ISSN 0280–5316
ISRN LUTFD2/TFRT--7586--SE

Integrated Control and Scheduling

Karl-Erik Årzén
Bo Bernhardsson
Johan Eker
Anton Cervin
Patrik Persson
Klas Nilsson
Lui Sha

Department of Automatic Control
Lund Institute of Technology
August 1999

Integrated Control and Scheduling

Karl-Erik Årzén, Bo Bernhardsson
Johan Eker, Anton Cervin

Dept of Automatic Control
Lund Institute of Technology
Box 118, 221 00 Lund

Klas Nilsson, Patrik Persson

Dept of Computer Science
Lund Institute of Technology
Box 118, 221 00 Lund

Lui Sha

Dept of Computer Science
University of Illinois at Urbana-Champaign
1304 West Springfield Avenue
Urbana, Illinois 61801

Abstract: The report gives a state-of-the-art survey of the field of integrated control and scheduling. Subtopics discussed are implementation and scheduling of periodic control loops, scheduling under overload, control and scheduling co-design, dynamic task adaptation, feedback scheduling, and scheduling of imprecise calculations. The report also presents the background, motivation, and research topics in the ARTES[1] project "Integrated Control and Scheduling".

1. Introduction

Real-Time Systems is an inter-disciplinary research area that includes aspects of Control Engineering and Computer Science. It is an area that is of vital importance to all control engineers. Today almost all controllers

[1] A Swedish research network and graduate school on real-time systems.

are implemented on digital form with a computer. A successful implementation of a computer-controlled system requires a good understanding of both control theory and real-time systems.

Many real-time control systems are embedded systems where the computer is a component in a larger engineering system. These systems are often implemented on small microprocessors using a real-time programming language such as Ada or Modula-2 with a real-time kernel or run-time system, or using a standard sequential programming language such as C or C++ together with a real-time operating system (RTOS). Examples of such systems are found in mechatronic systems such as industrial robots, in aerospace applications such as airplanes and satellites, in vehicular systems such as cars, and in consumer electronic products. Control applications in the process and manufacturing industry are often implemented with distributed control systems (DCS) or programmable logic controllers (PLC). These systems are often programmed using special programming languages such as sequential function charts (SFC), function block languages, or relay or ladder diagram languages. Distributed control systems and programmable logic controllers share many of the characteristics of embedded systems. The implementation of the systems is often based on real-time operating systems.

Traditionally, when implementing computer control systems, the control community has assumed that the computer platform used can provide the deterministic, equi-distant sampling that is the basis of sampled computer control theory. However, many of the computing platforms that are commonly used to implement control systems are not able to give any deterministic guarantees. This is especially the case when commercial off-the-shelf (COTS) operating systems such as, e.g., Windows NT are used. These systems are, typically, designed to achieve good average performance rather than guaranteed worst-case performance. They often introduce significant non-determinism in task scheduling. For computation intensive high-end applications, the large variability in execution time caused by modern hardware architecture becomes visible.

On the other hand, the real-time scheduling community generally assumes that all control algorithms should be modeled as tasks that:

- are periodic with a set of *fixed periods*,
- have *hard deadlines*, and
- have known *worst-case execution times* (WCETs).

This simple model has permitted the control community to focus on its own problem domain without worrying about how scheduling is being done, and it has released the scheduling community from the need to understand what impact scheduling delays have on the stability and performance of the plant under control. From a historical perspective, the separated development of control and scheduling theories for computer-based control systems has produced many useful results and served its purpose.

However, upon closer inspection it is quite clear that neither of the three above assumptions need necessarily be true. Many control algorithms are not periodic, or they may switch between a number of different fixed sampling periods. Control algorithm deadlines are not always hard. On the contrary, many controllers are quite robust towards variations in sampling period and response time. It is also in many cases possible to compensate

on-line for the variations by, e.g., recomputing the controller parameters. Obtaining an accurate value for the WCET is a difficult problem in the real-time scheduling area. It is not likely that this problem is significantly simpler for control algorithms. It is also possible to consider control systems that are able to do a tradeoff between the available computation time, i.e., how long time the controller may spend calculating the new control signal, and the control loop performance.

The objective of the ARTES project "Integrated Control and Scheduling" is to go beyond the simple "fixed sample period and known WCET" model and to develop theory that is able to address dynamic interaction between control and scheduling. The optimality of computer control is subject to the limitations of available computing resources, especially in advanced applications where we want to control fast plant dynamics and to use sophisticated state estimation and control algorithms. On the other hand, the true objective of real-time scheduling for control is to allocate limited computing resources in such a way that the state estimation and control algorithms can ensure the system's stability and optimize the system's performance. The computing resources could include CPU time and communication bandwidth. In this overview we have focused on CPU time. However, most of the issues brought up also apply to distributed system scheduling of communication bandwidth.

The approach taken in the project is based on using dynamic feedback from the scheduler to the controllers and from the controllers to the scheduler. The idea of feedback has been used informally for a long time in scheduling algorithms for applications where the dynamics of the computation workload cannot be characterized accurately. For instance, the VMS operating system uses multi-level feedback queues to improve system throughput, and Internet protocols use feedback to help solve the congestion problems. Recently, under the title of quality of service (QoS), the idea of feedback has also been exploited in multi-media scheduling R&D.

Given this, one might expect that the use of feedback in the scheduling of feedback control systems would have been naturally an active area. On the contrary, the scheduling research of feedback control systems are dominated by open loop analytic scheduling methods such as rate or deadline based algorithms. This is not an accident but rather the consequence of some serious theoretical challenges that require the close cooperation between control and scheduling communities.

However, there are still systems in which true hard deadlines have to be met. For such cases, we have ideas about how to estimate the WCET in a practical way. In the case we want to accurately estimate the WCET, it would be ideal to obtain execution times for each piece of code incrementally during coding. To achieve this, our idea is to utilize the so called Mjölner approach, [Knudsen *et al.*, 1994], and to use attribute grammars for incremental semantic analysis of the program. The WCET of each piece, statement, or block of code would then be handled by another aspect of our object-oriented semantic language (OOSL).

Still, we want to allow language constructs that principally have unknown WCET, like recursion and loops depending on sensor inputs. Since, in our system, code generation is based on grammar attributes, it is straight forward to generate additional exception handling code and enforce the programmer to declare timing bounds and to define how to cope with unexpected delays. The combination of control and scheduling as described

above then makes such exception handling feasible.

Aim of the report

The aim of this report is to give the background and motivation for the research project, provide a survey of existing work in the area, and attempt to outline what the important research questions that need answers are. The area that we are entering is quite large, still relatively little work has been performed so far especially by the control community. This leads us to believe that there is room for a lot of work, of both practical and theoretical nature.

Outline of the report

The vision of a flexible environment with on-line interaction between control algorithms and on-line scheduling is outlined in Section 2. An overview of hard real-time scheduling is given in Section 3. A subproblem of integrated control and scheduling is how to correctly implement and schedule periodic controller tasks. This problem is discussed in Section 4 together with different approaches to jitter handling. One of the key issues in the project is the relaxation of the requirement on a known worst-case task execution time. The simplest approach to this is to simply treat the actual execution times that are longer than the worst case bound as overload conditions. Scheduling in the presence of overload is discussed in Section 5. A prerequisite for an on-line integration of control and scheduling theory is that we are able to make an integrated off-line design of control algorithms and scheduling algorithms. This area is surveyed in Section 6. Dynamic task adaptation is the key element of this project. This subject is discussed is Section 7. Issues discussed include period skipping, quality of service resource allocation schemes, task attribute adjustment schemes, statistical scheduling, scheduling of imprecise calculations, mode-change protocols, and on-line system upgrades. In Section 8 it is shown how the different flexible scheduling models proposed in earlier sections match what control theory and control engineering offer. Section 9 describes the approach to WCET analysis taken in the project. Finally, in Section 10, some possible research directions are outlined.

2. The Vision

Our work is based upon a vision of a dynamic, flexible, and interactive integrated control and scheduling environment with the following features:

- The control design methodology should take the availability of computing resources into account during the controller design.
- The requirement of known worst-case execution times should be relaxed. Instead the system should be able to guarantee stability and a certain level of control performance based only on knowledge of nominal execution times.
- The system should be able to adapt task parameters in overload situations in such a way that stability and an acceptable level of control performance are maintained.

- The system should be able to dynamically trade-off control performance and computing resource utilization.
- The system should support on-line information exchange between the on-line scheduler and the control tasks. The information could for example consist of mode change requests from the control tasks, execution time allotments from the on-line scheduler, etc.
- The system should be able to measure the actual execution time spent by the different tasks, and take appropriate actions in case of overruns.
- The required execution time analysis should be made part of an interactive tool for automatic code generation.

In order to make this possible, a lot of information needs to be provided. For example, the control tasks must be able to provide information of the following kind to the on-line scheduler.

- The desired period of the control task together with a period range for which the controller can guarantee acceptable control performance. Alternatively this information can be stated as a cost function. The period information can be static or dynamic. Dynamic period constraints can, e.g., be useful in order to handle transients.
- The nominal execution time of the control task. This can possibly be combined with estimates of the minimum and maximum execution times. Alternatively, the control task can specify a desired execution time together with information about which variations of the execution time that the control task can handle while maintaining satisfactory control. In the same way as for the period this may also be stated as a cost functions. The information may be static or dynamic.
- The desired deadline for the control task. Alternatively this information can be provided indirectly by instead providing information about the desired or acceptable computational delay for the control task.

3. Real-Time Scheduling

It is very difficult to ensure hard deadlines by *ad hoc* methods. Recently, theoretical scheduling results have been derived that make it possible to *a priori* prove that a set of tasks meet their deadlines. Scheduling theory is studied in two research communities: operations research and computer science. Within operations research the problems studied are typically job shop or flow shop problems. In these problems the scheduling typically concerns jobs, orders, batches, or projects. The resources involved could be machines, factory cells, unit processes, people, etc. Within computer science the scheduling problem instead concerns the scheduling of tasks on a uni- or multiprocessor environment and the scheduling of communication bandwidth in distributed systems. Real-time scheduling has been a very fertile area of research during the last few decades.

In hard real-time systems it is crucial that the timing requirements always are met. Hence, it is necessary to perform an off-line analysis

that guarantees that there are no cases in which deadlines are missed. In scheduling theory we assume that we have events that occur and require computations. Associated with an event is a task that executes a piece of code in response to the event. The events could be periodic, sporadic, or aperiodic. A sporadic event is non-periodic but has a maximum inter-arrival frequency. An aperiodic event has an unbounded arrival frequency and could have many active associated tasks. Each event has a required computation time, in the sequel denoted C. This is the worst-case CPU time it takes to execute the piece of code associated with the event. Obtaining this time can in practice be quite difficult, as will be discussed in Section 9. Each event also has an associated deadline, denoted D. This is an upper bound on the allowed time taken to execute the piece of code associated with the event.

In this report we will primarily consider scheduling of CPU time for periodic tasks. Two main alternatives exist: *static cyclic executive scheduling* and *priority-based scheduling*. Static cyclic executive scheduling is an off-line approach that uses optimization-based algorithms to generate an execution table or calendar [Locke, 1992]. The execution table contains a table of the order in which the different tasks should execute and for how long they should execute. The run-time part of the scheduling is extremely simple. The drawback that makes cyclic executive scheduling unsuitable for our purposes is its static nature. It does not support on-line admission of new tasks, and dynamic modifications of task parameters. Hence, in this report we will focus on dynamic approaches to scheduling.

In 1973, Liu and Layland proposed in their seminal paper [Liu and Layland, 1973] two optimal priority-based scheduling algorithms, *rate-monotonic (RM) scheduling* and *earliest deadline first scheduling* (EDF). The EDF scheduling method is based on the principle that it is the task with the shortest remaining time to its deadline that should run. The approach is dynamic in the sense that the decision of which task to run is made at run-time. The deadline can also be viewed as a dynamic priority, in contrast to the RM case where the priority is fixed. The latter is the reason why rate-monotonic scheduling also is referred to as fixed priority scheduling.

Formal analysis methods are available for EDF scheduling. In the simplest case the following assumptions are made:

- only periodic tasks exist,
- each task i has a period T_i,
- each task has a worst case execution time C_i,
- each task has a deadline D_i,
- the deadline for each task is equal to the task period ($D_i = T_i$),
- no interprocess communication, and
- an “ideal” real-time kernel (context switching and clock interrupt handling takes zero time).

With these assumptions the following necessary and sufficient condition holds:

THEOREM 3.1—EDF SCHEDULING
If the utilization U of the system is not more than 100% then all deadlines will be met.

$$U = \sum_{i=1}^{i=n} \frac{C_i}{T_i} \leq 1$$

□

The utilization U determines the CPU load. The main advantage with EDF scheduling is that the processor can be fully utilized and still all deadlines can be met. More complex analysis exists that loosens some of the assumptions above.

Rate monotonic (RM) scheduling is a scheme for assigning priorities to tasks that guarantees that timing requirements are met when preemptive fixed priority scheduling is used. The scheme is based on the simple policy that priorities are set monotonically with task rate, i.e., a task with a shorter period is assigned a higher priority.

With essentially the same assumptions as in the EDF case, a sufficient schedulability condition for RM scheduling was derived in [Liu and Layland, 1973].

THEOREM 3.2—RM SCHEDULING
For a system with n tasks, all tasks will meet their deadlines if the total utilization of the system is below a certain bound.

$$\sum_{i=1}^{i=n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

□

As $n \to \infty$, the utilization bound $\to 0.693$. This has led to the simple rule-of-thumb that says that

> "If the CPU utilization is less than 69%, then all deadlines are met".

Since 1973 the analysis has evolved and many of the restrictive assumptions have been relaxed [Audsley *et al.*, 1995; Sha *et al.*, 1994]. In 1986 a sufficient and necessary condition was derived [Joseph and Pandya, 1986]. The condition is based on the notion of worst-case response time, R_i, for a task i, i.e., the maximum time it can take to execute the task. The response time of a task is computed by the recursive equation

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{1}$$

where $hp(i)$ is the set of tasks with higher priority than task i. The task set is schedulable if $R_i \leq D_i$ for all tasks i. The model also allows deadlines that are shorter than the period ($D_i < T_i$).

The rate-monotonic priority scheme is not very good when $D_i \ll T_i$. An infrequent but urgent task will be given a very low priority. In this case the *deadline-monotonic priority scheme* is better suited. Here it is the

task deadline that decides the priority rather than the period. A task with a short deadline gets high priority. This policy has been proved optimal when $D \leq T$ in the sense that if the system is unschedulable with the deadline-monotonic priority ordering, then it is unschedulable also with *all* other orderings [Leung and Whitehead, 1982]. Equation 1 holds also for the deadline-monotonic case.

