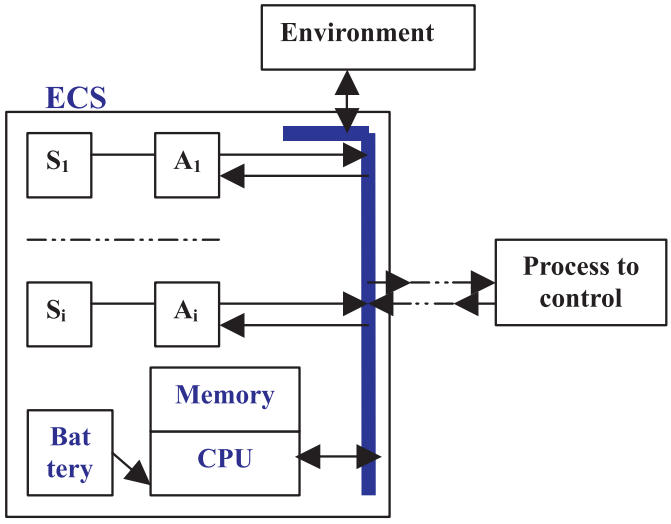

Figure 3.2: Embedded Control System layout

Its main characteristics being the autonomy and the limitation in the resources, one CPU, with its own power supply, must control a number of variables in an uncertain environment. The problems to be considered are related to their implementation, the computational load and resources sharing and the control performance degrading.

1. From the implementation point of view

   - The same resource must be shared between different tasks. As a result of this, the timing of the tasks is not fully determined and the time delays and jitter should be taken into account.

- Alternative control algorithms should be prepared to be switched in. Working in a dynamic environment, the control goals and options may change and the control algorithms should be adequate to new scenarios.
- Working conditions, such as priority, allocated time and memory or signals availability may change. Thus, the complexity, structure and basic properties of the control system will change.
- Variable delays should be considered. The synchronicity of signals cannot be ensured anymore.
- Validation and certification. Any embedded control system should be proved to be reliable and safe operation should be ensured.

2. From the computational point of view

- Economic algorithms should be designed e.g., to reduce the computation time.
- Easy update of information should be provided. The time required to change the parameters of a controller or switch between different controllers should be minimized.
- Hybrid systems should be considered. Logic, discrete time and continuous time information should be merged.
- CPU utilization measurements and optimisation. Being a scarce resource, the current use of the CPU should be measured and corrective actions should be applied to optimise its use.
- Optional tasks must be considered. Control algorithms should be split into mandatory and optional parts, the last ones being only run if time is available.
- On-line scheduling. Based on the current control goals as well as the availability of resources the required tasks should be scheduled in the optimal way.
- Memory saving. Being also a limited resource, computation tasks should rely on few data, also providing a reduced number of intermediate results to reduce memory storage capacity and accessing time.
- Economic hardware redundancy. To increase the reliability, hardware redundancy may be implemented if its cost, size and involved complexity is affordable.
- Fault detection and isolation As well as reconfigurability, this is necessary to allow autonomous behaviour.

3. From the control algorithm point of view, specific design methodologies should be used

- Reduced order models. The complexity of the controller is directly related to the complexity of the plant. More properly, it is related to the complexity of the model used in the design. Thus model reduction techniques should be considered to simplify either the plant model or the designed controller.

- Decision and supervisory control. Changes in the environment and availability of resources should be monitored and changes in the operation mode should be decided by an upper level of control.

- Hybrid control systems. There is also a lot of research on this topic. It is interesting to point out that some authors dealing with embedded control systems only focus on the hybrid character of the controller as there is the need of combining discrete and continuous time dynamics. But this is just one, although fundamental, property of the control system.

- Multimode control. A control problem cannot be approached in the same way if the environmental conditions change. Thus, the control strategy must consider changes in the controller and caution should be taken to guarantee the correct operation (stability is the minimum) of the controlled system.

- Sampling rate changes. It is well known that, in general, the quality of the digital control decreases if the sampling period increases. Thus, if resources are available, the control tasks period should be reduced and this implies changes in the controller parameters as well as in the stored information.

- Non-conventional sampling and updating patterns. The time instants when sampling is performed and when the control signals are generated should be appropriate for the required controlled plant dynamics and the available resources. This implies the option of different sampling rates (multirate control) and to consider the loss of synchronicity.

- Missing data control. Data availability is not always guaranteed in all operating conditions. Thus, control algorithms should cope with missing data situations.

- Event-triggered control. Most control algorithms are time driven. The sampling-updating pattern is regular and synchronous, with a constant time interval between actions (periodic actions). But dealing with harsh and uncertain environments, some activities are triggered by external events which may occur randomly.

- Degraded and back-up (safe) control strategies. Different control requirements should be considered, according to the operating conditions. In emergency cases, a degraded behaviour should be accepted, ensuring the safety of the operation.

- Fault-tolerant control. There is a lot of research and literature on this topic. It is crucial for embedded control systems as there is no option to externally reconfigure the control structure. Thus, supervisory control involving fault detection and isolation as well as a decision system to select the most suitable controller must be considered. Alternatively, time-invariant fault tolerant controllers can be considered if performance degradation is allowed in faulty conditions.

- Battery monitoring and control. Any autonomous system should adapt its activities to the available power. The monitoring and control of the on-board batteries will result in forced changes in goals and control structures.

4. From the hardware point of view specific different implementations structures may be considered [Cumplido *et al.*, 2005]

- Standard CPUs, as previously considered, with the inherent advantages of using a general purpose device, support for multi-operation, and reusability, but with the disadvantage of involving extra facilities not required for the particular application.
- Standard DSP units, where special controller operations should be implemented.
- Special purpose processors based on application-specific integrated circuits (ASICs), designed for control purposes. In this case, modularity is a major issue in order to provide flexibility in the design.

The decision about the hardware structure will affect the general programming effort and specially the real-time issues, as well as the control performance connected to precision and time responses.

3.3.1 Real-time issues

The embedded operating system usually should have the following characteristics:

- Configurable and scalable: the design of operating systems based on components permits the selection of the appropriate components to build the specific kernel for the embeddable application.
- Innovative techniques in scheduling to select the most efficient scheme for the application combining different policies at several levels of scheduling.
- Resource management allowing the use of Quality of Service techniques improving the efficiency and flexibility of a real-time system. An efficient approach to real-time resource management consists of applying feedback control theory to real-time scheduling. So, a scheduling strategy naturally adapts to the application needs and the resources allocated to an application are automatically updated to its needs.
- Fault-tolerance mechanisms which are able to make the results available on a timely basis even if this fact could lead to a functioning in "degraded mode". This is a pre-required quality for real-time embedded systems.
- Small footprint to embed the system in different configuration.
- Multi-platform. The embedded operating system has to be portable or recompilable for different processor architectures.

- Power aware techniques. Development of systems supporting dynamic power management strategies based on dynamic voltage/frequency scaling for tuning the power-performance trade-off to the needs of the application.

- Efficient memory management including special forms of memory management that limit the possibility of memory fragmentation, and assure a minimal upper bound on memory allocation and de-allocation times.

- On-line re-scheduling. To take into account changes in the control relevance of some actions [Albertos *et al.*, 2000a].

- Algorithms for dynamic memory allocation in real-time systems.

- Development of open components for real-time embedded operating systems [OCERA, 2002].

- Scheduling policies to enhance quality of service, [Marzario *et al.*, 2004].

3.3.2 Control issues

Embedded control systems, by their own conception, are application-oriented and thus, the encountered control problems very much depend on the application. Anyway, a number of common issues can be outlined

Controller design under non-conventional sampling patterns. Classical control theory assumes a regular and uniform sampling and updating pattern for all the signals. But this is a theoretical constraint which can be violated in practical applications or even can be specially designed to achieve improvements in the control performances by taking into account the sampling pattern when designing the controllers. Some previous works in this direction are reported in [Albertos and Crespo, 1999], [Albertos and Crespo, 2001].

Controller parameters and data updating under sampling rate changes. Changes in the CPU load demand changes in the control tasks periodicity. The controlled plant being continuous time, a change in the sampling period involve changes in the controller discrete time model as well as in the stored data [Albertos *et al.*, 2003b].

Missing data control. Due to communication congestion or data acquisition systems failures in uncertain environments reliable output estimators (virtual sensors) to cope with scarce or missing data should be developed. A general perspective of this problem is discussed in [Albertos *et al.*, 1999] and [Sanchis and Albertos, 2002].

Control kernel. As previously discussed, changes in the operating conditions require changes in the controller. Under any working condition, the system must keep some basic properties. If this is not the case, the emergency routines should take care of the system by moving it to a safe, even shut-down, situation.