During the last decade the rate and deadline monotonic analysis have been extended in various directions [Klein *et al.*, 1993]. It has been extended to cover situations where we have processes that communicate using, e.g., a monitor or using shared data protected by a mutual exclusion semaphore. To do this it is necessary to have an upper bound on how long a high-priority process may be blocked by a low-priority process due to interprocess communication. The priority ceiling protocol for mutual exclusion synchronization gives such a worst case bound [Sha *et al.*, 1990]. The analysis has also been extended to cover release jitter, i.e., the difference between the earliest and latest release of a task relative to the invocation of the task (the arrival of the event associated with the task), nonzero context switching times, and clock interrupts.

3.1 Alternative scheduling models

Several suggestions have also been made for alternative scheduling models. The *multi-frame model* [Mok and Chen, 1997] is a generalization of the periodic task model in which the execution times for periodic tasks are allowed to vary from job to job according to a regular pattern. Hence, the execution time is given by a finite list of numbers from which the execution times of successive jobs are obtained by repeating this list. Sufficient and necessary schedulability conditions for the multi-frame model are presented in [Baruah *et al.*, 1999]. In the generalized multi-frame model the task deadlines are also allowed to change according to a known sequential pattern [Baruah *et al.*, 1999] The *recurrent task model* [Baruah, 1998a], is a further extension that permits the modeling of certain forms of conditional code (“if-then-else” and “case”) which can be represented by task graphs that are directed acyclic graphs.

In the recurrent model each task is divided into subtasks. This approach has also been adopted in other contexts. In [Gonzalez Härbour *et al.*, 1994] each periodic task is divided into serially executed subtasks, each characterized by an execution time, a priority, and a deadline. The method proposed can be used to analyze the schedulability of any task set on a uni-processor whose priority structure can be modeled as serially executed subtasks.

The periodic task model is based on the assumption that all tasks may arrive simultaneously. If the task set is schedulable for this worst case it will be schedulable also for all other cases. In many cases this assumption is unnecessarily restrictive. Tasks may have precedence constraints that make it impossible for them to arrive at the same time. Alternatively, for independent tasks it is sometimes possible to shift them in time, i.e., introduce task offsets, to avoid the simultaneous arrival. If simultaneous arrivals can be avoided, the schedulability of the task set increases [Audsley *et al.*, 1993b]. Schedulability analysis, both for the case of static and dynamic task offsets, is presented in [Gutierrez and Harbour, 1998].

3.2 Aperiodic scheduling

Scheduling of soft aperiodic tasks in combination with hard periodic tasks is a large area where a lot of research is performed. The simplest approach is to use periodic tasks to poll aperiodic events. This may, however, increase processor utilization unnecessarily. Another approach is to use a special server for aperiodic events [Lehoczky *et al.*, 1987]. The main idea of the server approach is to have a special task, the server (usually at high priority), for scheduling the pending aperiodic work. The server has time tickets that can be used to schedule and execute the aperiodic tasks. If there is aperiodic work pending and the server has unused tickets, the aperiodic tasks execute until they finish or the available tickets are exhausted. Several servers have been proposed. The *priority-exchange server* and the *deferrable server* were proposed in [Lehoczky *et al.*, 1987]. The *sporadic server* was introduced in [Sprunt *et al.*, 1989]. The main difference between the servers concerns the way the capacity of the server is replenished and the maximum capacity of the server. In [Bernat and Burns, 1999] exact schedulability tests are presented for the sporadic and the deferred servers. It is also claimed that the deferred server is superior, since it has the same performance as the sporadic server and is easier to implement. The idea behind the *slack stealer* proposed in [Lehoczky and Ramos-Thuel, 1992] is to steal all the possible processing time from the periodic tasks, without causing their deadlines to be missed. The approach has a high overhead, but provides an optimal lower bound on the response times of aperiodic tasks. The method has also been extended to cover hard aperiodic tasks.

The idea of stealing time from hard periodic tasks is also used in *dual priority scheduling* [Davis and Wellings, 1995]. The priority range is divided into three bands: upper, middle, and lower. Hard tasks are assigned two priorities, one in the upper range and one in the lower range. At runtime, other tasks, e.g., soft aperiodic tasks, are assigned priorities in the middle band. The hard tasks are assigned their lower priorities upon release. A fixed time interval after their release they are promoted to their upper priority. During the initial phase of a task, the soft tasks will have higher priority, and thus execute. The net effect of the approach is that the execution of the hard tasks is shifted in such a way that the end of the task execution is always close to the response time of the task.

The above servers have been developed for the fixed-priority case. Similar techniques also exist for the dynamic-priority case (EDF). In [Spuri and Buttazzo, 1996] a comparative study of five different methods of handling soft aperiodic tasks in a dynamic priority setting is made. The methods compared include a dynamic version of the priority-exchange server, a dynamic version of the sporadic server, and the total-bandwidth server.

4. Implementation and scheduling of periodic control loops

A control loop consists of three main parts: data collection, algorithm computation, and output transmission, as shown in Fig. 1. In the simplest case the data collection and output transmission consist of calls to an external I/O interface, e.g. AD and DA converters or a field-bus interface. In a more

complex setting the input data may be received from other computational elements, e.g., noise filters, and the output signal may be sent to other computational elements, e.g., other control loops in the case of set-point control.

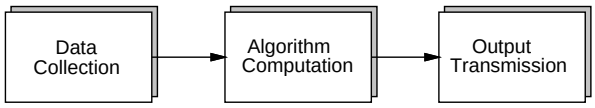

Figure 1 The three main parts of a control loop.

In most cases the control is executed periodically with a constant period, or sampling interval, T (often denoted h), that is determined by the dynamics of the process that is controlled and the requirements on the closed loop performance. This assumption will also be made in this section. Further on in the report we will relax the requirement on constant sampling intervals.

There are two main ways to implement control loops using tasks in a real-time operating system. In the first alternative, the *single-task approach*, the control loop is implemented as a single periodic task. The different computation steps are implemented as sequential code within the task. In the second alternative, the *multiple-task approach*, each subpart of the control loop is implemented as a separate task, e.g., the control loop in Fig. 1 would be implemented as three separate periodic tasks. The multiple-task approach originates from the Real-Time Structured Analysis and Design Method (RTSAD), [Ward and Mellor, 1985], a software engineering methodology where each data transformation is mapped onto a separate task. A drawback with the approach is that it leads to a large number of concurrent tasks with a potentially large overhead for synchronization, communication and context switches. An advantage with the approach is that it can handle multi-processor and distributed systems in a straightforward way.

The single-task approach is instead based on design methodologies that emphasize task structuring criteria. Data transformations are grouped with other data transformations into tasks, based on the temporal sequence in which the transformations are executed. Examples of these methodologies are DARTS [Gomaa, 1984] and the entity-life model [Sanden, 1994]. The advantages with the approach are: fewer tasks, less demand on synchronization and inter-process communication, and smaller context switch overhead. The approach is also the one that is most natural for control engineers, especially when implementing small embedded control systems on uni-processor platforms. The approach is also similar to the computation model used in the programmable logic controllers (PLCs) that are commonly used for implementing industrial control systems. For these reasons we will mainly focus on this approach in our work. However, it is important to have in mind that the single-task approach as described here only concerns the implementation of the controller. For schedulability analysis purposes it may be advantageous to treat the task as if it consists of multiple subtasks. This will be discussed further later on in this section.

In certain situations it can be difficult to chose which method to use. This is particularly the case when the subparts of the control loop execute at different frequencies. A common example of this is when anti-aliasing fil-

tering is implemented with analog low-pass filters in combination with discrete low-pass filters. As soon as analog signals are sampled, anti-aliasing filtering is preferable to avoid aliasing of frequencies above the Nyquist frequency. The filter has to be analog and designed in accordance with the sampling frequency. This causes problems if we want to be able to increase the sampling rate programmatically. An alternative approach is to use a short, fixed sampling period and a fixed analog filter designed for the fast sampling period together with a flexible discrete low-pass filter that executes at the same frequency as the sampler. The situation is shown in Fig. 2 for the case of PID control and where the data collection and output transmission consist of A/D and D/A conversions. The system can either

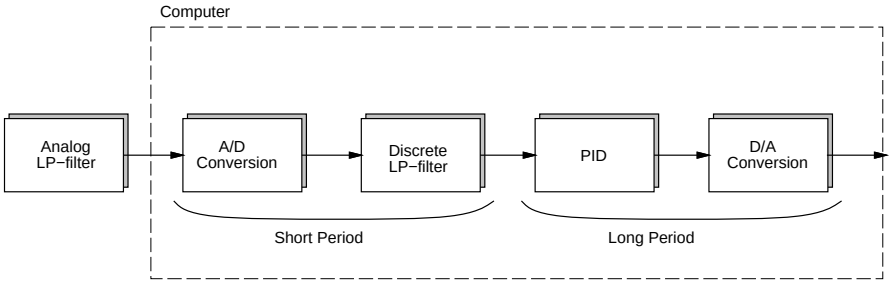

Figure 2 Combination of analog and discrete anti-aliasing filtering.

be implemented as two tasks: one with short sampling period and one with long sampling period, or as a single task that runs at the short sampling period, but only executes the PID and D/A parts at a multiple of this sampling period. The second case can be modeled by the multi-frame task model.

4.1 Loop timing constraints

The basic control loop has two timing constraints. The first is the period which should be constant, i.e., without jitter. The second constraint involves the *computational delay* from the input data collection to the output transmission. This is also known as the *control delay* or the *input-output latency*. From a control point of view this delay has similar effects as a pure time delay on the input of the controlled process. An overview of control loop timing constraints is given in [Törngren, 1998].

Four approaches are possible for the control delay. The simplest approach is to implement the system in such a way that the delay is minimized and then ignore it in the control design. The second approach is to try to ensure that the delay is constant, i.e., jitter free, and take this delay into account in the controller design. One way of doing this is to wait with the output transmission until the beginning of the next sample. In this way the computational delay becomes close to the sampling period. If the controller is designed with discrete (sampled) control theory it is especially easy to compensate for this delay. However it is also relatively easy to compensate for delays that are shorter than the sampling period. The third approach is to explicitly design the controller to be robust against jitter in the computational delay, i.e., the delay is treated as a parametric uncertainty. This approach is, however, substantially more complex than the first two. The fourth approach, finally, is based on the idea that the control algorithm in certain situations can actively, every sample, compensate

for the computational delay or parts of it. This approach has been used to compensate for the computational delays obtained when a control loop is closed over a communication network [Nilsson, 1998].

4.2 Single-task implementations

In the single-task approach a control loop is represented as a task with the following loop structure:

```
LOOP
  Await clock interrupt;
  Data collection;
  Algorithm calculation;
  Output transmission;
END;
```

A task with the above structure can be implemented in different ways depending on which timing primitives that are available in the real-time kernel. Some implementations are better than others in the sense that the period jitter and control delay jitter become smaller. A good implementation is the following:

```
CurrentTime(t);
LOOP
  Data collection;
  Algorithm calculation;
  Output transmission;
  t := t + h;
  WaitUntil(t);
END;
```

Here `t` is a timer variable, `CurrentTime` returns the current time, h is the period, and `WaitUntil (DelayUntil)` suspends the calling task until a certain future time. An alternative approach is to require that the real-time kernel provides special support for periodic processes. The application programmer's interface (API) for this could, e.g., consist of a procedure whereby the user registers a periodic task in the kernel with the task period and the task body procedure as arguments. However, this is normally not supported by commercial real-time operating systems.

4.3 Jitter

Using the single-task approach it is relatively straightforward to map the control loop timing constraints into task parameters. The sampling time of the control loop corresponds to the period of the corresponding task. However, the requirements on computational delay and jitter are more difficult to handle. The simplest approach is to transmit the control output at the end of the task. This will imply that the computational delay is always smaller then the task deadline. The problem with this is that the jitter may become considerable. The execution of two successive jobs (instances) of the task will not be separated exactly by the period T_i. For example, the upper bound, J_i, on the relative jitter in the start time of two successive jobs when $T_i = D_i$ is given as

$$J_i = 2(1 - \frac{C_i}{T_i})$$

The upper bound can reach a value of 200% when $C_i \ll T_i$. Similar expressions can be obtained for the end time jitter and the computational delay jitter. The maximum jitter is also priority dependent. The smallest jitter is obtained for the task with highest priority. The situation is shown in Fig. 3.

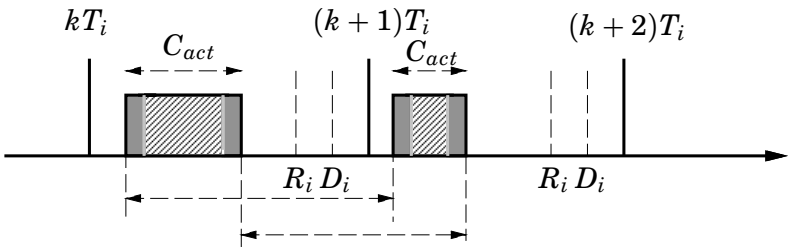

Figure 3 Jitter in successive jobs of a periodic control task.

The actual execution time, C_{act}, of the task varies from invocation to invocation (job to job). It is, however, always smaller than C_i. The execution of the task is always finished by the response time R_i of the task. In order for the task system to be schedulable, R_i should always be less than or equal to the deadline D_i. Jitter is obtained both in the time between subsequent start times of the task, i.e., in the input data collection, and in the time between subsequent end times of the task, i.e., in the output generation.

A common way to handle the computational delay is to write the code in such a way that the delay is minimized. The standard way to do this is to separate the algorithm calculations in two parts: *calculate output* and *update state*. "Calculate output" contains only the parts of the algorithm that make use of the current sample information. "Update state" contains the update of the controller states and the pre-calculations that are needed to minimize the execution time of "calculate output". The code below illustrates the concept:

```
CurrentTime(t);
LOOP
  Data Collection;
  Calculate output;
  Output transmission;
  Update state;
  t := t + h;
  WaitUntil(t);
END;
```

Going even further, we could identify timing requirements for the different parts of the control task:

Data collection should be performed at the same instant every period, in order to eliminate the sampling jitter.

Calculate output should commence and complete as soon as possible after the sample is available, so that the computational delay is minimized.

Output transmission should be performed immediately after "calculate output", or at a fixed instant after the data collection, depending on how the controller was designed.

Update state has to finish before the beginning of the next period, or at least before the release of "calculate output" in the next period.

In the scheduling analysis, the parts of the code could be scheduled as one, two, three, or four separate tasks. However, in many cases it is not relevant to treat the A-D and D-A conversions as separate tasks. They could be implemented as low-cost, high-priority tasks [Locke, 1992], and then be neglected in the analysis. I/O cards with support for periodic sampling is another alternative.

4.4 Scheduling of control tasks

This section summarizes several task models and schedulability analysis methods that could be used when scheduling control tasks.

Fixed-priority scheduling: In the simplest possible setting, each digital controller is described as a periodic task τ_i having a fixed period T_i, a deadline D_i, a worst-case execution time C_i, and a fixed priority P_i. If it is assumed that $D_i = T_i$, then the *rate-monotonic* priority assignment is optimal [Liu and Layland, 1973]. The rate-monotonic approach is appropriate in the case where the control signal is not sent out until the beginning of the next period. The schedulability analysis is very simple, and it is also very simple to compensate for the one-sample delay in the controller.

For some controllers, we could allow $D_i \leq T_i$, in which case the *deadline-monotonic* priority assignment is optimal. The deadlines could be used to improve the response times of some important, but long-period control tasks. Of course, decreasing the deadlines of all controllers will decrease the system schedulability and may even cause the task set to become unschedulable.

Sub-task scheduling: The possibility of decomposing a control task into subtasks was identified in [Gerber and Hong, 1993]. There, the Update State part is named an *unobservable* part, because its result is not directly visible in the output in that period. The reason for the decomposition is to transform an unschedulable task set into a schedulable one. Each task is divided into *three* subtasks (!), one for Calculate Output (retaining the original period) and two for Update State (doubling the original period and thus allowing for deadlines greater than the period). A modified, dynamic rate-monotonic scheduler is used, in which the deadlines of the Update State tasks may be postponed. A *priority exchange* (cf. dual priority) scheme is used to preserve the order among the subtasks.

In [Burns *et al.*, 1994], it was pointed out that Gerber and Hong's method was unnecessary complicated and not optimal. Using response-time analysis, more exact analysis is possible. Also, it is possible to include the case when $D > T$ in the response-time test [Tindell *et al.*, 1994]. When $D > T$, the deadline-monotonic priority assignment is no longer optimal, but priorities can be assigned using an $O(n^2)$ algorithm [Audsley *et al.*, 1993b] where n is the number of tasks.

In a later article [Gerber and Hong, 1997], the authors have abandoned the dynamic priority in their early paper in favor of Burns' response time analysis. Furthermore, they argue that "slicing" a task incurs extra overhead, and they give an algorithm for selecting which tasks should be sliced. The reason for this is not obvious. If the subtasks have the same priority,

why do they have to be implemented as separate tasks? Also, if the subtasks are implemented as separate tasks, why could they not be given different priorities?

A task model with subtasks having different priorities is analyzed in [Gonzalez Härbour *et al.*, 1994]. Here, a task τ_i is described by a worst-case execution time C_i, a period T_i, and a deadline D_i. Each task consists of a number of subtasks τ_{ij}, having a worst-case execution time C_{ij}, a deadline D_{ij}, and a fixed priority P_{ij}. It is assumed that a subtask is not placed in the ready-queue until its predecessor has completed. Thus, we can for instance analyze a set of control tasks with the following structure:

```
LOOP
   SetPriority(P1);
   Wait(ClockInterrupt);
   A_D_Conversion;
   SetPriority(P2);
   CalculateOutput;
   SetPriority(P3);
   D_A_Conversion;
   SetPriority(P4);
   UpdateState;
END
```

In the same article, the special case of subtasks with non-ascending priorities and deadlines less than the period is treated. It is proved that a deadline-monotonic priority assignment among all subtasks is optimal for such a task set. Optimal priority assignment in the general case is not discussed.

Another way of treating subtasks is to use offset scheduling [Audsley *et al.*, 1993b]. Each subtask is assigned an *offset* O_i relative to the release of the first subtask. The correct execution order of the subtasks can be enforced by assigning suitable priorities or offsets. An algorithm for optimal priority assignment with complexity $O(n^2)$ is also given.

Deadline assignment: All of the above schedulability analysis assumes that complete timing attributes of the subtasks are given, i.e. T_i, D_i, C_i, and for offset scheduling O_i. For digital controllers, the period and the worst-case execution time can be assumed to be known. The deadlines and the offsets on the other hand must be *assigned* suitable values. In the two-task model, the deadlines of the Update State parts can be assigned to the period time, while we would like to minimize the deadlines of the Calculate Output parts. The problem of optimal deadline assignment is known to be NP-hard. For large task sets or on-line reconfiguration, heuristic assignment methods must be used.

Deadline assignment has been studied especially in the context of *end-to-end scheduling* in distributed systems. In [Sun *et al.*, 1994], some priority (deadline) assignment schemes are mentioned. The approach of interest to us is the *effective-deadline-monotonic* assignment. Subtasks are assigned deadlines according to

$$D_{ij} = D_i - \sum_{k=j+1}^{n_i} C_{i,k} \tag{2}$$

That is, later subtasks are given just enough time to execute after the deadlines of earlier subtasks.

In [Kao and Garcia-Molina, 1993], several schemes for subtask deadline assignment are evaluated, e.g., *effective deadline* (the same as effective deadline-monotonic), *equal slack*, and *equal flexibility*. In equal slack, the total slack time in the period is divided evenly among the subtasks. This produces earlier deadlines than the effective deadline assignment. In equal flexibility, the total slack time is divided among the subtasks in proportion to their execution times. In terms of the number of missed deadlines, the two last approaches are superior to effective deadline, with equal flexibility being the best. These deadlines assignment schemes are however intended for *soft* real-time systems—applied on the parts of digital controllers, deadlines may very well be missed.

In [Cervin, 1999], the following heuristic is presented for minimizing the deadlines of the Calculate Output parts, while maintaining schedulability:

1. Assign *effective* deadlines to the subtasks, i.e. set $D_{CO\,i} := T_i - C_{US\,i}$.
2. Assign deadline-monotonic priorities.
3. Calculate the response times of the subtasks.
4. Decrease the deadlines of the Calculate Output parts by assigning $D_{CO\,i} := R_{CO\,i}$.
5. Repeat from 2 until no further improvement is given.

The heuristic works because of the optimality of the deadline-monotonic priority assignment. The task set is schedulable after each improvement, and the algorithm terminates after a finite number of steps.

Delay jitter minimization: The second way of solving the computational delay problem is to try to make the delay as deterministic as possible. Our desire is to be able to make the time between the start of the calculate output subtask and the end of the calculate output subtask as deterministic as possible, i.e., to obtain an upper and lower bound on this time. From a control point of view it may be an advantage with tight upper and lower bounds, rather than having the smallest lower bound (which is equal to the best case execution time for the calculate output subtask). How to best obtain such tight bounds in a pre-emptive scheduling setting is still an open question.

Quite a lot of work has been performed on jitter control in real-time scheduling. In the case of static cyclic executive scheduling, the problem is essentially solved. For example, in [Xu and Parnas, 1990] an algorithm is presented that finds an optimal schedule for a given task set under the presence of a variety of different constraint types. However, also in priority-based scheduling a lot of work has been performed. The work in [Audsley *et al.*, 1993a; Baruah *et al.*, 1997] addresses the problem of accommodating input jitter, i.e., scheduling systems of periodic tasks in which the ready-times of jobs cannot be predicted exactly a priori, but are subject to timing uncertainties at run-time. In [Lin and Herkert, 1996] a distance-constrained task model is studied in which the difference between any two consecutive end-times of the same task is required to be bounded by a specified value. This approach attempts to minimize the output jitter. In [Baruah and Gorinsky, 1999] an attempt is made to identify some of the

properties that any jitter-minimization scheme should satisfy. The scheme is not allowed to buffer a job that is ready to execute. The scheme may not insert idle time in the execution of a job. The scheme is not allowed to buffer a job that is completed. The scheme should not require too many preemptions and the on-line scheduling overhead should be small. Within the context of these conditions two jitter-minimizing scheduling algorithms are proposed. In [Cottet and David, 1999] a scheme is proposed that reduces jitter in the case of deadline monotonic and EDF scheduling. The scheme is based on shifting the periodic tasks by introducing task offsets and to ensure that the tasks have highest priority when they are requested. If this scheme is combined with postponing output transmission until its deadline; if it finishes before its deadline is reached, the jitter in computational will be very small.

Multiple-task approach: In the multiple-task approach the problem of mapping the control loop timing constraints into task attributes becomes more complicated. The *period calibration method (PCM)*, [Gerber *et al.*, 1995], derives task attributes from a given task graph design of a system and the specified end-to-end constraints (e.g., constraints on input-output latency). The tasks are implemented as periodic tasks that are invoked and executed periodically at a fixed rate. The method involves the solution of an NP-hard nonlinear optimization problem. Two heuristics based on harmonicity constraints between the task periods are presented in [Ryu and Hong, 1999]. Based on these, a polynomial-time algorithm is proposed that in most cases finds solutions that are very close to the optimal ones. The problem of task attribute assignment is also studied in [Bate and Burns, 1999].

5. Scheduling under overload conditions

One of the key issues in this project is the relaxation of the requirement on a known worst-case task execution time. Our main approach to this will be to use on-line measurements of actual execution time. However, there are also other possibilities. The simplest approach is to simply treat the actual execution times that are longer than the worst-case bound as overload conditions and then try to use some of the techniques that have been developed for this, for example robust scheduling. In this section we will give an overview of the work that has been performed in this area. For now we will focus on static approaches—dynamic approaches will be discussed in Section 7.

The model used in overload scheduling often associates a value with the completion of a task within its deadline. For a hard task, the failure to meet the deadline is considered intolerable. For soft tasks, a positive value is associated with each task. If a soft task succeeds then the system gains its value. If a soft task fails, then the value gained by the system decreases. Sometime also the notion of a *firm task* is used. For a firm task there is no value for a task that has missed its deadline, but there is no catastrophe either. An overview of value-based scheduling is given in [Burns, 1998]. When a real-time system is overloaded, not all tasks can be completed by their deadlines. Unfortunately, in overload situations there is no optimal on-line algorithm that can maximize the cumulative value of a task set.

Hence, scheduling must be made using *best-effort* algorithms. The objective is to complete the most important of the tasks by their deadline, and to avoid unwanted phenomena such as the so called *domino effect*. This happens when the first task that misses its deadline may cause all subsequent tasks to miss their deadlines. For example, dynamic-priority schemes such as EDF are especially prone to domino effects. In fixed-priority scheduling the user has more control over which tasks that will meet their deadlines also in the case of overloads.

It is normally assumed that when a task is released its value and deadline are given. The computation time may be known either precisely or within some range. The *value density* of a task is its value divided by the computation time. The *importance ratio* of a set of tasks is the ratio of the largest value density to the smallest value density. When the importance rate is 1, the task set has *uniform value density*, i.e. the value of a task equals its computation time. An on-line scheduler has a competitive factor, $r, 0 < r \leq 1$, if and only if it is guaranteed to achieve a cumulative value of at least r times the cumulative value achievable for an off-line scheduler on any task set. The competitive multiplier is defined as $1/r$. In [Shasha and Koren, 1995], D^{over}, an optimal on-line scheduling algorithm is presented. It schedules to completion all tasks in non-overload periods and achieves at least $1/(1+\sqrt{k})^2$ of the value of an off-line scheduler, where k is the importance ratio. The method also relaxes the firm deadline assumption by giving a positive value to firm tasks even if they complete after their deadlines.

In [Buttazzo *et al.*, 1995], a comparative study is performed of four priority-assignment schemes, EDF, HVF (highest value first), HDF (highest value density first), and a mixed scheme where the priority is computed as a weighted sum of the value and the deadline. The four basic algorithms were all extended into two additional classes: a class of guaranteed algorithms, characterized by a task acceptance test, and a class of robust algorithms, characterized by a task rejection mechanism. Simulation experiments showed that the robust versions of the algorithms were the most flexible ones.

The transform-task method [Tia *et al.*, 1995] uses a threshold value to separate jobs guaranteed by rate-monotonic scheduling from those which would require additional work. The latter jobs are split into two parts. The first part is considered as a periodic job with a resource requirement equal to the threshold. The second part is considered to be a sporadic job and is scheduled via a sporadic server when the periodic part has completed.

It is difficult to improve overload performance with purely algorithmic techniques, what is needed is essentially more processing power. Several approaches have considered the use of more processing power in order to reduce the execution requirements of the tasks, e.g., [Baruah and Haritsa, 1997]. Another approach is to replicate the processor with several identical copies. In [Baruah, 1998b] a characterization is given of the relationship between overload performance and the number of processors needed.

6. Control and Scheduling Co-Design

A prerequisite for an on-line integration of control and scheduling theory is that we are able to make an integrated off-line design of control algorithms and scheduling algorithms. Such a design process should ideally allow an incorporation of the availability of computing resources into the control design by utilizing the results of scheduling theory. This is an area where, so far, relatively little work has been performed.

One of the first references that addressed this problem was [Seto *et al.*, 1996]. An algorithm was proposed that translates a control performance index into task sampling periods considering schedulability among tasks running with pre-emptive priority scheduling. The sampling periods were considered as variables and the algorithm determined their values so that the overall performance was optimized subject to the schedulability constraints. Both RM and EDF scheduling were considered. The performance index was approximated by an exponential function only and the approach did not take input-output latency into account. The approach was further extended in [Seto *et al.*, 1998b].

An approach to 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. A heuristic iterative algorithm is proposed for the optimization of these parameters subject to schedulability constraints. The algorithm is based on using the period calibration method (PCM) for determining the task attributes. A case study involving the control design for a CNC controller is presented. The tasks are scheduled using EDF and a cyclic executive is used for run-time dispatching. The same application is further explored in [Kim *et al.*, 1999] where it was revealed that the intertask communication scheme of PCM may incur large latencies, and that the absence of overload handling is a critical limitation.

7. Dynamic Task Adaptation

A key issue in any system that allows dynamic feedback between the control algorithms and the on-line scheduler is the ability to dynamically adjust task parameters. Reasons for the adjustments could for example be to improve the performance in overload situations and to dynamically optimize control performance. Examples of task parameters that could be modified are periods and deadlines. One could also allow the maximum allowed worst-case execution time for a task to be varied. In order for this to be realistic, the controllers must support dynamically changing execution times. Changes in the task period and in the execution time both have the effect of changing the utilization that the task requires.

7.1 Quality-of-service resource allocation

Much of the work on dynamic task adaptation during recent years is motivated by the requirements of multimedia applications. Activities such

as voice sampling, image acquisition, sound generation, data compression, and video playing are performed periodically, but with less rigid timing requirements than those that can sometimes be found in closed-loop control systems. Missing a deadline may decrease the quality of service (QoS) but does not cause critical system faults. Depending on the requested QoS, tasks may adjust its attributes to accommodate the requirements of other concurrent activities.

On-line admission control has been used to guarantee predictability of services where request patterns are not known in advance. This concept has also been applied to resource reservation for dynamically arriving real-time tasks, e.g. in the Spring kernel [Stankovic and Ramamritham, 1991]. A main concern of this approach is predictability. Run time guarantees given to admitted tasks are never revoked, even if they result in rejecting subsequently arriving, more important requests competing for the same resources. In soft real-time systems, services are more concerned with maximizing overall utility, by serving the most important requests first, than guaranteeing reserved resources for individual requests. Priority-driven services can be categorized this way, and are supported in real-time kernels such as Mach [Tokuda *et al.*, 1990]. Under overload conditions, lower-priority tasks are denied service in favor of more important tasks. In the Rialto operating system [Jones and Leach, 1995], a resource planner attempts to dynamically maximize user-perceived utility of the entire system.