These basic properties should be captured by a kernel representation of the system in order to apply an essential control. This control should be able to ensure the system stability under a shortage of resources. Thus, in this case the system will fall under this essential control, until more resources (options) become available.

As a result, the following challenges emerge:

Research challenges

- Prioritize the control activities, from the point of view of safety and relevance in the operation
- Fault-tolerant control design, in both hardware and software
- Co-design of the control algorithm and its implementation in a multitasking, resource constraint environment
- Optimize the control algorithm design and its implementation
- Robustness of the control against uncertainties in the timing, delays and resource allocation
- Designing control systems considering the execution platform and its constraints is one of the major challenges.
- Structure of the control code. The control kernel should cope with the fundamental activities.
- A larger variety of sensor sources and actuators to interact with will be integrated to interact wit a wider range of processes. Standards for sensors and actuators integration will facilitate it.
- Adaptability to changes in the environment.
- Easy tools for control solutions development, testing and validation.
- In platforms which controls several independent processes the temporal and spatial isolation of applications is required for safety purposes.
- In real-time operating systems for embedded control systems the ability to build several partitions where different specific real-time operating systems for their applications requirements is one of the important issues. In addition, the run-time system must be certifiable to a level of criticality as high or higher than that of any program running on the processor.

Chapter 4

Integrated real-time and control design: Techniques

4.1 Design Process for Control System Implementation

Here a process is proposed that takes the temporal non-determinism of the implementation platform into account in various ways. The design process can, alternatively, be viewed as different maturity levels for implementation-aware control systems. The process consists of the following steps:

1. Maximize temporal determinism
2. Temporal robustness analysis
3. Control-based off-line compensation
4. Control-based on-line compensation
5. Control and scheduling codesign
6. Feedback scheduling

The first step aims to separate the concerns of the control application from the implementation level concerns. If this is successful than no additional steps are needed. However, as pointed out previously, a complete separation is seldom achieved. Step two involves analyzing how sensitive the control loop is to temporal nondeterminism, i.e., to analyze the interactions. Steps three and four are focused on how to use control techniques to compensate for the nondeterminism. Step four discusses different approaches to an integrated design of the control and computing subsystems, whereas the approach

in step five is based on dynamic, feedback-based, allocation of computing resources to controllers.

4.1.1 Maximize temporal determinism

The first step in any controller implementation is to try to maximize the temporal determinism, i.e., chose an implementation techniques in such a way that the jitter is minimized and the latency is either minimized or made constant. This involves both computing and control.

For applications with very high demands on temporal determinism a time-triggered static-scheduling approach is best, e.g., the time-triggered architecture (TTA) proposed in [Kopetz and Bauer, 2003]. Using this approach it is possible to guarantee that deadlines are met and provide guaranteed tight upper and lower bounds on latencies. For distributed applications with networked control loops the static scheduling should be done globally involving both the computations and the communication. A drawback with static scheduling approaches is that they can be quite inflexible, does not scale so well, and and are sometimes incompatible with existing non-real-time legacy applications or components based on event-driven scheduling. However, for safety-critical applications a static approach has strong advantages.

Alternatively event-driven scheduling using either fixed priorities or dynamic priorities can be applied. Modern schedulability analysis techniques makes it possible to analyze both the computations and the communication. However, as stated before the emphasis is often only on ensuring that deadlines are met, disregarding the latency jitter.

In addition to the pure implementation techniques above it is also possible to apply other techniques to maximize the temporal determinism. One 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, $L = T$, of one sample is achieved, something which is particularly easy to compensate for in the control design. However, also if the latency is compensated for the control performance often becomes worse with this approach than to allow a shorter, albeit varying, latency. In a networked control loop with varying communication delays it is possible to introduce buffering to get a constant delay.

A similar separation has been suggested in the embedded systems programming model Giotto [Henzinger *et al.*, 2001]. 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. 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].

If the worst-case execution times of the controller tasks are small compared to the controller task periods, and also small compared to the execution times and periods of the other tasks in the system, then non-preemptive scheduling of the controller tasks and preemptive scheduling of the remaining tasks may also be an option. This will minimize the input-output latency at the price of possibly more sampling jitter.

4.1.2 Verify robustness to implementation effects

Guaranteeing a very high level of determinism can both be difficult and costly. In event-driven scheduling it often leads to relatively low CPU utilization, which not always can be tolerated for other reasons. The difficulties associated with worst-case execution time analysis and measurements also adds to the problem.

Using temporal robustness analysis techniques it is possible to decide how much sampling jitter and how large variations in input-output latency that a particular control loop can tolerate before loosing performance or becoming unstable. This information can then be used to relax the requirements on the implementation platform or to verify that the level of determinism that a given implementation platform can provide is acceptable.

There are several approaches that partly can be applied to verify temporal robustness. The MATLAB/Simulink-based tool TRUETIME [Cervin *et al.*, 2003] allows cosimulation of real-time kernels and networks, controller tasks, and the dynamics of the controlled processes. Using TRUETIME is possible to evaluate temporal robustness through simulation. JITTERBUG [Lincoln and Cervin, 2002] is a MATLAB-based toolbox that makes it possible to compute a quadratic performance criterion for a linear control loop under various timing conditions without resorting to simulation. Using the toolbox, one can easily and quickly assert how sensitive a control system is to input-output latency, jitter, lost samples, etc.

The robustness to constant input-output latencies for linear systems is decided by the classical phase or delay margin of the system. Alternatively the phase margin can be parameterized in terms of the gain of the controller or in terms of the control effort, which depends on the difference in control requirements between the open loop system and the closed loop system. The control effort will be described in more detail later on in the roadmap.

An extension of the classical delay margin to time-varying delays has been 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. The jitter margin can also be used when selecting network protocol for networked control loops. A drawback with the jitter margin is that it assumes deterministic sampling, i.e., no sampling jitter is allowed.

Another possibility is to use Lyapunov function theory. The problem of guaranteeing stability of a system in the presence of both sampling jitter and input-output latency can be stated as the search for common quadratic Lyapunov function. This can be formulated as a Linear Matrix Inequality (LMI) problem for which convex optimization routines are available.

4.1.3 Control-based off-line compensation

If information is available about the timing variations then the timing variations can be compensated for off-line at design time. Different types of information are possible, ranging from simple average values or maximum and minimum values, to complete statistical distribution functions.

There are several possible compensation methods that apply to sampling jitter, ranging from simple ad-hoc techniques to quite advanced techniques requiring extensive calculations. The choice of compensation method depends on the range of sampling period variations, how often the changes occur and how sensitive the system is to variations. The cost of implementation is also an important factor. A related issue is intentional changes in sampling period. Most compensation schemes assume that the sampling period variations are unintentional and unknown in advance. This is the case when the variations are due to clock inaccuracy, overload or computer failure.

One approach is to make the nominal design robust to system variations in general. Many robust design methods can be used, such as H_∞, Quantitative Feedback Theory (QFT) and μ-design. Note that the result of variations in sampling delay can be captured by an increased process noise level in the system by writing the system as

$$\begin{aligned} x_{k+1} &= \Phi(h_{nom})x_k + \Gamma(h_{nom})u_k + \overline{d}_k \\ y_k &= Cx_k + Du_k + n_k \end{aligned}$$

where the new process noise

$$\overline{d}_k = d_k + (\Phi(h_k) - \Phi(h_{nom})x_k + (\Gamma(h_k) - \Gamma(h_{nom}))u_k$$

has an increased covariance. This is interesting, because it relates to a specific design method, namely LQG/LTR, a variation of the linear quadratic design method where increased robustness is achieved by introduction of artificial process noise. The method usually results in a faster observer design.

There are also compensation schemes based on optimal control methods and stochastic control theory. A more involved design method, which uses probabilistic information about typical sampling variations, is presented in [Nilsson, 1998].

If the input-output latency is fixed, and known in advance, it is possible to compensate for it in the controller design. If the controller is designed in discrete-time this is particularly simple. The state vector of the system is simply augmented with old values of the control signal. Once this is done any discrete-time design method can be applied. The control delay has the same malign influence on the system as any process delay. The resulting achievable control performance will be decreased the longer the control delay is, but the detrimental effects can be minimized by careful controller design.

When the latency is varying from sample to sample the problem becomes more complicated. Stability and performance of control systems with time-varying latencies have been studied since the late 1950s, e.g. [Kalman and Bertram, 1959; Davidson, 1973; Nilsson, 1998; Kao and Lincoln, 2004]. One possibility is to design for a good estimate of the latency, such as the average value. If probabilistic knowledge of the latency is present stochastic control theory can be used to optimize performance. In [Nilsson, 1998] this situation is studied in the context of networked control loops. The conclusion is that very good performance can be obtained even under quite large variations of latency. The case of joint variations in control delay τ and sampling period h is treated in [Nilsson *et al.*, 1998].