Q-RAM, a resource allocation scheme for satisfying multiple QoS dimensions in resource constrained environments was presented in [Rajkumar *et al.*, 1997]. Using the model, available system resources can be apportioned across multiple applications such that the net utility accrued to the end users of those applications could be maximized. In [Lee *et al.*, 1998], the mirror problem of apportioning multiple resources to satisfy a single QoS dimension is studied. In [Abdelzaher *et al.*, 1997] a QoS renegotiation scheme is proposed as a way to allow graceful degradation in cases of overload, failures or violation of pre-run-time violations. The mechanism permits clients to express, in their service requests, a spectrum of QoS levels they can accept from the provider and perceived utility of receiving service at each of these levels. Using this, the application designer, e.g., control engineer, is able to express acceptable tradeoffs in QoS and their relative cost/benefit. The approach is demonstrated on an automated flight-control system.

7.2 Period skipping

A simple task attribute adjustment is to skip an instantiation of a periodic task. This is equivalent to require that the task period should be doubled for this particular instantiation, or that the maximum allowed execution time should be zero. Scheduling of systems that allow skips is treated in [Koren and Shasha, 1995] and [Ramanathan, 1997]. The latter paper considers scheduling that guarantees that at least k out of n instantiations will execute. A slightly different motivation for skipping samples is presented in [Caccamo and Buttazzo, 1997]. Here the main objective is to use the obtained execution time to enhance the responsiveness of aperiodic tasks.

7.3 Task attribute adjustments

In [Buttazzo *et al.*, 1998] an elastic task model for periodic tasks is presented. Each task is characterized by five parameters: computation time C_i, a nominal period T_{i_0}, a minimum period $T_{i_{min}}$, a maximum period $T_{i_{max}}$, and an elasticity coefficient $e_i \geq 0$. A task may change its period within its bounds. When this happens the periods of the other tasks are adjusted so that the overall system is kept schedulable. An analogy with a linear spring is used, where the utilization of a task is viewed as the length of a spring that has a given rigidity coefficient $(1/e_i)$ and length constraints. The elasticity coefficient is used to denote how easy or difficult it is to adjust the period of a given task (compress the string). A task with $e_i = 0$ can arbitrarily vary its period within its range, but it cannot be varied by the scheduler during load reconfiguration. The approach can be used under fixed or dynamic priority scheduling. In principal it is possible to modify the approach so that it also adjusts execution times.

Adjustment of task periods has also been suggested by others. For example, [Kuo and Mok, 1991] propose a load-scaling technique to gracefully degrade the workload of a system by adjusting the task periods. Tasks are assumed to be equally important and the objective is to minimize the number of fundamental frequencies to improve schedulability under static priority assignments. In [Nakajima and Tezuka, 1994] a system is presented that increases the period of a task whenever the deadline of the task is missed. In [Lee *et al.*, 1996] a number of policies to dynamically adjust task rates in overload conditions are presented. In [Nakajima, 1998] it is shown how a multimedia activity can adapt its requirements during transient overloads by scaling down its rate or its computational demands.

The MART scheduling algorithm [Kosugi *et al.*, 1994; Kosugi *et al.*, 1996; Kosugi and Moriai, 1997] also supports task-period adjustments. In [Kosugi *et al.*, 1999] MART is extended to also handle task execution time adjustments. The system handles changes in both the number of periodic tasks and in the task timing attributes. Before accepting a change request the system analyzes the schedulability of all tasks. If needed it adjusts the period and/or execution time of the tasks to keep them schedulable with the rate monotonic algorithm. For the task execution time it is assumed that a minimum value exists in order for the task to guarantee a minimum level of service. For the task-period, neither minimum nor maximum are assumed to exist. The MART system is implemented on top of Real-Time Mach.

In [Shin and Meissner, 1999] the approach in [Seto *et al.*, 1996] is extended, making on-line use of the proposed off-line method for processor utilization allocation. The approach allows task-period changes in multiprocessor systems. A performance index for the control tasks is used to determine the value to the system of running a given task at a given period. The index is weighted for the task's importance to the overall system. The paper also discusses the issue of task reallocation from one processor to another, the need for consideration of the transient effects of task reallocations, and the question of determining a value for running a redundant shadow task as opposed to fast recovery. Two algorithms are given for task reallocation and period adjustments. An inverted pendulum control system is used as an example.

7.4 Mode changes

Mode changes for priority-based preemptive scheduling is an issue that has received some interest. In the basic model, the system consists of a number of tasks with task attributes. Depending on which modes the system and the tasks are in, the task attributes have different values. During a mode change, the system should switch the task attributes for a task and/or introduce or remove tasks in such a way that the overall system remains schedulable during and after the mode change.

A simple mode change protocol was suggested in [Sha *et al.*, 1989]. The protocol assumes that an on-line record of the total utilization is kept. A task may be deleted at any time, and its utilization may be reclaimed by a new task at the end of the old task's period. The new task is accepted if the resulting new task set is schedulable according to the rate-monotonic analysis. The locking of semaphores during the mode change (according to the priority ceiling protocol) is also dealt with.

In [Tindell *et al.*, 1992], it was pointed out that the analysis of Sha *et al.* was faulty. Tasks may miss their deadlines during a mode change, even if the task set is schedulable both before and after the switch. The transient effects of a mode change can be analyzed by extending the deadline-monotonic framework. Formulas for the worst-case response times of old and new tasks across the mode change are given. New tasks may be given release offsets (relative to the mode change request) to prevent tasks from missing their deadlines. No hints are given as to how these offsets should be chosen.

The deadline-monotonic mode change analysis was both extended and modified in [Pedro and Burns, 1998]. The analysis can account for old tasks that are aborted instantly at the mode change request. Furthermore, *all* tasks present in the system after the mode change (i.e. unchanged, changed, and wholly now tasks) must be assigned release offsets relative to the mode change request. Since the offsets are constant (and assigned off-line), even unchanged tasks will experience unpredictable and unnecessary delay (jitter) during the mode change. Again, few clues are given as to how the offsets could be assigned.

It is interesting to note, that under EDF scheduling, the reasoning about the utilization from [Sha *et al.*, 1989] actually seems to hold. In [Buttazzo *et al.*, 1998], EDF scheduling of a set of tasks with deadlines equal to their periods is considered. It is shown that a task can decrease its periods at its next release, as long as the total utilization remains less than one. It is not known whether the computation time may be changed at the same time. A drawback of using EDF is that the case of deadlines less than periods is more difficult to analyze. Also, the computation of worst-case response times (if needed) is more complex than under fixed-priority scheduling. Another challenge is how to manage overloads, e.g., ensuring that the deadlines of critical tasks will not be missed.

7.5 Feedback scheduling

Viewing a computing system as a dynamical system or as a controller is an approach that has proved to be fruitful in many cases. For example, the step-length adjustment mechanism in numerical integration algorithms can be viewed as a PI-controller [Gustafsson, 1991], and the traveling salesman optimization problem can be solved by a nonlinear dynamical

system formulated as a recurrent neural network. This approach can also be adopted for real-time scheduling, i.e., it is possible to view the on-line scheduler as a controller. Important issues that then must be decided are what the right control signal, measurement signals, and set-points are, what the correct control structure should be, and which process model that may be used. The idea of using feedback in scheduling has to some extent been used previously in general purpose operating systems in the form of multi-level feedback queue scheduling [Kleinrock, 1970; Blevins and Ramamoorthy, 1976; Potier *et al.*, 1976]. However, this has mostly been done in an ad-hoc way.

So far very little has been done in the area of real-time feedback scheduling. A notable exception is [Stankovic *et al.*, 1999] where it is proposed to use a PID controller as an on-line scheduler under the notion of Feedback Control-EDF (FC-EDF). The measurement signal (the controlled variable) is the deadline miss ratio for the tasks and the control signal is the requested CPU utilization. Changes in the requested CPU utilization are effectuated by two mechanisms (actuators). An admission controller is used to control the flow of workload into the system and a service level controller is used to adjust the workload inside the system. The latter is done by changing between different versions of the tasks with different execution time demands. A simple liquid tank model is used as an approximation of the scheduling system.

Using a controller approach of the above kind it is important to be able to measure the appropriate signals on-line, for example to be able to measure the deadline miss ratio, the CPU utilization, or the task execution times. On-line measurements of these entities are not so often discussed. One exception is [Mok and Liu, 1997].

The controller approach proposed in [Stankovic *et al.*, 1999] is centralized in the sense that a central controller/scheduler controls the global schedulability of the system. An interesting approach would be to try to instead use decentralized control. Using this approach the periodic tasks themselves could be viewed as local controllers that adjust their timing attributes in such a way that the local and the global performance is optimized.

7.6 Statistical scheduling

A slightly different approach to flexibility and uncertainty handling in scheduling is obtained by using statistical scheduling. Using this approach things like, e.g., task execution times are modeled by stochastic variables with a given distribution rather than constant variables. Instead of generating hard guarantees that the tasks will meet their deadlines or that dynamically generated tasks will be admitted, the guarantees are probabilistic.

Statistical approaches to scheduling are relatively easy to find. In [Tia *et al.*, 1995] an analysis was given for the probability that a sporadic job would meet its deadline when using the transform-task method. The Statistical Rate Monotonic Scheduling (SRMS) [Atlas and Bestavros, 1998] is a generalization of the classical RM scheduling and the semi-periodic task model of [Tia *et al.*, 1995]. A task is modeled by three attributes: the period, the probability density function for the task's periodic resource utilization requirement, and the task's requested QoS. The SRMS algorithm consists of two parts: a job admission controller and a scheduler. The scheduler is a

simple pre-emptive fixed priority scheduler. The job admission controller is responsible for maintaining the QoS requirements of the tasks through admit/reject and priority assignment decisions. Using SRMS the probability that an arbitrary job is admitted can be calculated.

In the model proposed in [Abeni and Buttazzo, 1999] each task is described by a pair of probability density functions: $U_i(c)$, which decides the probability that the execution time is c and $V_i(t)$ which decides the probability that the minimum inter-arrival time of the task is t. The method guarantees that *probabilistic deadlines*, δ, are respected with a given probability. A dynamic priority scheduler based on EDF is used. Each soft task is handled by a dedicated constant bandwidth server. The server assigns each job an initial deadline. The assigned deadline is postponed each time the task requests more than the reserved bandwidth.

7.7 Imprecise calculations

The possibility to adjust the allowed maximum execution time for a task necessitates an approach for handling tasks with imprecise execution times, particularly the case when the tasks can be described as "any-time algorithms", i.e., algorithms that always generate a result (provide some QoS) but where the quality of the result (the QoS level) increases with the execution time of the algorithm.

The group of Liu has worked on scheduling of imprecise calculations for long time [Liu *et al.*, 1987; Chung *et al.*, 1990; Liu *et al.*, 1994]. In [Liu *et al.*, 1991] imprecise calculation methods are categorized into *milestone methods*, *sieve function methods*, and *multiple version methods*. Milestone methods use monotone algorithms that gradually refine the result and 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]. Two models are proposed. 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. The two models differ with respect to if the optional part needs to complete or not. Scheduling of tasks where it is the average error between the results from consecutive jobs that is important uses a conservative and predictable strategy for the mandatory parts and a less conservative strategy for the optional parts. A number of RM-based scheduling algorithms for these types tasks are described. The schedulability of tasks where it instead is the cumulative effect of the errors that is important is also discussed.

7.8 Dynamic system upgrades

An alternative view of the need for flexibility is obtained if we look upon the problem of performing safe on-line upgrades of real-time systems, in particular safety-critical real-time control systems. The Simplex group at SEI/CMU has developed a framework, Simplex, that allows these types of on-line upgrades [Seto *et al.*, 1998a; Sha, 1998]. The basic building block

of Simplex is the replacement unit. A replacement unit consists of a task together with a communication template. The replacement units are organized into application units that also contain communication, and task management functions. A special safety unit is responsible for basic reliable operation and operation monitoring. A typical example of a replacement unit is a controller. A common setup is that the system contains two units: a safety controller and a baseline controller. The baseline controller provides the nominal control performance and it is assumed that this controller is executing when a new upgraded version of this controller should be installed.

Simplex considers a three-dimensional fault model consisting of timing faults, system faults, and semantical faults that effect the logical control behavior of the system. RM analysis is used to ensure schedulability under fault-free conditions. If a new controller does not return an answer before the deadline, Simplex switches in the safety controller, and then eventually, when the controlled process is back in a safe state, switches back to the baseline controller. System faults generated by the new controller are trapped by the operating system. In the same way as before Simplex will switch to the safety controller and then back to the baseline controller. Semantic faults are caught by the analytical redundancy obtained by having the safety controller monitoring the operation. The safety controller is responsible for monitoring that the controlled system is in a safe state and that the performance is better then that obtained by the baseline controller. If any of these two conditions should not be fulfilled, the safety controller is switched in the same way as before.

Dynamic change management and version handling is a large area where a lot of work has been performed, e.g., [Kramer and Magee, 1990; Stewart *et al.*, 1993; Gupta and Jalote, 1996]. However, most of this is outside the area of this report.

8. Flexibility and aperiodicity in control

The aim of this section is to give an overview of the possibilities that control theory provides in the context of integrated control and scheduling. Special emphasis will be given to how changes in sampling periods could be handled, and to which extent different control algorithms can viewed as imprecise algorithms of the type discussed in Section 7.

8.1 Controller implementation

A computer-based control system can be designed in two different ways:

1. Discrete-time design
2. Discretization of a continuous-time design

In both cases the interface to the process consists of AD and DA converters. The AD converter acts as a sampler that returns a snapshot value of a continuous time signal, and the DA converter acts as a hold circuit that takes a discrete-time signal and converts it into a continuous-time signal. Normally zero-order hold is used, in which case the resulting continuous-time signal is piecewise constant between the DA conversions. In certain situations it is advantageous to instead use first-order hold, where the continuous-time

signal becomes piecewise linear. In the first-order hold case, the DA conversion is implemented as a high-frequency periodic process, unless special DA hardware is used.

Discrete-time design: The basic idea behind discrete or sampled control theory is to only consider the system through the values of the system inputs and outputs at the sampling instants, i.e., from the point of view of the computer according to Fig. 4.