4.1.4 Control-based on-line compensation

If it additionally is possible to on-line measure the timing variations, or in some other way obtain information about them, then it is possible to actively compensate for them, or parts of them.

If the changes of sampling period are rare, or if the sampling period varies slowly, then the problem can be solved using gain-scheduling. This means that several sets of controller parameters are pre-calculated and stored in a table with the sampling rate as input parameter. When the sampling rate changes a new set of controller parameters are used. Small transients can occur when the controller parameters changes and special care must be taken to minimize these mode bumps. For low-order controllers, e.g., PID controllers, it is sometimes possible to include the sampling period in the parameterization of the controller. This can be useful if the sampling period changes frequently, e.g., from sample to sample. The idea must be modified for higher order controllers. If the nominal controller instead is a linear feedback with Kalman filter then a solution may be to use a time-varying Kalman filter that gives state estimates at the actual time instances. More involved schemes can also be used, where also where the feedback gain and the observer gain depend on $h_k = t_{k+1} - t_k$.

A varying input-output latency is difficult to compensate for. However, in a networked control loop the part of the latency caused by the network delay between the sensor node and the controller node can sometimes be measured. The work in [Nilsson, 1998]

on networked control loops assume that this information is available to the controller.

4.1.5 Control and scheduling codesign

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?

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 decreasing attention. A nice survey of the area from the mid 1980s is [Hanselmann, 1987].

In recent years, a number of research efforts have been devoted to control and scheduling codesign of real-time control systems. One of the very first papers that addressed the issue of joint controller and scheduler design was [Seto *et al.*, 1996]. The paper considers optimal sampling period selection for a set of digital controllers. The performance of each controller is described by a cost function (a performance index), which is approximated by an exponential function of the sampling frequency. An optimization problem is formulated, where the sum of the cost functions should be minimized subject to the schedulability constraint. Both fixed-priority and earliest-deadline-first scheduling is considered.

The previous paper does not consider the input-output latency in the controllers. 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; Ryu and Hong, 1998]. 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 goal of the optimization is to minimize the utilization of the task set. A heuristic iterative algorithm is proposed for the optimization of these parameters subject to schedulability constraints. The algorithm is based on the period calibration method (PCM) [Gerber *et al.*, 1995] for determining the task attributes.

Optimal sampling period selection for a set of linear-quadratic (LQ) controllers is treated in [Eker *et al.*, 2000]. Similar to [Seto *et al.*, 1996], an optimization problem is formulated where the total performance of the controllers should be optimized subject to a schedulability constant. Analytical expressions for optimal LQ cost as a function of the sampling interval are derived.

The integration of static cyclic scheduling and optimal (LQ) control is the topic treated in [Rehbinder and Sanfridson, 2000]. A number of processes should be controlled by a set of state-feedback controllers. The performance of each controller is measured by a quadratic cost function. A combinatorial optimization problem is formulated that, given a schedule length, attempts to minimize the sum of the control costs. The solution contains the periodic task schedule as well as the state feedback gains of the controllers. A similar codesign problem is formulated in [Lincoln and Bernhardsson, 2000]. There, the optimal switching sequence (i.e., schedule) for a set of controllers should be found. Unlike the previous paper, however, the obtained schedule is not necessarily periodic.

In [Palopoli *et al.*, 2002], sampling periods for a set of controllers should be chosen such that a certain robustness measure is maximized. An optimization procedure is

used to derive the gains of the controllers together with the sampling periods, subject to a schedulability constraint. The method is limited in its applicability in that it only handles first-order plants.

A codesign method that assumes run-time scheduling decisions based on current plant state information is presented in [Zhao and Zheng, 1999]. There, the controller associated with the plant with the largest current control error is chosen for execution, while the other plants are fed zeros. An algorithm is given that determines the (static) state feedback laws to meet a given performance specification.

The strong interactions between the network architecture, resource allocation, control algorithms and the overall system performance makes it natural to attempt to co-design network and control. Such approaches are of particular interest in resource-scarce systems, such as control systems with a large number of sensors and actuators operating over a shared wireless network or in wireless sensor networks. However, codesign problems can become quite complex and only a few attempts can be found in the literature.

In [Xiao *et al.*, 2003], the develop a method for co-design of linear controllers and estimators, data rates for sensor and actuator data, and resource allocation in the underlying wireless network. The paper [Liu and Goldsmith, 2003] presents wireless communication tradeoffs in distributed control, by mapping coding, modulation and retransmission schemes onto a communication delay distribution, and studying the performance of the associated optimal linear quadratic controllers. The problem of joint estimator and transmit power allocation in a wireless sensor network is studied in [Xiao *et al.*, 2004], where it is shown that sensors with poor communication channels or noisy observations should decrease their rates or even become inactive.

4.1.6 Feedback scheduling

In feedback scheduling the dynamic allocation of computing and communication resources to tasks, jobs, or requests is viewed as a control problem. This can also be applied to control systems. This topic is described more in detail in the complementary Artist2 roadmap on control of real-time computing system.

4.2 Techniques to reduce performance degradation

In this section different techniques that can be used to reduce the performance degradation caused by the implementation platform are discussed in more detail.

4.2.1 Scheduling influence over performance degradation

The implementation of multi-loop control systems implies the definition of a set of tasks running under timing constraints. The task scheduling is a fundamental issue in real-

time control algorithm implementation. But, for high-performance applications, the delays introduced by the digital control due to resource limitations, should be considered. This is mainly the case when the same CPU is used to run a number of control algorithms. In this multitasking environment, the scheduling must consider the effect of these delays on the different control loops.

Some previous work has pointed out the need for an integration of the control design and implementation. Output jitter is the main factor that affects to the performance degradation in a control system. In the previous section, some methods to reduce the jitter have been described. In [Real and Crespo, 2004], a method to consider the period range of the tasks involved in a control design as criteria to obtain an optimal use of the system resources is proposed. In this integrated approach, the implementation is tuned to execute the control tasks at the maximum frequency while ensuring the system schedulability. The result of this approach is a system with a high system (CPU) utilization.

However, the influence of the output jitter or output delay on the performance of the control system can be different depending on the controlled process. Moreover, the output jitter is not the same in any control task. As was stated in the previous section, when the scheduling policy is based on fixed priority (RM or DM) the less priority tasks suffer more important delays than tasks with higher priority. If the scheduling policy is based on dynamic priorities (EDF), the delays are more evenly distributed.

The effect of the delay is not the same in any control task. It is clear that the open-loop control system performance is even not very sensitive to time delays. On the other hand, in feedback control systems, the controlled system behavior can be highly degraded if time delays are not included in the design approach. But the main problem is that controlled systems that seem to be closed-loop well-damped and stable may present stability problems if significant time delays happen. In order to evaluate the time delay effect a new concept, the control effort, has been introduced [Albertos and Olivares, 1999]. The point here is that if a great control effort has been used to achieve the nominal behavior, very short time delays may become significant.

Moreover, the task priority plays a different role from the control viewpoint than from the real-time viewpoint. In general, most control loops have the same priority from the control point of view, although larger sampling periods could imply less stringent timing. But the larger the control effort the more sensitive to delays the control loop is. On the other hand, from the real-time viewpoint, the priority is in general related to the task period or deadline.

The control effort measures the effort needed in terms of shifting of the specification from the open-loop to the closed-loop. In order to better express this concept, state-feedback control is assumed, although the results are easily extended to a general feedback control loop, [Albertos, 2002], and will be practically shown by some examples.

4.2.2 The control effort

When controlling a process, its time response behavior to changes in references and disturbances determines its time scale. A fast process will require short sampling periods and a slow process will not demand such a fast sampling. But, nowadays, this will not result in a problem for timing the digital controllers (CPU). The real problem appears if the control goal is to get a much faster dynamics in closed loop. This will result in a strong “control effort” and, in this case, small delays, loss of data, loss of synchronism or minor changes in the control parameters will influence the controlled plant behavior.

In a state feedback control, the control effort can be measured as the shifting of the poles from open to closed loop. Under sufficiently strong control actions, the reduction in the phase margin if a time delay in the control loop appears is proportional to the cut-off frequency. But if the initial system is designed assuming a known delay, this is included in the calculation of the initial phase margin by using the appropriate technique (Smith predictors and similar) and the same conclusion is valid for any additional delay.

For a given phase margin, usually desirable to be around 60 degrees, the maximum allowable control effort is inversely proportional to the unforeseen possible time delay. For control loops with high control efforts, the influence of the output jitter can be significant. On the other hand, control loops that require lower control effort, the effects of the delay are not relevant.

4.2.3 Control effort and task priorities

Under RM or DM, task priority is assigned considering the period or deadline of the task. This priority assignment that is optimal from the timing constraints, is not the best from the control point of view. In this case, the scheduling policy introduce higher jitter to lower priority tasks.

If the priority assignment criteria was the control performance, the priority should be assigned considering the control effort parameter. In this case, the task with higher control effort will receive the higher priority. So, task priorities are assigned monotonically with control effort parameter of each control loop.

However, this assignation has two effects:

- the system can unschedulable under the control effort criteria and schedulable under RM or DM criteria.
- the delays are shifted to the other tasks.

In order to achieve the best priority assignment from the control effort point view while maintaining the system schedulable the following priority assignment algorithm can be applied: "The priority assignment algorithm starts from a schedulable proposal based on the RM or DM approach. The algorithm promotes those tasks with higher control

effort to upper priority levels ensuring the schedulability of the task set". A detailed description of this algorithm with examples can be found in [Albertos *et al.*, 2000a].

The following methodology allows integrating both design phases: the controller design, leading to a given control effort, and the control implementation, resulting in an estimated average delay.

- As result of the control design, it is possible to determine the control effort of each control loop.
- The schedulability of the partitioned system is analyzed, and the minimum and maximum delays are evaluated.
- If so required, the control design considers the minimum delays as a parameter to redesign the control algorithm, determining updated control efforts.
- A priority assignment algorithm based on the required control effort is applied to the task set in order to obtain a modified priority assignment. As a result, the priority of tasks with higher control effort is increased. The scheduling algorithm has to ensure the system schedulability.

Under EDF scheduling, task priority is assigned dynamically depending on its deadline. The output jitter of a task can be decreased reducing the task deadline. A similar method to the described above can be applied to integrate both criteria. "The priority assignment algorithm starts from a schedulable proposal based on the EDF approach. The algorithm reduces the deadline of those tasks with higher control effort to have a deadline shorter than the next task with lower deadline ensuring the schedulability of the task set".

These methods can also be applied to a partitioned task model like the one detailed in the previous section. In this model, each task is first partitioned into an initial or acquisition part, a mandatory part for the algorithm evaluation and a final part to deliver the action. Minimizing the final part the average delay can be incorporated in the controller design. In this case, the target is the average degradation of control performance which is directly related to both the control effort and the delay. Thus, the scheduling policy distributes the control action interval among the different tasks in order to minimize this index.

4.2.4 Controllers and period changes

It is well known that the performance of a digitally implemented controller degrades as the sampling rate decreases, [Albertos and Crespo, 2001]. In fact, the controlled plant behaves as if in open loop during the inter-sample periods. If the control action is rather strong, or the plant is open-loop unstable, changes in the sampling periods may affect the stability of the plant. In any case, performance degradation appears if the sampling period is enlarged. Thus, if enough computing resources are available, it would

be convenient to execute more frequently these tasks. If the number of tasks increases, or the frequency of some high priority tasks is reduced, the time allocated to evaluate the control algorithm should be decreased. Otherwise, the CPU would be overloaded and the real time constraint will be violated.

Increasing the sampling period will affect the controlled plant performance. To diminish this effect, the controller parameters can be tuned for the new sampling period, trying to reach the same closed-loop performance. Thus, any change in the resource availability should determine a change in the activation rate of the tasks. For the controller tasks, this will imply a variation on the sampling period modifying the dynamic behaviour of the controlled system.

Stability theory for switching control has received the attention of many recent papers, see for instance [Hespanha and Morse, 2002] and the references herein. A general concern is that the stability of the different control structures does not guarantee the stability of the entire switched system, if an arbitrary switching pattern is applied.

In our case, the stability is not a relevant issue because the change in the sampling period, and hence the change of the controller, does not happen frequently. In any case, we will assume that the controlled plant reaches the stationary behavior between to consecutive switching times. On the other hand, the unplanned time of switching may result in bumping oscillations if the transfer is not properly done. What matters in many control applications is to carry out a bumpless controller transfer.

As all the control tasks share the CPU, even with some other tasks, the sampling period updating may happen at any time with respect to a given control task. That is, the controller transfer may be required under any dynamic behavior and transient circumstances. There is no problem if the control loop is at a stationary point. But, if the control loop is in a transient period, in order to avoid or at least reduce the bumping, the decision based system must provide not only the parameters and/or structure of the new controller but also its internal data to ensure a smooth change.

If the controller change is motivated by a tuning of the sampling period, both controllers have the same structure and order, being designed for the same purpose but with different sampling rates. Under these conditions, the new controller should be launched and, even if the controller parameters can be pre-computed and available, the appropriated input-output data are not known, because the stored system signals were taken at different instants of time if an input-output modeling for the controller is used. An approach to reduce the impact of the sampling rate change in this cases has been presented in [Albertos *et al.*, 2003a]. The solution involves the computation of the controller parameters and the estimation of the sequence of input-output signals at the new sampling rate.

When a change on the sampling rate of the task implementing the controller is carried out, the first problem that appears is that the values of the parameters for the controller model have to be evaluated for the new sampling period. In general, another problem appears related with the historical samples corresponding to the previous sampling rate, because these values are unknown at the new sampling time instants. At

[Albertos *et al.*, 2003a] an approximation is provided in order to solve both problems for the input-output modeling case. The problem can be simplified by implementing the control through an internal representation. This will require updating the state feedback law, the observer model and the observer gain, but, on the other hand, the past information of the controller is summarized in the observer's state.

4.3 Control kernel

In real time embedded systems, an operating system kernel should be provided to handle the basic requirements of the whole system even working in the emergency conditions. Similarly, a control solution should be implemented to ensure the safe behaviour of the system even in the case of resource shortage. This implies a reduced order model of the plant, a simplified control algorithm and a reliable operation. These basic properties should be captured by a kernel representation of the system in order to apply an essential control.

General speaking, the kernel representation implies a reduced order model of the system as well as a mechanism to transfer from a normal, extended model, to the kernel and vice versa. Thus, a simple algorithm to transfer between models, also recovering the involved data, should be provided.

The kernel concept must consider to cope with essential goals connected to safety in the operation as well as resource shortage, such as loss of data or time limitation. Thus, in this case, the system will fall under this control until more resources (options) become available. Let us discuss the model reduction issue under time constraints, illustrated by an example.

An embedded control system is used to control a mobile platform with a manipulator arm in the top. Among other emergency conditions, the following circumstances should be analyzed [Albertos *et al.*, 2005]:

- A warning signal about a possible collision is activated. The time allocated for control computation is reduced and a safe operation (with degraded performance) is foreseen. Thus, the sampling period may be enlarged and only the slow part of the plant, dealing with the driving of the platform, could be considered,
- A fast emergency action is required on the arm, the platform movement being irrelevant or null. Under this emergency condition, a quick reaction of the control is expected. The sampling period is reduced and the slow behaviour of the plant may be considered as unchanged.

In both cases a reduced model of the plant is desirable and a change in the model parameters should be implemented. A similar approach will be used if there are several interconnected controlled subprocesses and some of them are more relevant than others.

Under resource constraints, the global model is reduced and only the partial models related to the controlled subprocesses are used. (Some slave signals, such as temperatures or vibrations, are forgotten and the resources are concentrated to control the primary signals.

The same can be done if dealing with the controller model. There are many model reduction techniques but the real-time transfer between equivalent representations should be investigated.

This is the main theoretical idea. Generally speaking, the kernel representation implies a reduced order model of the system and a mechanism to transfer from a normal, extended model to the kernel and vice versa. For that purpose, a simple algorithm to transfer between models, also recovering the involved data, should be provided. The transferring approaches require scheduling support to allow mode changes in the system. A mode change should be initiated whenever the systems detects a change in the environment or in the internal state that must drive it from one operating mode to another allowing the use of reduced models and the transfer between models. Several protocols for mode changes can be found in the literature. A survey on mode change protocols can be found in [Real and Crespo, 2004]. The main requirements to be achieved by the protocols are: schedulability, periodicity, promptness and consistency.

Chapter 5

Real time networks for control systems

A real time communication system is used in distributed control applications where time is a critical factor. Real time messages must be transmitted within precise time bounds to assure coherence between senders and receivers. The response time is a crucial parameter in distributed real time systems. Therefore, it is needed to analyze whole system in order to know, where the delays arise. Worst case end to end communication delay for a message must be known and bounded. The communication delay is measured from the instant when the first bit of message is sent out from the transmitter to the instant when the last bit of message is delivered to receiver. The transmission delay is calculated as time difference between the beginning of transmission of the first bit and the end of transmission of the last bit of the message. The propagation delay depends on parameters such as the type of physical medium, the distance between the source and the destination, and baud rate. In the shared medium the collision delay is added and the device competes for the access to this medium at this time. If the medium access is deterministic, the calculation of the collision delay is relatively easy. But the analysis tends to be infeasible when the media access is random (non deterministic). This delay has the most importance to the total communication delay. After the entire packet has completed reception at the destination, the number of bit errors that occurred during communication is allocated. This result is generally dependent on a bit error probability which reflects the quality of the link, and the message length. Depending on error allocation (how many errors occurred), the error correction procedure is executed.