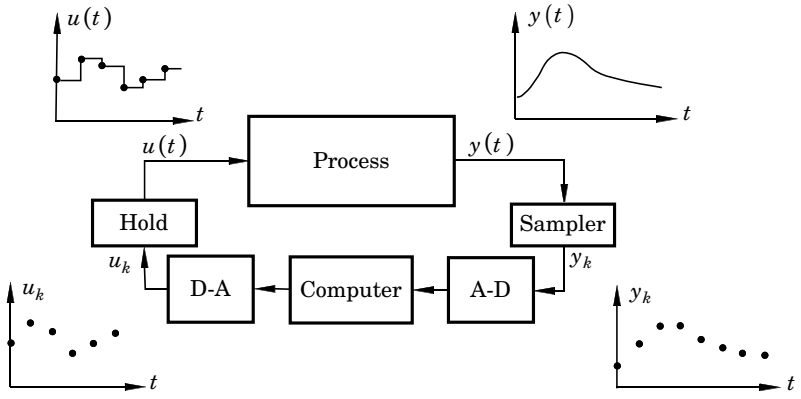

Figure 4 Sampled control loop

In order to do this, a sampled version of the continuous system model is derived. This is done by letting the process inputs be piecewise constant signals and then solving the system equation by calculating step responses as in the following. Given the continuous time system description

$$\begin{aligned}\frac{dx}{dt} &= Ax(t) + Bu(t)\\ y(t) &= Cx(t) + Du(t),\end{aligned}$$

the solution to the system equation is given by

$$\begin{aligned}x(t) &= e^{A(t-t_k)}x(t_k) + \int_{t_k}^{t} e^{A(t-s')}Bu(s')\,ds'\\ &= e^{A(t-t_k)}x(t_k) + \int_{t_k}^{t} e^{A(t-s')}\,ds'\,Bu(t_k) \quad (u \text{ piecewise constant})\\ &= e^{A(t-t_k)}x(t_k) + \int_{0}^{t-t_k} e^{As}\,ds\,Bu(t_k) \quad (\text{variable change})\\ &= \Phi(t,t_k)x(t_k) + \Gamma(t,t_k)u(t_k)\end{aligned}$$

From this the values at $t = t_{k+1}$ are given by

$$\begin{aligned}x(t_{k+1}) &= \Phi(t_{k+1},t_k)x(t_k) + \Gamma(t_{k+1},t_k)u(t_k)\\ y(t_k) &= Cx(t_k) + Du(t_k)\end{aligned}$$

where

$$\begin{aligned}\Phi(t_{k+1},t_k) &= e^{A(t_{k+1}-t_k)}\\ \Gamma(t_{k+1},t_k) &= \int_{0}^{t_{k+1}-t_k} e^{As}ds\ B\end{aligned}$$

The expression above is valid both for periodic and aperiodic sampling. However, normally periodic sampling, i.e., $t_k = k \cdot h$, is assumed. This leads to the well-known discrete-time system description.

$$\begin{aligned} x(kh+h) &= \Phi x(kh) + \Gamma u(kh) \\ y(kh) &= Cx(kh) + Du(kh) \end{aligned}$$

where

$$\begin{aligned} \Phi &= e^{Ah} \\ \Gamma &= \int_0^h e^{As}\, ds\, B \end{aligned}$$

The resulting system description is time-invariant and only describes the system at the sampling instants. Using similar techniques it is possible to also sample systems with time delays, both in the simple case when the time delay is a multiple of the sampling interval and in the case when the time delay is a fraction of the sampling interval.

From the state-space description above it is also possible to derive input-output models. These are often based on the forward shift operator

$$qf(k) = f(k+1)$$

or the backward shift (delay) operator

$$q^{-1}f(k) = f(k-1)$$

Alternatively, input-output models can be based on z-transforms.

The sampling intervals for discrete-time control designs are normally based on the desired speed of the closed loop system. A common rule-of-thumb is that one should sample $4--10$ times per rise time T_r of the closed loop system.

$$N_r = \frac{T_r}{h} \approx 4--10$$

This gives relatively long sampling intervals, compared to what is used when discretization-based design is used. The long sampling interval also means that it may take long time before, e.g., load disturbances are detected by the controller. The reason for this is that the disturbances are not synchronized with the sampling.

Discretization of continuous-time design: The idea behind this design philosophy is to perform the design in the continuous time domain, and then approximate this design by a computer-based controller through fast sampling. Using this approach it is not necessary to employ any special sampled control design theory. The price that one pays for this is higher requirements on fast sampling.

Assume that a controller has been designed in continuous time and that this controller is expressed on input-output form, i.e., on Laplace transform form $G(s)$. The goal now is to approximate this design in such a way that A/D + Algorithm + D/A $\approx G(s)$ according to Fig. 5

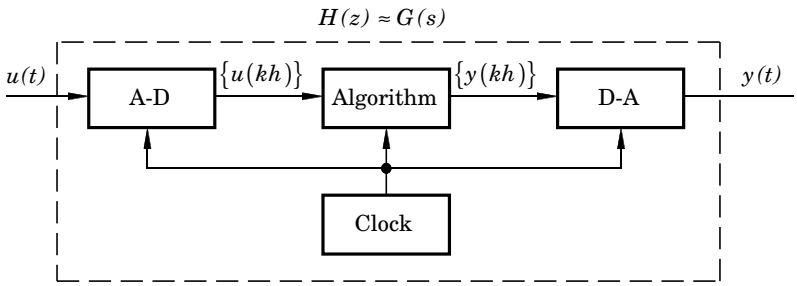

Figure 5 Approximation of continuous-time design

This can be done in several ways. The most straightforward way is to use a simple Euler forward or backward approximation. In forward approximation a derivative is replaced by its forward approximation, i.e,

$$\frac{dx(t)}{dt} \approx \frac{x(t+h)-x(t)}{h}$$

This is equivalent to replacing the Laplace operator s with $(z-1)/h$ in $G(s)$. In the backward approximation the derivative is instead replaced by

$$\frac{dx(t)}{dt} \approx \frac{x(t)-x(t-h)}{h}.$$

This is equivalent to replacing s with $(z-1)/zh$. The forward approximation maps the continuous-time stability region (the left half plane) to the half plane to the left of the line with real part equal to one in the complex plane. Since the discrete time stability region is the unit circle the approximation of a stable continuous-time system may lead to an unstable discrete-time system. The backward approximation instead maps the left half plane into a smaller circle within the unit circle. A one-to-one mapping between the left half plane and the unit circle is given by the Tustin approximation where s is instead replaced by $2(z-1)/h(z+1)$.

Several rules-of-thumbs exist for choosing the sampling interval for discretization based designs. A simple rule is the faster the better up to a certain limit when the limited word-length of the computer becomes a problem. One example of a rule-of-thumb is

$$h\omega_c \approx 0.15 - 0.5$$

where ω_c is the cross-over frequency of the continuous-time system (the frequency where the gain is 1). This gives substantially smaller sampling intervals then for discrete-time design. However it also means that the resulting controller is less sensitive to relative variations in the sampling interval. A variation of 100% in the sampling interval for a discretization based design may be fairly benign whereas a similar relative variation in the sampling interval for a discrete-time design would in many cases lead to an unstable system.

8.2 Compensation for varying sampling intervals

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.

There are several possible compensation methods, ranging from simple ad-hoc techniques to quite advanced techniques requiring extensive (off-line or on-line) 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.

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. However, the changes in sampling period often occur continuously and should not be treated this way. The simplest compensation methods are ad-hoc, but seem to work quite well. One possibility is to modify the approximation methods mentioned in Section 8.1. For example, the (backward) approximation method becomes

$$\frac{dx(t)}{dt} \approx \frac{x(t_{k+1}) - x(t_k)}{h_k}$$

where $h_k = t_{k+1} - t_k$ is time-varying. This works fine for the I- and D- parts in a PID controller[2]

```
h = time-oldtime
e = r-y
P = Kp*e
I = I + Ki*e*h
D = Kd*(e-olde)/h
u = P + I + D
oldtime = time
olde = e
```

The idea must be modified for higher order controllers, see [Wittenmark and Åström, 1980], [Albertos and Salt, 1990].

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 (here illustrated with the Kalman filter without direct term):

$$\begin{aligned} u(t_k) &= -L\hat{x}(t_k) \\ \hat{x}(t_{k+1}) &= \Phi(h_k)\hat{x}(t_k) + \Gamma(h_k)u(t_k) + Ke(t_k) \end{aligned}$$

[2]The code for PID and Kalman filtering shown in this section are only skeletons showing the main details, and are not recommended for implementation.

where $\Phi(h_k)$ and $\Gamma(h_k)$ can be pre-calculated in one-dimensional tables. More involved schemes can also be used, where also L and K depend on $h_k = t_{k+1} - t_k$.

Another 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 *et al.*, 1998b; Nilsson *et al.*, 1998a].

8.3 Compensation for computational delay

In the analysis in the beginning of Section 8 and in the previous discussion about varying sampling period it was assumed that the A-D node and the D-A node where synchronized so that the input signal changes instantaneously at the A-D sampling instances t_k. There are several reasons why this might not hold in reality. First of all, the A-D node, the computer node, and the the D-A nodes might use different clocks if they are implemented in different hardware nodes. They may also be implemented as separate tasks, so that one or several context switches need to be performed. Secondly, the A-D and D-A action might be separated in time, both by the computation time of the control action and by transmission time of signals over a communication network. The time between the the related sampling and actuation actions is called the *control delay*.

To sample a system composed of a control delay and a continuous-time state-space system, see Figure 6, one can follow the calculation in the beginning of Section 8 replacing the system equations with

$$\begin{aligned} \frac{dx}{dt} &= Ax(t) + Bu(t-\tau) \\ y(t) &= Cx(t) + Du(t) \end{aligned}$$

The sampled system will now be on the form (assuming that the control delay τ is less than the sampling interval h)

$$x(kh+h) = \Phi x(kh) + \Gamma_0 u(kh) + \Gamma_1 u(kh-h),$$

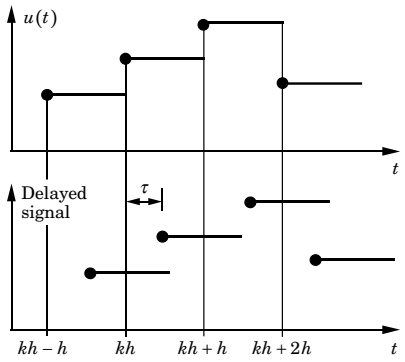

Figure 6 Relationship among $u(t)$ and the delayed signal $u(t-\tau)$, and the sampling instances.

where

$$\Phi = e^{Ah}$$

$$\Gamma_0 = \int_0^{h-\tau} e^{As}\, ds\, B$$

$$\Gamma_1 = \int_{h-\tau}^{h} e^{As}\, ds\, B$$

If the control delay is fixed, and known in advance, it is possible to compensate for it in the controller design. 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 control delay is varying from sample to sample the problem becomes more complicated. If the control delay is bounded, i.e. $0 \leq \tau \leq \tau_{max}$ then one possibility is to introduce buffers and always wait for the longest possible delay τ_{max}. This can however be quite pessimistic and lead to decreased obtainable performance. Another possibility is to design for a good estimate of the control delay, such as the average value, or the last value of the delay.

A third option is possible if probabilistic knowledge of the delay is present. Then stochastic control theory can be used to optimize performance. The setup in Fig. 7 is studied in [Nilsson, 1998]. The conclusion in this work is that very good performance can be obtained even under quite large variations of control delay. The case of joint variations in control delay τ and sampling period h is treated in [Nilsson *et al.*, 1998a].

8.4 Imprecise control algorithms

Scheduling of imprecise calculations is an area where the scheduling community has been active. As discussed in Section 7 imprecise calculation methods can be categorized into milestone methods, sieve function methods, and multiple version methods. Examples of all three types can be found in control. Control algorithms that are based on on-line numerical solution to an optimization problem every sample can be viewed as milestone or "any-time" methods. The result generated from the control algorithm is gradually refined for each iteration in the optimization up to a certain

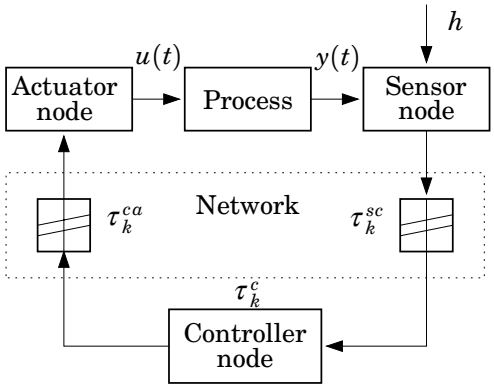

Figure 7 Distributed digital control system with communication delays, τ_k^{sc} and τ_k^{ca}. The computational delay, τ_k^c, is also indicated. The control delay equals $\tau_k^{sc} + \tau_k^c + \tau_k^{ca}$.

bound. Examples of this type of control algorithms are found in model-predictive control. Similar situations exists for all control algorithms that can be written on iterative form, e.g., on series form. One possibility could, e.g., be controllers on polynomial forms where the terms are arranged in decreasing order of magnitude.

It is also straightforward to find examples of controllers that be cast on sieve function form. For example, in an indirect adaptive controller the updating of the adapted parameters can be skipped when time constraints are hard, or, in a LQG controller the updating of the observer can be skipped when time constraints are hard. Similarly, in a PID controller the updating of the integral part may be skipped if needed. The extreme case is of course to skip the entire execution of the controller for a certain sample. This is equivalent to temporarily doubling the sampling interval. However, in all cases it is important to guarantee that the skips are only performed temporarily, and not too frequent. Also, the system may be more or less sensitive to skips depending on the external conditions. For example, during system transients due to, e.g., a set-point change or a load disturbance it the skip of a part of the control algorithm and the resulting degraded performance may be less welcome than during steady-state operation. A similar situation holds for variations in sampling interval and variations in computational delay.

Finally it is also possible to find examples of multiple version methods in control, e.g., situations with one nominal control algorithm and one backup algorithm. However, in most cases the motivation for having the backup algorithm is control performance rather then timing issues, compare the discussions about the Simplex algorithm in Section 7.

Model predictive control: Model predictive control (MPC) is methodology for solving control problems in the time domain. It is based on three main concepts:

1. Explicit use of a model to predict the process output at future discrete time instants, over a prediction horizon.
2. Computation of a sequence of future control actions over a control horizon by minimizing a given objective function, such that the pre-

dicted process output is as close as possible to a desired reference signal.

3. Receding horizon strategy, so that only the first control action in the sequence is applied, the horizons are moved towards the future and optimization is repeated.

The future process outputs are predicted over the *prediction horizon* H_p using a model of the process. The predicted output values, denoted $\hat{y}(k+i)$ for $i = 1, \ldots, H_p$, depend on the state of the process at the current time k (for input–output models, for instance, represented by a collection of past inputs and outputs) and on the future control signals $u(k+i)$ for $i = 0, \ldots, H_c - 1$, where H_c is the *control horizon*. If H_c is chosen such that $H_c < H_p$, the control signal is manipulated only within the control horizon and remains constant afterwards, i.e., $u(k+i) = u(k+H_c-1)$ for $i = H_c, \ldots, H_p - 1$, see Figure 8.