Deterministic systems reserve system resources before operation, which prevents resource contention. Communication systems usually achieve determinism by cyclic operation, using division of media access under either a master controlled, token passing, or clocked operation.

Following characteristics of real time communication systems are vital in control appli-

cations:

- periodic transmission with short periods and low jitter
- presence of aperiodic traffic like alarms, which need to be reliably transmitted with bounded latencies
- safety critical messages must be transmitted reliably within an application-dependent deadline
- there should be support for message priorities
- messages with stringent timing constraints typically have a small size
- fault tolerant which allows node and network failures without loss of functionality

5.1 The access to the shared communication medium

The media access control is important as the communication medium is shared by several devices, and only one of them can transmit at a given time. In the synchronous access methods, the time is divided into slots and each of devices have predefined slot, when it can transmit to the medium. The synchronous access methods are typical for time triggered architecture, where all information about the behavior of the system, e.g., which node has to send what type of message at a particular point in time, is known a priori (at design time) to all nodes. The clock synchronization is necessary to provide all nodes with an equivalent time concept. The clock can be synchronized externally by the specific node or autonomously by the nodes themselves. The messages are usually transmitted without addresses and are identified by the time instant at which they are transmitted. This method has predictable network load. A time triggered communication system is ideal for applications in which the data traffic is of a periodic nature. The main method based on the synchronous access is time division multiple access (TDMA) and protocols using this method are for example TTP, FlexRay or TT CAN.

No prior time division is made by asynchronous access. This control access can be further divided into random and deterministic access. The random (uncontrolled) access methods are typical for even triggered architecture, where the data transmission is generated by event occurrence. This architecture has an unpredictable network load if arrival rate of events can not be bounded. In a purely event triggered systems it is impossible to implement a timely membership service at the protocol level. The main method belonging to the random access is the carrier sense multiple access (CSMA). Protocols using CSMA method are for example CAN (CSMA/DCR) or Ethernet (CSMA/CD).

The deterministic (controlled) control access can be divided in the centralized and distributed access. The typical centralized control method is master/slave, where any node transmits messages only upon receiving an explicit command message issued by one particular node called master. The merit is relatively precise timeliness depending on

the master. The drawbacks are a considerable protocol overhead and inefficient handling of event triggered traffic. The typical master/slave protocols are WorldFIP and FFT Ethernet. The distributed control method is represented by token passing, where one token circulates in a logical ring and when the device gets the token, then it makes its own transmission and passes the token to the next device in the ring. Real time operation requires bounded token holding time. The typical token passing protocol is Profibus.

Variants of above mentioned access methods can be found in several communication protocols originated in the field or office area. Both wired and wireless networks are of crucial importance for control applications.

5.2 Fieldbus systems

Many of the embedded control systems are distributed today and very important part of these systems is the field area, which connects the processing power with sensed/driven objects usually via third party products. That is why the communication protocols exhibiting good real-time properties [Tindell *et al.*, 1995; Almeida *et al.*, 2002; Tovar and Vasques, 1999; Krakora and Hanzalek, 2004] are widely used in the filed area and they are usually called fieldbus systems. A distributed control system based on a fieldbus technology has various advantages in comparison with the conventional centralized one:

- As there is usually one configuration tool for all devices it is easier to configure and parameterize the distributed system. Also other engineering work like preventive maintenance and system start-up is much faster. This lowers the maintenance costs.

- As there is one fieldbus going through all components there is considerable saving of cables connecting computer to the technology. Accordingly, the installation is easier and faster.

- The resulting distributed system is more flexible than the centralized one. This leads to the improved functionality and reduced down time. In addition, the fieldbus systems are designed in such a way that they offer fast, accurate and reliable diagnostics.

The above mentioned reasons contribute to the fact that the fieldbus technology is used in all industrial branches dealing with perception and control (including automotive, production lines, process engineering, machine internal architectures, power distribution, building automation, supervision systems). Today, fieldbus technologies are mature and widely used in distributed control applications.

5.3 General purpose networks

Today, Ethernet is widely spread in office computer communication, which implies high availability and low implementation cost. Therefore well known Ethernet is very attractive for the embedded systems area due to high performance, wide diagnostics possibilities (using several protocols simultaneously running on the same medium) and easy integration with office networks (important for logging, management and multi-level integration, e.g. CIM). But distributed control applications have two main requirements for data delivery: time determinism and reliability. Ethernet in general is not deterministic due to its media access control (CSMA/CD) and therefore its behavior under transient overload is not sufficient for any real-time application. Moreover Ethernet has high communication overhead, high computing power requirements, star topology which is not always adequate for distributed applications, and existing protocol stacks (mainly IP) are not optimized for real-time operation [Decotignie, 2005; Felser, 2005]. On the other hand, if the applications have predictable and bounded number of requests, the behavior of Ethernet can be "nearly" real-time (see various industrial solutions in IEC 61784-2). Several ways to achieve low probability of the delayed data delivery and packet loses can be classified as follows [Almeida, 2005]:

- To keep reasonably low number of accesses compared to the high-bandwidth performance (NDDS, ORTE - open source implementation of real-time Publish Subscribe developed in IST project OCERA [Dolejs *et al.*, 2004]).
- To avoid bursts.
- To modify the back-off and retry mechanism.
- To control transmission instants (e.g. token-passing (RETHER, RT-EP), master-slave (FTT-Ethernet, Ethernet-Powerlink), TDMA (PROFINET)).
- To avoid collisions by network micro-segmentation using switches (IEEE 802.1D).

5.4 Wireless networks

A wireless network is a flexible data communication technology, which uses wireless media such as radio frequency technology to transmit and receive data over the air (i.e. as their physical layer), minimizing the need for wired connections. Varying delay and packet loss are much more pronounced in wireless networks than in wired networks due to limited spectrum, limited power, time varying channel gains, and interference. Consequently, the wireless networks are not well suited for real-time communications. In spite of it, they are very attractive for control applications while offering the following productivity and convenience over traditional wired networks:

- Mobility: the mobile user can travel, without getting disconnected, while supporting productivity and service opportunities not possible with wired networks.

- Installation speed and simplicity: installing a wireless system can be fast and easy and can eliminate the need to pull cable through walls and ceilings.
- Reach of the network: the network can be extended to places which can not be wired (e.g. explosive areas in process control).
- More flexibility: wireless networks offer more flexibility and adapt easily to changes in the configuration of the network.
- Scalability: wireless systems can be configured in a variety of topologies to meet the needs of specific applications and installations. Configurations can be easily changed and range from mesh networks suitable for a small number of users to large infrastructure networks that enable roaming over a broad area.

Typically, we deal with two main categories of wireless networks:

- Infrastructure-based wireless networks (star topology) where the packets transmitted by the mobile node are received by the base station (or Access Point - AP) connected together with other base stations by a wired backbone network (pre deployed infrastructure). Mobile nodes communicate with each other through base stations.
- Ad-hoc wireless networks (mesh topology) do not have any fixed communication infrastructure (network topology) or central administration. In this decentralized approach all network activity including discovering the topology and delivering messages must be executed by the nodes themselves. Due to the limited radio propagation range of wireless devices, routes often traverse several intermediate nodes on the way from sender to receiver i.e. multi hop communications. Since the network topology is dynamic, data routes are subject to frequent disconnections. Ad-hoc networks are autonomous, self-organizing, self-configuring, multi-hop wireless networks.

In the wireless systems, reliability and timing requirements are significantly more difficult to meet, due to the adverse properties of the wireless radio channels, which often exhibit a substantial and time varying amount of transmission errors. A transmitted waveform experiences several distortions as path loss, attenuation, reflection, diffraction, scattering or interference. Attenuation is responsible for the phenomena called the hidden node and exposed node [Willig *et al.*, 2005] that jeopardize transmission control based on carrier sensing (CSMA). CSMA relies on a station ability to transmit and receive simultaneously on the same channel, witch is impossible with half duplex wireless transceivers. Two approaches to solve these problems are the busy tone solutions and the RTS/CTS protocol [Willig, 2005].

Instead of seeking absolute guarantees it is appropriate to formulate another criteria as the percentage of safety critical messages which can be transmitted reliably within a pre specified time bound (e.g. > 99,x %). Of course, this stochastic real time approach limits the application areas of wireless field-bus systems.