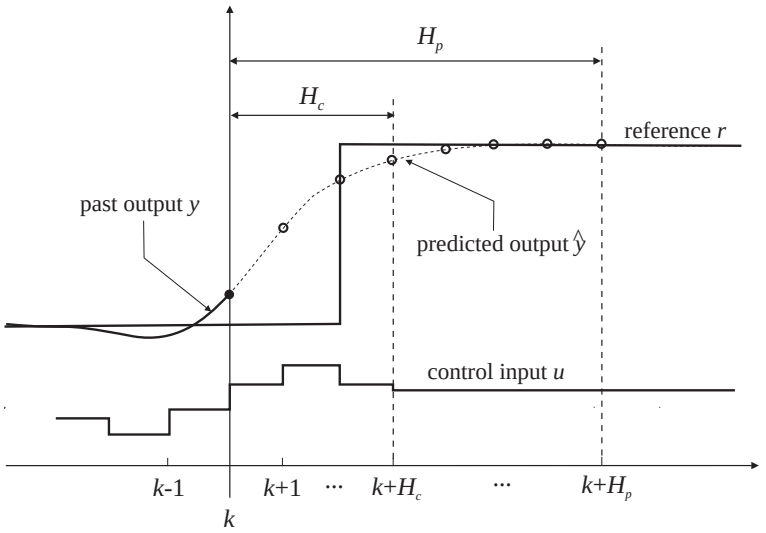

Figure 8 The basic principle of model predictive control.

The sequence of future control signals $u(k+i)$ for $i = 0, \ldots, H_c - 1$ is computed by optimizing a given objective (cost) function, in order to bring and keep the process output as close as possible to the given reference trajectory r, which can be the set-point itself or, more often, some filtered version of it. The most often used objective functions are modifications of the following quadratic function [Clarke *et al.*, 1987]:

$$J = \sum_{i=1}^{H_p} \alpha_i \left(r(k+i) - \hat{y}(k+i)\right)^2 + \sum_{i=1}^{H_c} \beta_i \Delta u(k+i-1)^2 \qquad (3)$$

The first term accounts for minimizing the variance of the process output from the reference, while the second term represents a penalty on the control effort (related for instance to energy). The latter term can also be expressed by using u itself or other filtered forms of u, depending on the problem. The vectors α and β define the weighting of the output error and the control effort with respect to each other and with respect to the prediction step. Constraints, e.g., level and rate constraints of the control input or other process variables can be specified as a part of the optimization

problem. Generally, any other suitable cost function can be used, but for a quadratic cost function, a linear, time-invariant model, and in the absence of constraints, an explicit analytic solution of the above optimization problem can be obtained. Otherwise, numerical (usually iterative) optimization methods must be used. In the more industrially relevant case of a quadratic cost function, a linear, time-invariant model and inequality constraints on $u, \delta u$, and y, the resulting optimization problem is quadratic and must be solved every sample.

Only the control signal $u(k)$ is applied to the process. At the next sampling instant, the process output $y(k+1)$ is available and the optimization and prediction can be repeated with the updated values. This is called the receding horizon principle. The control action $u(k+1)$ computed at time step $k+1$ will be generally different from the one calculated at time step k, since more up-to-date information about the process is available.

MPC fits nicely into the general scheme described in this report. The optimization technique employed is based on sequential unconstrained minimization techniques. The calculations performed in every sample are organized as a sequence of iterations (outer iterations). In every outer iteration a number of inner, Newton iterations are performed. An interesting possibility is to calculate bounds on how much the objective function decreases for each new iteration.

MPC can also be used in other ways. In every sample the control signals for the full control horizon are calculated. Normally only the first one is used, i.e., applied to the process, and the rest are discarded. However, another possibility is to instead store the future values of the control signals and to use them if the execution time is limited and/or the measured values of y correspond well with the ones that were used in optimization step. In this case it is essentially possible to use the MPC in open loop if there is not enough time to even start a new optimization.

8.5 Event-based sampling

The traditional way to design digital control systems is to sample the signals equidistantly in time. Although fixed sampling periods is adequate for many simple control loops, there are a lot of control problems where it is more natural to use varying sampling intervals. One such example is fuel injection engines that are sampled against engine speed, another example is in manufacturing systems, such as paper machines, where sampling is related to production rate. Another reason for using non equidistant sampling is if there is a large cost related to sampling, or if the rate of sampling is governed by unpredictable events. Such examples occur for example in biology and in economics. One motivation for using clever sampling is that the communication resources are limited and no unnecessary signals should be sent in the network.

There are several alternatives to periodic sampling. A basic prerequisite for the implementation of event-triggered sampling is that 'significant events' are detected. The traditional techniques to do this are to use fast polling (sampling) or event-triggered sensors that generates an event whenever the signal passes a certain limit. A typical situation is that the sensor sends a signal whenever the scalar signal $y(t)$ passes any of a certain pre-specified levels in the sampling set $Y = \{y_1, \ldots, y_N\}$. The times

t_k when this happens will be called *sampling times* and are defined by

$$y(t_k) = y_{i_k}$$

where $i_k \in \{1, \ldots, N\}$ determine the *sampling level* at sampling time t_k. Such a scheme has many conceptual advantages. Control is not executed unless it is required, control by exception, see [Kopetz, 1993]. This type of sampling is natural when using many digital sensors such as encoders. A disadvantage is that analysis and design are complicated.

Event-based sampling occurs naturally in many contexts. A common case is in motion control where angles and positions are sensed by encoders that give a pulse whenever a position or an angle has changed by a specific amount. Event based sampling is also a natural approach when actuators with on-off characteristic are used. Satellite control by thrusters is a typical example, [Dodds, 1981]. Systems with pulse frequency modulation, [Polak, 1968], [Pavlidis and Jury, 1965], [Pavlidis, 1966], [Skoog, 1968], [Noges and Frank, 1975], [Skoog and Blankenship, 1970], [Frank, 1979], [Sira-Ramirez, 1989] and [Sira-Ramirez and Lischinsky-Arenas, 1990] are other examples. In this case the control signal is restricted to be a positive or negative pulse of given size. The control actions decide when the pulses should be applied and what sign they should have. Other examples are analog or real neurons whose outputs are pulse trains, see [Mead, 1989] and [DeWeerth *et al.*, 1990]. Analysis of systems with event based sampling are related to general work on discontinuous systems, [Utkin, 1981], [Utkin, 1987], [Tsypkin, 1984] and to work on impulse control, see [Bensoussan and Lions, 1984]. It is also relevant in situations where control complexity has to be weighted against execution time. It also raises other issues such as complexity of control. Control of production processes with buffers is another application area. It is highly desirable to run the processes at constant rates and make as few changes as possible to make sure that buffers are not empty and do not overflow, see [Pettersson, 1969]. Another example is where limited communication resources put hard restrictions on the number of measurement and control actions that can be transmitted.

Much work on systems of this type was done in the period 1960–1980. Analysis of event-based sampled systems is harder than for time-based sampled systems. This is due to the fact that sampling is no longer a linear operation. There are several papers that treat special system setups, such as observers for linear system with quantized outputs, [Sur, 1996], [Delchamps, 1989] many of which use classical ideas from Kalman observer design. To illustrate the techniques consider an output $z(t)$ observed with additive noise. This is usually modeled by

$$y(t) = z(t) + e(t)$$

where the noise $e(t)$ is white with covariance matrix $R(t)$. The covariance matrix $R(t)$ describes the accuracy of the measurements at time t. There are well understood techniques for doing optimal filtering and control in this situation.

The case where a signal is sent exactly when the output $y(t)$ passes a level y_k, can be modeled by setting

$$R^{-1}(t) = \delta(t - t_k),$$

where $\delta(t - t_k)$ is a Dirac function at time t_k. This means that there is zero information about the signal y until it hits the level y_k. With this model the analysis fits the standard mathematical framework and classical observers can be tried. The approach is however suboptimal since there is information also before the signal reaches the new level y_k. Actually, one has the additional information that the signal has NOT reached a new level. Optimal observers that use this information are probably much more complicated.

In [Åström and Bernhardsson, 1999] it is shown that event-based sampling is more efficient than equidistant sampling. For example, an integrator system driven by white noise must be sampled 3–5 times faster using equidistant sampling than using event-based sampling to achieve the same output variance.

Systems with mixed continuous and discrete variables, such as event-triggered sampled systems, are also studied in *hybrid control systems*. This is a rapidly increasing field in control theory. A hybrid control system mixes continuous variables with discrete variable (on/off etc) It is hard to make an overview since there are almost as many approaches as authors in the field. There are several conferences devoted to hybrid control, see for example [Morse, 1995]. There are also many sessions devoted to hybrid control on the large international conferences such as IEEE Conference on Decision and Control (CDC), Automatic Control Conference (ACC) and International Federation of Automatic Controls conferences (IFAC).

Just as event-triggered sampling can be used for deciding suitable measurement times, one can use event-triggering instead of time-triggering for choice of actuator signal. The motivation can be that it costs energy etc, to change the control signal, so such changes should be avoided. The standard linear-quadratic control theory where quadratic weight are put on states and control signals has been extended also with discontinuous costs on changes in control signal. Such a cost can be either fixed or dependent on the size of the control change.

9. WCET analysis

Essentially all real-time scheduling algorithms are based on estimations of the worst-case execution time (WCET) of scheduled tasks. A WCET prediction should provide a prediction that is a tight upper bound of the actual WCET. That is, WCET estimations should be close to, but no lower than, the actual WCET.

The problem of bounding the WCET is twofold:

Hardware-level analysis: determining the WCET (in cycles or nanoseconds) of a piece of object code on a particular hardware platform.

Structure-level analysis: determining the execution time of the longest possible path in a program (or part of it) based on the execution times of individual pieces of the program.

Note that this division resembles the division of compilers into backends and frontends.

Structure-level analysis

The timing schema approach [Shaw, 1989] was the basis for an experimental timing tool targeted at a subset of the C language [Park and Shaw, 1991]. However, in that work the timing schema concept was used as a "reasoning methodology for deterministic timing" rather than as an actual implementation.

In the timing schema approach, a WCET prediction is associated with each "atomic block", where an atomic block is essentially any piece of sequential source code. (The original timing schema approach is concerned with source code rather than object code.) Once timing predictions have been produced for the atomic blocks in a program, predictions can be calculated for composite constructions using their constituents. For example, the execution time of the assignment statement

```
a = b * c;
```

may be described as $T(b)+T(c)+T(*)+T(a)+T(=)$, where $T(X)$ denotes the worst-case execution (or evaluation) time of the node X in the abstract syntax tree.

An important subproblem in structure level analysis is to determine bounds for loop iterations and recursion depths. Either the programmer supplies explicit annotations on loops and recursive calls, or the analysis somehow computes the bounds automatically.

One way to automatically determine some loop bounds (and recursion depths) is to use symbolic execution [Ermedahl and Gustafsson, 1997; Lundqvist and Stenström, 1998; Altenbernd, 1996], that is, to execute the program as usual but allow variables to assume the special value *unknown*. Program input assumes the value *unknown*. Any computation involving an *unknown* value yields the value *unknown*. Whenever execution reaches a branch in control flow that depends on an *unknown* value, both paths are executed. The most important drawback of symbolic execution is that it may never terminate, even if the analyzed program has a bounded WCET.

Another approach is to formulate the WCET of a program as an integer linear programming (ILP) problem [Li *et al.*, 1995]. As indicated below, the approach can be extended to model cache behavior as well. However, the ILP approach is comparatively computing-intensive and unsuitable for interactive analyses.

Hardware level analysis

Modern processors employ a variety of techniques to enhance performance, such as caching, pipelining, and speculative execution. Although these techniques enhance average-case performance, they also make it considerably more difficult to predict the worst-case performance. Assuming the worst case to occur for each instruction can give WCET overestimations by a factor of 10-100 or more. A scheduling based on such pessimistic assumptions effectively wastes the performance offered by modern processors.

Existing hardware level analysis techniques typically operate on object code representations of programs. One approach is the iterative low-level data-flow analysis employed by [White *et al.*, 1997], addressing the effects of data caches, instruction caches, and pipelining. Symbolic execution, as used for structure level analysis, can in some cases be used to predict low-level hardware performance [Lundqvist and Stenström, 1998]. Other

approaches include abstract interpretation [Alt *et al.*, 1996] and extending the ILP approach (mentioned above) [Li *et al.*, 1995].

The original timing schema approach (presented above) does not deal with hardware aspects of WCET analysis. However, the extended timing schema [Lim *et al.*, 1996] is designed to accommodate such hardware aspects. In the extended timing schema approach, the WCET estimations of atomic blocks are not represented by simple integers (i.e. constant times), but rather by a "worst-case timing abstraction" (WCTA). The WCTA data structure for an atomic block b is a representation of how the execution of b would affect the processor state (pipeline and caches).

10. Research problems

Integrated control and scheduling contains a number of interesting research problems. Some of them are outlined here.

10.1 Execution time measurements

A prerequisite for integrated control and scheduling is on-line measurements of the actual task execution times. Several open issues exist. One possibility is to let the task dispatcher be responsible for execution time measurements. Another alternative is to let the tasks themselves perform the time measurements by calling appropriate kernel primitives at vital points in the code. One possibility is that the time measurement code is automatically added to the application task by the programming environment. Another open issue is how to handle interrupts and the execution time needed by the interrupt handler. Should this be included in the execution time of the application tasks or should it be measured separately?

10.2 Mode changes

Mode changes are frequent in adaptive real-time control systems. For instance, every adjustment of task timing attributes in a feedback-scheduling system results in a mode change. A key question is whether the transient effects of mode changes in real-time control systems can be ignored or not. Ignoring the effects gives a simpler mode-change protocol and probably better average-case performance (no unnecessary delays and less overhead). Still, it is probably necessary to have a fall-back strategy and some guarantees about the worst-case behavior. This could perhaps include a calculation of the worst-case delay and an estimate of the resulting loss in control performance.

10.3 Cost functions

Control performance optimization in the context of task attribute adjustments needs cost functions that relate the sampling period with the control performance for each control loop, cost functions that relate the input/output latency with the control performance and, perhaps also, cost functions that relate the execution time for the controller (e.g., the number of iterations in a MPC) with the control performance.

An interesting question is the general nature of these cost functions. For example, it is not always so that faster sampling gives better performance.

For example consider the cost function for a pendulum, described by the following equations:

$$dx = \begin{bmatrix} 0 & 1 \\ \alpha \cdot \omega_0^2 & -d \end{bmatrix} x dt + \begin{bmatrix} 0 \\ \alpha \cdot b \end{bmatrix} u dt + dv_c$$
$$y = [\,1 \quad 0\,]\, x, \qquad R_{1c} = \begin{bmatrix} 0 & 0 \\ 0 & \omega_0^4 \end{bmatrix}$$

The natural frequency is ω_0, the damping $d = 2\zeta\omega_0$, and $b = \omega_0/9.81$. If $\alpha = 1$ the equations describe the pendulum in the upright position (the inverted pendulum), and with $\alpha = -1$ they describe the pendulum in the downward position. The incremental covariance of v_c is R_{1c}, which corresponds to a disturbance on the control signal. The cost for the system is described by a linear quadratic function:

$$J(h) = \frac{1}{h} \int_0^h [\,x^T(t) \quad u^T(t)\,]\, Q \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} dt$$

The cost function, J, for the inverted pendulum as a function of the sampling interval h is shown in Figure 9 (left). The corresponding function for the stable pendulum is shown in Figure 9 (left). Figure 9 (right) clearly

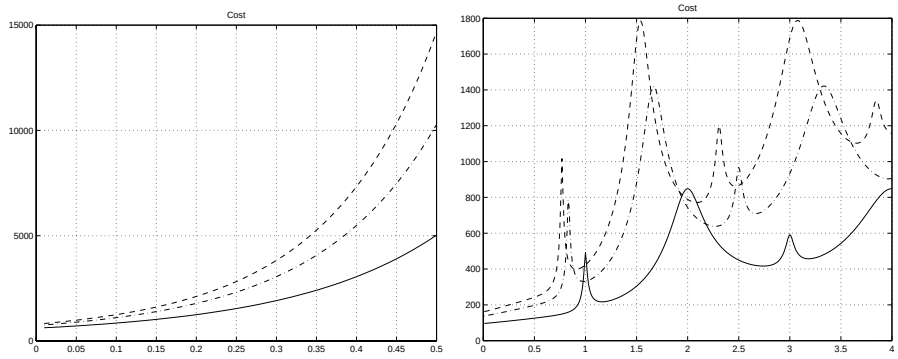

Figure 9 The cost $J_i(h)$ as a function of the sampling interval for the inverted pendulum (left) and for the normal pendulum (right). The plots shows the graphs for $\omega_0 = 3.1416$(full), 3.7699(dot-dashed), and 4.0841(dashed). Note that the cost does not depend monotonously on the sampling period.

demonstrates that faster sampling not necessarily gives better control performance.

Cost functions may be used as criteria for choosing sampling interval. In a setup where several control loops compete for the CPU resources, cost functions can be used as an instrument for portioning resources. To optimize the resource sharing using cost functions, we need to investigate how they depend on the resource, e.g. we need ways of calculating $\frac{dJ}{dh}$.

10.4 Task attribute adjustments

Most work on task attribute adjustments assume that the cost functions at hand have certain properties, e.g., are continuous and monotone. in the control context this is most likely not true. For example, concerning adjustments of the task period it may from a control point of view only be possible to adjust the period in discrete steps. An interesting question is

whether it is possible to state the problem as a global optimization problem, and in that case, what the nature of this optimization is, e.g., convex or non-convex. If it is possible to state the problem as an optimization problem it is also important to find ways of solving this or an approximation of it on-line.

In a given situation it may be possible to reduce the CPU utilization of tasks in different ways, e.g., by increasing the period, reducing the allowed execution time, skipping a sample. An interesting question is to decide which way to use. Also, the possibility of a control task to handle task attribute adjustments is most likely dependent on external conditions, e.g., transient set-point changes, load disturbances, etc. During a set-point change the possibility to reduce the sampling time may be rather limited. However, in many cases the system has the possibility to postpone the set-point change for a short while, and thereby schedule it in a way that affects the other tasks as little as possible.

10.5 Simulation

Much of the work performed by both control engineers and real-time systems engineer are verified by simulation. There are tools available both for simulating different scheduling protocols and other tools for simulating different controller designs. However, the problem of how the implemented controller will actually interact with the plant has been studied very sparsely. A tool that allows the user to verify both the scheduling of the control loops and the control performance is needed. How will, for example, jitter caused by the fact that several control loops share CPU affect the performance of the controllers?

Consider a task set with three controllers. An activation graph of the three controllers when scheduled using the rate-monotonic priority assignment is shown in Fig. 10 Consider the control delay, i.e. the delay from reading an input signal until outputting a new control signal. For the high-priority task it is constant, while for the low-priority task it varies from one to three times the execution time. The interaction of the low-priority task with the plant is shown in Fig. 11.

To investigate the influence of the varying control delay a simulation tool is needed. A MATLAB tool-box that tries to fill this gap is presented in [Eker and Cervin, 1999]. It simulates a real-time kernel executing the control tasks while the plant dynamics may be described by a general Simulink model.

10.6 Interactive WCET analysis

For a WCET analysis tool to become useful in the development of real-time software, it must not only provide WCET predictions — it must also support a reasonable software development model. In particular, it is necessary to allow the developer to manage the system's timing requirements in an interactive manner throughout development.

An interesting possibility would be a WCET analysis tool the system's timing constraints are explicitly expressed by the programmer. This could be based on the use of code annotations on loops and recursive calls provided by the programmer. Such annotations should express the programmer's interpretation of the timing requirements and assumptions about the system. Additional analyses (such as symbolic execution) may be used

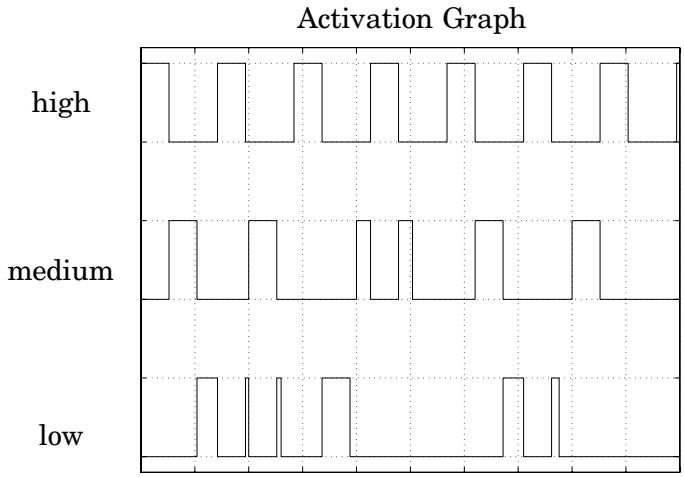

Figure 10 The resulting activation graph for three tasks, with fixed priorities (high, medium, low) running in a pre-emptive kernel. The execution times are the same for all three processes.

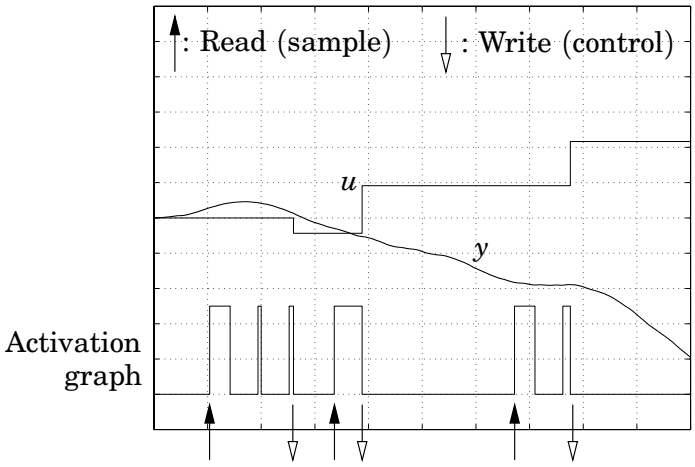

Figure 11 This is how the low priority task from Figure 10 interacts with its plant. (u is the control signal, y is the plant output.)

to verify these annotations, however. Furthermore, the timing predictions should be immediately available throughout development, preferably in an interactive environment.

The timing schema has been presented previously in a form that resembles an attribute grammar (AG) implementation. However, a number of issues complicate an AG implementation of a timing schema. By using an extended AG formalism, reference attributed grammars (RAGs) [Hedin, 1999], timing schemata can be implemented in a concise and modular manner for modern (e.g. object-oriented) languages [Persson and Hedin, 1999].

Although this does not directly address hardware level aspects of WCET analysis, it is important to enable a future integration with such analyses.

A particularly interesting problem in this area is to determine the interface between the two levels of analysis, similar to the interface between compiler backends and frontends. The division typically involves a tradeoff between analysis performance and precision.

A problem related to WCET analysis is that of predicting the execution time overhead of real-time garbage collection. Existing hard real-time garbage collection techniques [Henriksson, 1998; Nilsen, 1994] are based on estimations of the maximum amount of memory in use by the program (live memory). The execution time overhead (and general predictability) of garbage collection depends on the amount of live memory.

11. Summary

Control design and task scheduling are in most cases today treated as two separate issues. The control community generally assumes that the real-time platform used to implement the control system can provide deterministic, fixed sampling periods as needed. The real-time scheduling community, similarly, assumes that all control algorithms can be modeled as periodic tasks with constant periods, hard deadlines, and known worst case execution times. This simple model has made it possible for the control community to focus on its own problem domain without worrying how scheduling is being done and it has released the scheduling community from the need to understand how scheduling delays impact the stability and performance of the plant under control. From a historical perspective, the separated development of control and scheduling theories for computer based control systems has produced many useful results and served its useful purpose.

Upon closer inspection it is, however, quite clear that neither of the above assumptions need necessarily be true. Many of the computing platforms that are commonly used to implement control system, are not able to give any deterministic guarantees. Many control algorithms are not periodic, or they may switch between a number of different fixed sampling periods. Control algorithm deadlines are not always hard. On the contrary, many controllers are quite robust towards variations in sampling period and response time. It is also in many cases possible to compensate on-line for the variations by, e.g., recomputing the controller parameters. Obtaining an accurate value for the WCET is a general problem in the real-time scheduling area. It is not likely that this problem is significantly simpler for control algorithms. It is also possible to consider control systems that are able to do a tradeoff between the available computation time, i.e., how long time the controller may spend calculating the new control signal, and the control loop performance.

For more demanding applications requiring higher degrees of flexibility and for situations where computing resources are limited it is therefore motivated to study more integrated approaches to scheduling of control algorithms. The aim of this report has been to present the state-of-the-art in integrated control and scheduling. Issues that have been discussed include scheduling of control tasks, dynamic task attribute adjustments, feedback scheduling, probabilistic scheduling, and compensation for sampling period variations.

12. References

Abdelzaher, T., E. Atkins, and K. Shin (1997): "QoS negotiation in real-time systems, and its application to flight control." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Abeni, L. and G. Buttazzo (1999): "QoS guarantee using probabilistic deadlines." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*.

Albertos, P. and J. Salt (1990): "Digital regulators redesign with irregular sampling." In *IFAC 11th World Congress*, vol. IV of *IFAC*, pp. 465–469. Tallinn, Estonia.

Alt, M., C. Ferdinand, F. Martin, and R. Wilhelm (1996): "Cache behavior prediction by abstract interpretation." In *Proceedings of SAS '96: International Static Analysis Symposium, LNCS 1145, Springer, 1996*.

Altenbernd, P. (1996): "On the false path problem in hard real-time programs." In *Proceedings of the 8th Euromicro Workshop on Real-Time Systems*.

Åström, K. J. and B. Bernhardsson (1999): "Comparison of periodic and event based sampling for first-order stochastic systems." IFAC World Congress in Beijing.

Atlas, A. and A. Bestavros (1998): "Statistical rate monotonic scheduling." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Audsley, N., A. Burns, R. Davis, K. Tindell, and A. Wellings (1995): "Fixed priority pre-emptive scheduling: An historical perspective." *Journal of Real-Time Systems*, **8**, pp. 173–198.

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

Audsley, N., K. Tindell, and A. Burns (1993b): "The end of the line for static cyclic scheduling." In *Proceedings of 5th Euromicro Workshop on Real-Time Systems*.

Baruah, S. (1998a): "A general model for recurring real-time tasks." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Baruah, S. (1998b): "Overload tolerance for single-processor workloads." In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*. IEEE Computer Society Press.

Baruah, S., D. Chen, S. Gorinsky, and A. Mok (1999): "Generalized multi-frame tasks." Submitted for publication. Available from `www.emba.uvm.edu/~sanjoy`.

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

Baruah, S., D. Chen, and A. Mok (1999): "Static-priority scheduling of multi-frame tasks." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*.

Baruah, S. and S. Gorinsky (1999): "Scheduling periodic task systems to minimize output jitter." Submitted for publication. Available from `www.emba.uvm.edu/~sanjoy`.

Baruah, S. and J. Haritsa (1997): "Scheduling for overload in real-time systems." *IEEE Trans Computers*, **46:9**, pp. 1034–1039.

Bate, I. and A. Burns (1999): "An approach to task attribute assignment for uniprocessor systems." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*.

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

Bernat, G. and A. Burns (1999): "Exact schedulability analysis of aperiodic servers." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*.

Blevins, P. and C. Ramamoorthy (1976): "Aspects of a dynamically adaptive operating system." *IEEE Trans Computers*, **25:7**.

Burns, A. (1998): "The meaning and role of value in scheduling flexible real-time systems." vol. Proceedings of the IEEE Real-Time Computing Systems Application Conference (RTCSA), Hiroshima, Japan.

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

Buttazzo, G., G. Lipari, and L. Abeni (1998): "Elastic task model for adaptive rate control." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Buttazzo, G., M. Spuri, and F. Sensini (1995): "Value vs deadline scheduling in overload conditions." vol. Proceedings of the 16th IEEE Real-Time Systems Symposium.

Caccamo, M. and G. Buttazzo (1997): "Exploiting skips in periodic tasks for enhancing aperiodic responsiveness." In *Proceedings of the 18th IEEE Real-Time System Symposium*.

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

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

Clarke, D., C. Mohtadi, and P. Tuffs (1987): "Generalised predictive control. Part 1: The basic algorithm. Part 2: Extensions and interpretations." *Automatica*, **23:2**, pp. 137–160.

Cottet, F. and L. David (1999): "Time jitter handling in deadline based scheduling of real-time systems." Work in progress.

Davis, R. and A. Wellings (1995): "Dual priority scheduling." In *Proceedings of the IEEE Real-Time Systems Symposium*.

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

DeWeerth, S., L. Nielsen, C. Mead, and K. J. Åström (1990): "A neuron-based pulse servo for motion control." In *IEEE Int. Conference on Robotics and Automation*. Cincinnati, Ohio.

Dodds, S. J. (1981): "Adaptive, high precision, satellite attitude control for microprocessor implementation." *Automatica*, **17:4**, pp. 563–573.

Eker, J. and A. Cervin (1999): "A MATLAB toolbox for real-time and control systems co-design." Submitted to the 6th International IEEE Conference on Real-Time Computing Systems and Applications, Hong-Kong, China.