In wireless media the reliability can be ensured using different mechanisms such as retransmissions, packet duplications, multi-path modulation schemes (e.g. OFMD), spread spectrum modulations (e.g. DSSS, FHSS) or error correcting codes (e.g. FEC [Lin and Costello, 2005]). Important characteristics of wireless (radio) channel and transceiver that make wireless system different from wired field bus systems:

- path loss: The signal strength of a radio signal decreases with the distance between a transmitter and a receiver.
- limited physical security: Wireless networks are generally more prone to physical security threats than are wired networks. The increased possibility of eavesdropping, spoofing, and denial-of-service attacks should be carefully considered.
- energy supply and low power operation: Some or all of the nodes may rely on batteries or other exhaustible means for their energy. For these nodes, the important system design criteria may be energy conservation and low power operation.
- half duplex operation of transceivers: Wireless transceivers are not able to transmit and receive simultaneously on the same channel. The disadvantage is the time loss experienced from explicit receiver transmit turnovers.
- physical layer overheads: To let receiver of packet acquire synchronization despite a noisy channel, most wireless systems use extra training sequences of well known symbols.
- channel errors: The transmitted waveform can be subject to reflection, diffraction or scattering and these channels errors can cause packets to miss their deadlines.

Generally speaking, ad-hoc networks are massively distributed sensing and control systems which interact in real-time with a physical environment. Currently we have very little analytic tools for predicting and controlling the temporal behavior of such systems. It seems that fundamentally new system models and new approaches for the schedulability analysis are needed.

It is the medium access control and the link layers which are exposed most to the error characteristics of wireless channels and which should do most of the work to improve the quality and reliability of the wireless channel. Specifically for real time communications the MAC layer is a key component; if the delays on the MAC layer are not bounded, the upper layers cannot compensate this.

In the master/slave method, a master polls all other stations (slaves). In order to guarantee short response times, there is a bound in the time left to a slave to answer a master request. With a wireless transceiver, there is an additional delay introduced by the switching time from reception to emission and vice versa. The timeout should hence be extended accordingly thus reducing the performances.

The central bus arbiter (BA) technique exhibits similar problem. To handle cyclic traffic, the BA broadcasts the identifier of a variable. The producer of the variable responds by

sending the value and the consumers just capture it. In wireless systems, not all nodes may be visible from others. Additionally, failure of a BA is detected by the absence of traffic during a given period. With wireless transmission, this may occur as a result of fading and interference.

In token based systems, problems come from the hidden node effect that may preclude token passing as well as the longer delay in responses due to the transceiver switching time.

Carrier sensing (CSMA) method is possible but it is not very reliable. Collision detection is nearly impossible and deterministic collision resolution as in CAN cannot be implemented. Note that some application layers of CAN exploit the broadcast nature of the medium and will suffer from the effects mentioned above.

In preconfigured TDMA schemes, each station knows when it has to transmit. In most of the cases, additional gap time should be left between transmissions to cope with the transceiver longer switching times.

Due to the general tendency towards standardization and the fact that cheap, commercial Small Office and Home (SOHO) wireless technologies are available, it seems logical to investigate these for their suitability in industrial deployment. Of particular interest for industrial environments are technologies that do not require any sort of frequency licensing (i.e. ISM band). These technologies include the wireless personal area network (WPAN) technologies such as IEEE 802.15.1/Bluetooth and IEEE 802.15.4/ZigBee as well as the wireless local area network (WLAN) technologies from the IEEE 802.11 family. Seamless connectivity of moving entities might require hand-over algorithms. While for IEEE 802.11 roaming is specified in IEEE 802.11f, it is not covered in the specifications of Bluetooth and ZigBee.

Chapter 6

Future Research Directions

6.1 High-Level Objective

The high level objective in the research on implementation-aware control is to develop theory and methods that allow *faster* and more *efficient* development of embedded and networked control systems that are *safer*, more *flexible*, more *predictable*, more *resource-efficient*, and have better *performance* than what is possible today.

The way to reach this objective is to increase the understanding of the interactions between control, computing, and communication. It is also important and increase the understanding of when to apply codesign approaches, and when it is better to design the system in such a way that the concerns of the control, computing, and communication are truly separated.

6.2 Challenges

Several challenges are facing the control and computing communities in order to be able to achieve the high-level objective. These includes the following:

- **Theory for resource-aware control.** A theory for resource-aware control needs to be developed that makes it possible to take limited computing and communication resources into account both at design time and at run-time. This includes development of a theory for temporal robustness that is comparable to existing robust control theory, which primarily is intended for parametric plant uncertainties. It also include development of codesign methods.
- **Model-based development.** Integration is needed between the model-based development that currently is standard practice in the control community and model-

based or model-driven engineering and development from a software point of view. In control model-based development normally implies automatic generation of algorithmic code from modeling and simulation tools like Matlab/Simulink. The algorithmic parts is, however, normally only a relatively small part of the overall software in an embedded product. The remaining parts of the software are typically developed using modeling tools based on UML and real-time extension of UML.

- **Code generation.** Associated with the above item is the need for better ways to automatically generate software from high-level abstract models for both hard real-time control and implementation-aware real-time control, taking limited computing and communication resources into account.

- **Autonomic Control Systems.** The need for self-maintenance, self-organization, and self-monitoring is perhaps even larger in control systems than what it is in mainframe computing, for which IBM in 2001 initiated the autonomic computing initiative. The ultimate aim is to create self-managing computer systems to overcome their rapidly growing complexity and to enable their further growth. In control systems the self-monitoring and self-maintenance not only should be applied to the hardware and software, but also to the plant under control and its associated instrumentation. In many respects the control and automation industry have already reached further in this than what is the case in the computing industry. One example is the Industrial IT concept from ABB.

- **New models of computation.** Several of the problems associated with implementation of embedded control stem from inadequate computing and scheduling models. In the cases when codesign is not an option, better methods of separating the concerns of control and computing are needed then what is offered by today's priority-based systems. Alternative approaches are needed that provide temporal and functional isolation between modules or components and where it is possible to reserve resources statically and/or dynamically. Computing models are also needed that allow us to view controllers as real-time components that can be composed and have both predictable real-time behaviour and control performance. In order to achieve this it is important to be able to combine statically scheduled input and output operations with dynamically scheduled code execution.

- **Theory development within event-based control.** Event-based control, where the sampling is performed when an event occur, e.g., an error threshold crossing, rather than driven by time, has many interesting implications. A problem with aperiodic or event-based systems and aperiodic control of this type, though, is the lack of theory. This is because the resulting system descriptions are both time-varying and non-linear and hence very difficult to analyze.

  However, there are several indications that event-based control can have substantial advantages. Event-based control of first-order stochastic systems was studied in [Åström and Bernhardsson, 1999]. It was shown that an event-based controller for an integrator plant disturbed by white noise requires, on average, only one fifth of the sampling rate of an ordinary, periodic controller to achieve the same

output variance. Similarly, in [Speranzon and Johansson, 2003] event-triggered vs. periodic communication between mobile robots in a pursuit-evasion game was studied. Monte Carlo simulations showed that the event-triggered approach only required on average half the bandwidth of the periodic case. If it would be possible to derive a theory for aperiodically sampled systems, this could potentially have dramatic consequences for real-time computing and networking.

- **Integration of multiple models and management and re-use of models** Tools for model-based control engineering today only support the handling of a limited number of aspects (or views). A complete development chain typically involves a multitude of tools and pieces of information that are loosely integrated. Examples of problems in this area include efforts trying to combine Simulink (or SCADE or something else) with UML for software, coordinating function specification tools with specific engineering analysis models/tools (e.g. for safety and reliability), and coordinating control engineering tools with architectural design tools. These problems have led to intensive research and industrial efforts, where some efforts try to develop all-encompassing languages to cover as many aspects as possible. Examples in this direction include the EAST-ADL, the AADL and the UML. The former two are specialized for vehicular embedded systems development and provide constructs and properties required for analysis of safety, reliability and timing, as well as explicit models of software components, see [Freund *et al.*, 2004; AADL, 2004]. While the UML was initially intended for general purpose software systems, work is being undertaken to provide more capabilities for non-functional aspects by defining UML profiles, see [Bouyssounouse and Sifakis, 2005]. Other OMG related efforts focus on establishing the Model Driven Architecture for which a key ingredient is research on the systematization of transformations between models.

  Yet other efforts in different ways strive for integration platforms that allow models and tool services to be combined thus still allowing existing tools and modeling languages to be used, e.g. the Workshop on Tool Integration in System Development held at the ESEC/FSE Conference in Helsinki, Finland in 2003. The need to describe multiple aspects is also strongly relevant for CBD and software systems in general.