Ermedahl, A. and J. Gustafsson (1997): "Deriving annotations for tight calculations of execution time." In *Proceedings of EuroPar'97, LNCS 1300, Springer*.

Frank, P. M. (1979): "A continuous-time model for a pfm-controller." *IEEE Trans. of Automat. Control*, **AC-25:5**, pp. 782–784.

Gerber, R. and S. Hong (1993): "Semantics-based compiler transformations for enhanced schedulability." In *Proceedings of the 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**.

Gomaa, H. (1984): "A software design method for real-time systems." In *CACM*, pp. 938–949. Published as CACM, volume 27, number 9.

Gonzalez Härbour, M., 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.

Gupta, D. and P. Jalote (1996): "A formal framework for on-line software version change." *IEEE Trans Software Engineering*, **22:2**.

Gustafsson, K. (1991): "Control theoretic techniques for stepsize selection in explicit Runge-Kutta methods." *ACM Transactions on Mathematical Software*, **17:4**, pp. 533–554.

Gutierrez, J. and M. Harbour (1998): "Schedulability analysis for tasks with static and dynamic offsets." In *Proceedings of 19th IEEE Real-Time Systems Symposium, Madrid*.

Hedin, G. (1999): "Reference attributed grammars." In *Proceedings of WAGA'99: Second Workshop on Attribute Grammars and their Applications*.

Henriksson, R. (1998): *Scheduling Garbage Collection in Embedded Systems*. PhD thesis, Department of Computer Science, Lund University.

Jones, M. and P. Leach (1995): "Modular real-time resource management in the Rialto operating system." In *Proceedings of the of the Fifth Workshop on Hot Topics in Operating Systems*.

Joseph, M. and P. Pandya (1986): “Finding response times in a real-time system.” *The Computer Journal*, **29:5**, pp. 390–395.

Kao, B. and H. Garcia-Molina (1993): “Deadline assignment in a distributed soft real-time system.” In *Proceedings of the 13th International Conference on Distributed Computing Systems*.

Kim, N., M. Ryu, S. Hong, and H. Shin (1999): “Experimental assessment of the period calibration method: A case study.” *Journal of Real-Time Systems*. Accepted for publication.

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

Kleinrock, L. (1970): “A continuum of time-sharing scheduling algorithms.” In *Proc. of AFIPS, SJCC*.

Knudsen, J., M. Lofgren, O. Madsen, and B. Magnusson (1994): *Object-oriented environments: The Mjolner approach*. Prentice Hall.

Kopetz, H. (1993): “Should responsive systems be event triggered or time triggered?” *IEICE Trans. on Information and Systems*, **E76-D:10**, pp. 1525–1532.

Koren, G. and D. Shasha (1995): “Skip-over: Algorithms and complexity for overloaded systems that allow skips.” In *Proceedings of the IEEE Real-Time Systems Symposium*.

Kosugi, N., A. Mitsuzawa, and M. Tokoro (1996): “Importance-based scheduling for predictable real-time systems using MART.” In *Proceedings of the 4th Int. Workshop on Parallel and Distributed Systems*, pp. 95–100. IEEE Computer Society.

Kosugi, N. and S. Moriai (1997): “Dynamic scheduling for real-time threads by period adjustment.” In *Proceedings of the World Congress on Systems Simulation*, pp. 402–406.

Kosugi, N., S. Moriai, and H. Tokuda (1999): “Dynamic real-time scheduling incorporating task timing attribute adjustment.” Work in progress.

Kosugi, N., K. Takashio, and M. Tokoro (1994): “Modification and adjustment of real-time tasks with rate monotonic scheduling algorithm.” In *Proceedings of the 2nd Workshop on Parallel and Distributed Systems*, pp. 98–103.

Kramer, J. and J. Magee (1990): “The evolving philosophers problem: Dynamic change management.” *IEEE Trans Software Engineering*, **16:11**.

Kuo, T.-W. and A. Mok (1991): “Load adjustment in adaptive real-time systems.” In *Proceedings of the 12th IEEE Real-Time Systems Symposium*.

Lee, C., R. Rajkumar, J. Lehoczky, and D. Siewiorek (1998): “Practical solutions for QoS-based resource allocation.” In *Proceedings of the IEEE Real-Time Systems Symposium*.

Lee, C., R. Rajkumar, and C. Mercer (1996): "Experiences with processor reservation and dynamic QoS in real-time Mach." In *Proceedings of Multimedia Japan 96*.

Lehoczky, J. and S. Ramos-Thuel (1992): "An optimal algorithm for scheduling soft aperiodic tasks in fixed-priority preemptive systems." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Lehoczky, J., L. Sha, and J. Strosnider (1987): "Enhanced apriodic responsiveness in hard real-time environment." In *Proceedings of the 8th IEEE Real-Time Systems Symposium*.

Leung, J. Y. T. and J. Whitehead (1982): "On the complexity of fixed-priority scheduling of periodic, real-time tasks." *Performance Evaluation*, **2:4**, pp. 237–250.

Li, Y.-T. S., S. Malik, and A. Wolfe (1995): "Efficient microarchitecture modeling and path analysis for real-time system." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Lim, S.-S., Y. H. Bae, G. T. Jang, B.-D. Rhee, S. L. Min, C. Y. Park, H. Shin, K. Park, and C. S. Kim (1996): "An accurate worst case timing analysis technique for RISC processors." In *Proceedings of the IEEE Real-Time Systems Symposium*.

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

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

Liu, J., K.-J. Lin, W.-K. Shih, A. Yu, J.-Y. Chung, and W. Zhao (1991): "Algorithms for scheduling imprecise computations." *IEEE Trans on Computers*.

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

Locke, C. D. (1992): "Software architecture for hard real-time applications: Cyclic vs. fixed priority executives." *Real-Time Systems*, **4**, pp. 37–53.

Lundqvist, T. and P. Stenström (1998): "Integrating path and timing analysis using instruction-level simulation techniques." In *Proceedings of ACM SIGPLAN Workshop on Languaged, Compilers, and Tools for Embedded Systems*.

Mead, C. A. (1989): *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, Massachusetts.

Mok, A. and D. Chen (1997): "A multi-frame model for real-time tasks." *IEEE Transactions on Software Engineering*, **23:10**.

Mok, A. and G. Liu (1997): "Early detection of timing constraint violation at runtime." In *Proceedings of the IEEE Real-Time Systems Symposium*.

Morse, A. S., Ed. (1995): *Control Using Logic-Based Switching*, Block Island Workshop.

Nakajima, T. (1998): "Resource reservation for adaptive QoS mapping in real-time Mach." In *Proceedings of the Sixth International Workshop on Parallel and Distributed Real-Time Systems*.

Nakajima, T. and H. Tezuka (1994): "A continuous media application supporting dynamic QoS control on real-time Mach." In *Proceedings of ACM Multimedia'94*.

Nilsen, K. (1994): "Reliable real-time garbage collection of C++." *Computing Systems*.

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

Nilsson, J., B. Bernhardsson, and B. Wittenmark (1998a): "Some topics in real-time control." In *American Control Conference*. Philadelphia.

Nilsson, J., B. Bernhardsson, and B. Wittenmark (1998b): "Stochastic analysis and control of real-time systems with random time delays." *Automatica*, pp. 57–64.

Noges, E. and P. M. Frank (1975): *Pulsfrequenzmodulierte Regelungssysteme*. R. Oldenbourg, München.

Park, C. Y. and A. C. Shaw (1991): "Experiments with a program timing tool based on source-level timing schema." *IEEE Computer*, **24:5**.

Pavlidis, T. (1966): "Optimal control of pulse frequency modulated systems." *IEEE Trans. of Automat. Control*, **AC-11:4**, pp. 35–43.

Pavlidis, T. and E. J. Jury (1965): "Analysis of a new class of pulse frequency modulated control systems." *IEEE Trans. of Automat. Control*, **AC-10**, pp. 35–43.

Pedro, P. and A. Burns (1998): "Schedulability analysis for mode changes in flexible real-time systems." In *Proceedings of the 10th Euromicro Workshop on Real-Time Systems*.

Persson, P. and G. Hedin (1999): "Interactive execution time predictions using reference attributed grammars." In *Proceedings of WAGA'99: Second Workshop on Attribute Grammars and their Applications*.

Pettersson, B. (1969): "Production control of a complex integrated pulp and paper mill." *Tappi*, **52:11**, pp. 2155–2159.

Polak, E. (1968): "Stability and graphical analysis of first order of pulse-width-modulated sampled-data regulator systems." *IRE Trans. Automatic Control*, **AC-6:3**, pp. 276–282.

Potier, D., E. Gelenbe, and J. Lenfant (1976): "Adaptive allocation of central processing unit quanta." *Journal of ACM*, **23:1**.

Rajkumar, R., C. Lee, J. Lehoczky, and D. Siewiorek (1997): "A resources allocation model for QoS management." In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*.

Ramanathan, P. (1997): “Graceful degradation in real-time control application using (m,k)-firm guarantee.” In *Proceedings of the IEEE Real-Time Systems Symposium*.

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. and S. Hong (1999): “A period assignment algorithm for real-time system design.” In *Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99). Amsterdam, the Netherlands*.

Ryu, M., S. Hong, and M. Saksena (1997): “Streamlining real-time controller design: From performance specifications to end-to-end timing constraints.” In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*.

Sanden, B. (1994): *Software Systems Construction with Example in Ada*. Prentice Hall International.

Seto, D., B. Krogh, L. Sha, and A. Chutinan (1998a): “Dynamic control system upgrade using the Simplex architecture.” *IEEE Control Systems*, August.

Seto, D., J. Lehoczky, and L. Sha (1998b): “Task period selection and schedulability in real-time systems.” In *Proceedings of the IEEE Real-Time Systems Symposium*.

Seto, D., J. Lehoczky, L. Sha, and K. Shin (1996): “On task schedulability in real-time control systems.” In *Proceedings of the IEEE Real-Time Systems Symposium*.

Sha, L. (1998): “Dependable system upgrade.” In *Proceedings of IEEE Real Time Systems Symposium*.

Sha, L., R. Rajkumar, and J. Lehoczky (1989): “Mode change protocols for priority-driven pre-emptive scheduling.” *Real-Time Systems*, **1:3**, pp. 244–264.

Sha, L., R. Rajkumar, and J. Lehoczy (1990): “Priority inheritance protocols: An approach to real-time synchronization.” *IEEE Trans on Computers*, **39:9**.

Sha, L., R. Rajkumar, and S. Sathaye (1994): “Generalized rate-monotonic scheduling theory: A framework for developing real-time systems.” *Proceedings of the IEEE*, **82:1**.

Shasha, D. and G. Koren (1995): “D-over: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems.” *Siam Journal of Computing*, **24:2**, pp. 318–339.

Shaw, A. C. (1989): “Reasoning about time in higher-level language software.” *IEEE Transactions on Software Engineering*, **15:7**.

Shin, K. G. and C. L. Meissner (1999): “Adaptation of control system performance by task reallocation and period modification.” In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 29–36.

Sira-Ramirez, H. (1989): “A geometric approach to pulse-width modulated control in nonlinear dynamical systems.” *IEEE Trans. of Automat. Control*, **AC-34:2**, pp. 184–187.

Sira-Ramirez, H. and P. Lischinsky-Arenas (1990): “Dynamic discontinuous feedback control of nonlinear systems.” *IEEE Trans. of Automat. Control*, **AC-35:12**, pp. 1373–1378.

Skoog, R. A. (1968): “On the stability of pulse-width-modulated feedback systems.” *IEEE Trans. of Automat. Control*, **AC-13:5**, pp. 532–538.

Skoog, R. A. and G. L. Blankenship (1970): “Generalized pulse-modulated feedback systems: Norms, gains, lipschitz constants and stability.” *IEEE Trans. of Automat. Control*, **AC-15:3**, pp. 300–315.

Sprunt, B., L. Sha, and J. Lehoczky (1989): “Aperiodic task scheduling for hard real-time systems.” *The Journal of Real-Time Systems*.

Spuri, M. and G. Buttazzo (1996): “Scheduling aperiodic tasks in dynamic priority systems.” **10:2**, pp. 179–210.

Stankovic, J., C. Lu, and S. Son (1999): “The case for feedback control real-time scheduling.” In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 11–20.

Stankovic, J. and K. Ramamritham (1991): “The Spring kernel: A new paradigm for real-time systems.” *IEEE Software*, **8**.

Stewart, D., R. Volpe, and P. Khosla (1993): “Design of dynamically reconfigurable real-time software using port-based objects.” Technical Report CMU-R1-TR-93-11. CMU.

Sun, J., R. Bettati, and J. W.-S. Liu (1994): “An end-to-end approach to schedule tasks with shared resources in multiprocessor systems.” In *Proceedings of the 11th IEEE Workshop on Real-Time Operating Systems and Software*.

Sur, J. (1996): *State Observers for Linear Systems with Quantized Outputs*. PhD thesis, University of Santa Barbara.

Tia, T.-S., Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W.-S. Liu (1995): “Probabilistic performance guarantee for real-time tasks with varying computation times.” vol. Proceedings of the IEEE Real-Time Technology and Applications Symposium. Chicago, IL.

Tindell, K., A. Burns, and A. J. Wellings (1992): “Mode changes in priority preemptively scheduled systems.” In *Proceedings of the 13th IEEE Real-Time Systems Symposium*, pp. 100–109.

Tindell, K., A. Burns, and A. J. Wellings (1994): “An extendible approach for analyzing fixed priority hard real-time tasks.” *Real-Time Systems*, **6:2**, pp. 133–151.

Tokuda, H., T. Nakajima, and P. Rao (1990): “Real-time Mach: Towards a predictable real-time kernel.” vol. Proceedings of USENIX Mach Workshop.

Törngren, M. (1998): “Fundamentals of implementing real-time control applications in distributed computer systems.” *Real-time systems*, **14:3**.

Tsypkin, Ya. Z. (1984): *Relay Control Systems*. Cambridge University Press, Cambridge, UK.

Utkin, V. (1981): *Sliding modes and their applications in variable structure systems*. MIR, Moscow.

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

Ward, O. T. and S. J. Mellor (1985): *Structured Development for Real-Time Systems*, vol. Four volumes. Yourdon Press.

White, R. T., F. Mueller, C. A. Healy, D. B. Whalley, and M. G. Harmon (1997): "Timing analysis for data caches and set-associative caches." In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*.

Wittenmark, B. and K. J. Åström (1980): "Simple self-tuning controllers." In Unbehauen, Ed., *Methods and Applications in Adaptive Control*, number 24 in Lecture Notes in Control and Information Sciences, pp. 21–29. Springer-Verlag, Berlin, FRG.

Xu, J. and D. Parnas (1990): "Scheduling processes with release times, deadlines, precedence, and exclusion relations." *IEEE Trans Software Engineering*, **16:3**, pp. 360–369.