  The model management problem becomes apparent with an increasing usage of MBD. Working with MBD tools has, in fact, many similarities to component based development (CBD). A designer will work by composing model components from libraries. Newly developed models or just components thereof can be stored there as components, for reuse. However, models appear in many versions and variants, are used by many persons, and associated with even more information such as parameters, verification status etc. Re-use of model components and models (component assemblies) faces a reuse problem not far from that in CBD. That is, a pure textual description and interface definitions are not sufficient. Additional contextual information is also required. Keeping track of all model information requires explicit consideration, and tool support for larger systems. Proper configuration management thus has to be in place. This is a rather new problem for MBD in embedded systems, however less so for the more mature area of mechanical engineering (for indications on this see [Crnkovic *et al.*, 2003] and [Bouyssounouse and

Sifakis, 2005]). A key challenge for MBD in embedded systems is to combine such best practices with the emerging results in embedded systems MBD.

6.3 Research Directions

In addition to the research motivated by the challenges above, the following research directions are important:

- **Temporal robustness.** Development of temporal robustness indices that also apply to sampling jitter and nonlinear systems are important. It would also be valuable with a single index that includes both robustness towards sampling jitter and robustness towards input output latency.
- **Numerical robustness.** Although hardware support for floating point calculations (FPUs) are becoming increasingly common also within embedded systems the use of fixed point calculations is still common is applications with hard resource constraints, e.g., in sensor network applications. The discretization caused by the reduced resolution may create problems for feedback control applications. Methods for robustness analysis of the numerical effects of fixed reduced-word length calculations are necessary- The same situation can be present in networked control loops with severely limited communication bandwidth, e.g., when only a few bits may be used to transfer control loop information.
- **Control-aware implementation techniques.** Development of computational models, scheduling models, middleware, and operating system primitives/services that are tailored to and flexible with respect to the demands of control applications.
- **Codesign tools.** Development of generic as well as domain-specific codesign tools with a special emphasis on synthesis and code generation. Environments and platforms for integration of several tools, possibly involving multiple domain models, into tool chains that match the design work-flow process. Support for multiple views or aspects, and support for configuration management.
- **Tradeoff characterization.** Identification of which tradeoffs and design parameters to apply in codesign, e.g. word length. Improved understanding of when codesign is motivated. Tradeoffs in wired and wireless networked control systems, e.g., latencies vs. noise levels and packet loss probability. Characterization of what can be achieved using incremental changes to an existing design and what requires a complete re-design.
- **Domain and application characterization.** Identify/chart the real-time (control) related requirements in different industrial domains (e.g., aerospace, automotive, process automation, cellular phone systems, building automation, medical systems) and for different application types. Improve the understanding of what is important, difficult, and expensive in different domains.

- **New paradigms for real-time computing.** Hardware and software solutions better suited for control than the real-time OS and microprocessors of today. Development of execution-time predictable hardware and system software. Support for synchronous I/O. Support for common notions of time, also for distributed systems. New scheduling and resource allocation methods that are not inherited from the desktop computing world, but rather developed with the needs of real-time control applications in mind to begin with. Reservation-based models rather than task-based models. Architectures and kernels better suited to control applications, possibly based on Simplex-type ideas, see [Sha, 1998; Seto *et al.*, 1998].

- **Real-time Ethernet**An important research and development direction in the real-time Ethernet area is the optimization of the protocol stack on a given operating system. The reason for this is that for high bit rates the delays in the protocol stacks become more important than the network-induced delays. Redundancy and fault tolerant concepts will certainly remain important research issue for distributed control applications.

- **Wireless networked control** In a wireless environment, network devices work in group to carry out a given task. Hence, global communications (e.g. multicast, scattering etc.) play an important role in wireless networks. From the protocol stack perspective promising research are attribute-based addressing and ID-less QoS-sensitive routing at the network layer, mobile endpoints and group-to-group real-time communication at the transport layer, and new network programming paradigms and new task/computation models in general. Another important goal is to provide a middleware which controls power/information-quality trade-offs in wireless networks. Wireless networks in general is an open wide area of control applications and novel research topics especially in the area of massively distributed control systems.

Chapter 7

Bibliography

AADL (2004): "Architecture analysis description language." SAE Standard AS-5506.

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

Albertos, P. (2002): "Phase-conditionnally stable systems." In *Asian Control Conference Shanghai*. Singapore.

Albertos, P. and A. Crespo (1999): "Real-time control of non-uniformly sampled data systems." *Control Engineering Practice*, **38:8**, pp. 1053–1059.

Albertos, P. and A. Crespo (2001): "Integrated design and implementation of digital controllers." In *EUROCAST*, pp. 385–392.

Albertos, P., A. Crespo, I. Ripoll, M. Vallésï£¡, and P. Balbastre (2000a): "Rt control scheduling to reduce control performance degrading." In *Proc. 39th IEEE Conference on Decision and Control*. Sydney, Australia.

Albertos, P., A. Crespo, I. Ripoll, M. Vallï£¡, and P. Balbastre (2000b): "RT control scheduling to reduce control performance degrading." In *Proc. 39th IEEE Conference on Decision and Control*. Sydney, Australia.

Albertos, P., A. Crespo, M. Vallés, and P. Balbastre (2005): "Embedded control systems: some issues and solutions." In *Proceedings of the 16th IFAC World Congress*, pp. 257–262. Prague.

Albertos, P. and M. Olivares (1999): "Time delay limitations in control implementation." In *European Control Conference*. Karslrue. (D).

Albertos, P., M. Salgado, and M. Olivares (2000): "Are delays in digital control implementation always bad?." In *Asian Control Conference Shanghai*.

Albertos, P., R. Sanchis, and A. Sala (1999): "Output prediction under scarce data operation control applications." *Automatica*, **35:8**, pp. 1671–1681.

Albertos, P., M. Vallés, and A. Valera (2003a): "Controller transfer under sampling rate dynamic changes." In *IFAC Workshop on Modelling and Analysis of Logic Controlled Dynamic Systems*. Irkutsk (Russia).

Albertos, P., M. Vallés, and A. Valera (2003b): "Controller updating under operational logic changes." In *European Control Conference*.

Almeida, L. (2005): "Real-time networks for embedded systems: a focus on operational flexibility." In *IFAC Summer School on Control, Computing and Communication*. Prague.

Almeida, L., E. Tovar, J. Fonseca, and F. Vasques (2002): "Schedulability analysis of real-time traffic in worldfip networks: an integrated approach." *IEEE Trans. On Industrial Electronics*, **49:5**, pp. 1165–1174.

Åström, K. J. and B. Bernhardsson (1999): "Comparison of periodic and event based sampling for first-order stochastic systems." In *Preprints 14th World Congress of IFAC*, vol. J, pp. 301–306.

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

Audsley, N., A. Burns, M. Richardson, K. Tindell, and A. Wellings (1993): "Applying new scheduling theory to static preemptive scheduling." *Software Engineering Journal*, **8:5**, pp. 285–292.

Balbastre, P., I. Ripoll, and A. Crespo (2000): "Control task delay reduction under static and dynamic scheduling policies." In *Proc. 7th International Conference on Real-Time Computing Systems and Applications*.

Baruah, S., G. Buttazzo, S. Gorinsky, and G. Lipari (1999a): "Scheduling periodic task systems to minimize output jitter." In *Proc. 6th International Conference on Real-Time Computing Systems and Applications*.

Baruah, S., D. Chen, S. Gorinsky, and A. Mok (1999b): "Generalized multiframe tasks." *Real-Time Systems*, **17:1**, pp. 5–22.

Baruah, S., D. Chen, and A. Mok (1997): "Jitter concerns in periodic task systems." In *Proc. 18th Real-Time Systems Symposium*.

Bernat, G., A. Burns, and A. Liamosi (2001): "Weakly-hard real-time systems." *IEEE Transactions on Computers*, **50:4**, pp. 308–321.

Bouyssounouse, B. and J. Sifakis, Eds. (2005): *Embedded Systems Design: The ARTIST Roadmap for Reasearch and Development*. Number 3436 in LNCS. Springer-Verlag.

Burns, A., K. Tindell, and A. J. Wellings (1994): "Fixed priority scheduling with deadlines prior to completion." In *Proc. 6th Euromicro Workshop on Real-Time Systems*, pp. 138–142.

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

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

Cervin, A., D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén (2003): "How does control timing affect performance?" *IEEE Control Systems Magazine*, **23:3**, pp. 16–30.

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 *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications*. Göteborg, Sweden. Best paper award.

Chung, J.-Y., J. Liu, and K.-J. Lin (1990): "Scheduling periodic jobs that allow imprecise results." *IEEE Transactions on Computers*, **39:9**.

Crespo, A., I. Ripoll, and P. Albertos (1999): "Reducing delays in RT control: The control action interval." In *Proc. 14th IFAC World Congress*, pp. 257–262.

Crnkovic, I., U. Asklund, and D. A. Persson (2003): *Implementing and Integrating product data management and software configuration management*. Artech House Publishers.

Cumplido, R., S. Jones, R. Goodall, and S. Bateman (2005): "A high-performance processor for embedded real-time control." *IEEE Trans. on CST*, **13:3**, pp. 485–492.

Davidson, C. (1973): *Random sampling and random delays in optimal control systems*. PhD thesis PhD Dissertation 21429, Department of Optimization and Systems Theory, Royal Institute of Technology, Sweden.

Decotignie, J.-D. (2005): "Ethernet,based real-time and industrial communications." *Proc. of the IEEE*, **93:6**, pp. 1102–1117.

Dolejs, O., P. Smolik, and Z. Hanzalek (2004): "On the ethernet use for real-time applications." In *5th IEEE International Workshop on Factory Communication Systems, WFCS*. Vienna.

EAST-EEA (2002): "East-eea embedded electronics architecture: Scenario 2005 and study report." Technical Report. ITEA.

Eker, J., P. Hagander, and K.-E. Årzén (2000): "A feedback scheduler for real-time control tasks." *Control Engineering Practice*, **8:12**, pp. 1369–1378.

Felser, M. (2005): "Real-time ethernet - industry prospective." *Proc. of the IEEE*, **93:6**, pp. 1118–1129.

Freund, U., O. Gurrieri, J. Küster, H. Lonn, M.-O. Migge, T. Wierczoch, and Weber (2004): "An architecture description language for developing automotive ECU-software." In *Proceedings of INCOSE 2004*.

Gerber, R. and S. Hong (1993): "Semantics-based compiler transformations for enhanced schedulability." In *Proc. 14th IEEE Real-Time Systems Symposium*, pp. 232–242.

Gerber, R. and S. Hong (1997): "Slicing real-time programs for enhanced schedulabilty." *ACM Transactions on Programming Languages and Systems*, **19:3**, pp. 525–555.

Gerber, R., S. Hong, and M. Saksena (1995): "Guaranteeing real-time requirements with resource-based calibration of periodic processes." *IEEE Trans on Software Engineering*, **21:7**.

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

Harbour, M. G., M. G., M. H. Klein, and J. P. Lehoczky (1994): "Timing analysis for fixed-priority scheduling of hard real-time systems." *IEEE Transactions on Software Engineering*, **20:1**, pp. 13–28.

Henriksson, D., A. Cervin, J. Åkesson, and K.-E. Årzén (2002a): "Feedback scheduling of model predictive controllers." In *Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium*. San Jose, CA.

Henriksson, D., A. Cervin, J. Åkesson, and K.-E. Årzén (2002b): "On dynamic real-time scheduling of model predictive controllers." In *Proceedings of the 41st IEEE Conference on Decision and Control*. Las Vegas, NV.

Henzinger, T. A., B. Horowitz, and C. M. Kirsch (2001): "Giotto: A time-triggered language for embedded programming." In *Proc. First International Workshop on Embedded Software*.

Hespanha, J. and S. Morse (2002): "Switching between stabilizing controllers." *Automatic*, **38**, pp. 1905–1917.

Kalman, R. E. and J. E. Bertram (1959): "A unified approach to the theory of sampling systems." *Journal of the Franklin Institute*, **267:5**, pp. 405–436.

Kao, C.-Y. and B. Lincoln (2004): "Simple stability criteria for systems with time-varying delays." *Automatica*, **40:8**, pp. 1429–1434.

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

Kopetz, H., R. Obermaisser, P. Peti, and N. Suri (2004): "From a federated to an integrated architecture for dependable real-time embedded systems." Technical Report. TU Vienna.

Krakora, J. and Z. Hanzalek (2004): "Timed automata approach for can verification." In *11th IFAC Symposium on Information Control Problems in Manufacturing, INCOM*. Salvador.

Lin, K. and A. Herkert (1996): "Jitter control in time-triggered systems." In *Proc. 29th Hawaii International Conference on System Sciences*.

Lin, S. and D. J. Costello (2005): *Error Control Coding*. Englewood Cliffs, New Jersey: Prentice-Hall. 2nd Ed.

Lincoln, B. and B. Bernhardsson (2000): "Efficient pruning of search trees in LQR control of switched linear systems." In *Proc. 39th IEEE Conference on Decision and Control*.

Lincoln, B. and A. 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**, pp. 40–61.

Liu, D., X. S. Hu, M. D. Lemmon, and Q. Ling (2003): "Firm real-time system scheduling based on a novel QoS constraint." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Liu, J., K.-J. Lin, and S. Natarajan (1987): "Scheduling real-time, periodic jobs using imprecise results." In *Proc. IEEE Real-Time System Symposium*, pp. 252–260.

Liu, J., W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung (1994): "Imprecise computations." *Proc. IEEE*, Jan, pp. 83–93.

Liu, X. and A. Goldsmith (2003): "Wireless communication tradeoffs in distributed control." In *IEEE Conference on Decision and Control*, vol. 1, pp. 688–694. Maui, Hawaii.

Marzario, L., G. Lipari, P. Balbastre, and A. Crespo (2004): "Iris: A new reclaiming algorithm for server-based real-time systems." In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 211–218.

Nilsson, J. (1998): *Real-Time Control Systems with Delays*. PhD thesis ISRN LUTFD2/TFRT--1049--SE, Department of Automatic Control, Lund Institute of Technology, Sweden.

Nilsson, J., B. Bernhardsson, and B. Wittenmark (1998): "Some topics in real-time control." In *Proc. 17th American Control Conference*, pp. 2386–2390. Philadelphia, Pennsylvania.

OCERA (2002): "Ocera: Open components for real-time embedded applications." Technical Report. IST 35102 European Project. European Commission.

Palopoli, L., C. Pinello, A. Sangiovanni-Vincentelli, L. El-Ghaoui, and A. Bicchi (2002): "Synthesis of robust control systems under resource constraints." In *Proc. Workshop on Hybrid Systems: Computation and Control*.

Real, J. and A. Crespo (2004): "Mode change protocols for real-time systems: A survey and a new proposal." *Real-Time Systems*, **26:2**, pp. 161–197.

Rehbinder, H. and M. Sanfridson (2000): "Integration of off-line scheduling and optimal control." In *Proc. 12th Euromicro Conference on Real-Time Systems*.

Richard, J.-P. (2003): "Time-delay systems: an overview of some recent advances and open problems." *Automatica*, **39**, pp. 1667–1694.

Ryu, M. and S. Hong (1998): "Toward automatic synthesis of schedulable real-time controllers." *Integrated Computer-Aided Engineering*, **5:3**, pp. 261–277.

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.

Sanchis, R. and P. Albertos (2002): "Recursive identification under scarce measurements. convergence analysis." *Automatica*, **38:8**, pp. 535–544.

Seto, D., B. Krogh, L. Sha, and A. Chutinan (1998): "Dynamic control system upgrade using the Simplex architecture." *IEEE Control Systems*, **18:4**.

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*.

Sha, L. (1998): "Dependable system upgrade." In *Proc. IEEE Real-Time Systems Symposium (RTSS 98)*, pp. 440–449. IEEE CS Press.

Speranzon, A. and K. Johansson (2003): "Distributed pursuit-evasion game: Evaluation of some communication schemes." In *The Second Annual Symposium on Autonomous Intelligent Networks and Systems*.

Tindell, K., A. Burns, and A. J. Wellings (1995): "Calculating controller area network (can) message response times." *Control Engineering Practice*, **3:8**, pp. 1163–1169.

Tovar, E. and F. Vasques (1999): "Real-time fieldbus communications using profibus networks." *IEEE Transactions on Industrial Electronics*, **46:6**, pp. 1241–1251.

Törngren, M. and O. Larses (2004): "Characterization of model-based development of embedded control systems from a mechatronic perspective: drivers, processes, technology, and their maturity." In *Proceedings of the Summer School Model Driven Engineering for Embedded Systems*.

Willig, A. (2005): "Wireless lan technology for the factory floor." In *the Industrial Information Technology Handbook*, pp. 1–15.

Willig, A., K. Matheus, and A. Wolisz (2005): "Wireless technology in industrial networks." *Proceedings. of the IEEE*, **93:6**, pp. 1130–1151.

Xiao, J.-J., S. Cui, Z.-Q. Luo, and A. Goldsmith (2004): "Joint estimation in sensor networks under energy constraints." In *IEEE Conference on Sensor and Ad Hoc Communications and Networks*, pp. 264 – 271. Santa Clara, CA.

Xiao, L., M. Johansson, H. Hindi, S. Boyd, and A. Goldsmith (2003): "Joint optimization of communication rates and linear systems." *IEEE Transactions on Automatic Control*, **48:1**, pp. 148–153.

Zhao, Q. C. and D. Z. Zheng (1999): "Stable and real-time scheduling of a class of hybrid dynamic systems." *Journal of Discrete Event Dynamical Systems*, **9:1**, pp. 45–64.