Resource-Constrained Embedded Control Systems: Possibilities and Research Issues

Årzén, Karl-Erik; Cervin, Anton; Henriksson, Dan

Published in:
Proceedings of CERTS'03 – Co-Design of Embedded Real-Time Systems Workshop

2003

Resource-Constrained Embedded Control Systems: Possibilities and Research Issues

Karl-Erik Årzén, Anton Cervin, Dan Henriksson

Department of Automatic Control
Lund Institute of Technology
Box 118, SE-221 00 Lund, Sweden
(karlerik|anton|dan)@control.lth.se

Abstract

A survey that points out research issues and open problems in the area of integrated control and real-time scheduling. Issues that are discussed include temporal robustness, schedulability margin, optimal and direct feedback scheduling, quality-of-control, and tools.

1. Introduction

The pervasive/ubiquitous computing trend has increased the emphasis on embedded computing within the computer engineering community. Already today embedded computers by far outnumber desktop computers. Control systems constitute an important subclass of embedded computing systems. For example, within automotive systems computers are commonly denoted as electronic control units (ECU). A top-level modern car contains more than 50 ECUs of varying complexity. A majority of these implement different feedback control tasks, e.g., engine control, traction control, anti-lock braking, active stability control, cruise control, and climate control.

Embedded systems are often found in mass-market products and are therefore subject to hard economic constraints. The pervasive nature of the systems generate further constraints on physical size and power consumption. These product-level constraints give rise to resource constraints on the computing platform level, e.g., constraints on computing speed, memory size, and communication bandwidth. Due to the economic constraints this is true in spite of the fast development of computing hardware. In most cases it is not economically justified to use a processor with more capacity, and hence that is more expensive, than what is required by the application. The economical constraints also favor general-purpose computing components over specially designed hardware solutions.

The resource constraints increase the need for co-design. Co-design is needed at several levels. One example, is hardware and software co-design. Which system functions should be implemented in hardware and which should be implemented in software? The possibility to use programmable hardware, e.g., FPGAs, further increases the design complexity. Another example is the co-design of the mechanical design and the electrical design. The aim of this paper, however, is the co-design of the control system and computing system, with a special emphasis on integration of control and real-time scheduling.

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

The aim of this paper is to highlight important principles and unsolved research questions within the area of integrated control and real-time scheduling. Issues that will be discussed include temporal robustness, schedulability margins, quality-of-control, feedback scheduling, and co-design tools.

2. Temporal determinism

Computer-based control theory in most cases assumes equidistant sampling and negligible, or constant, input-output latencies. However, this can seldom be achieved in practice, or is too costly for the particular application. In a multi-threaded system tasks interfere with each other due to preemption and blocking due to task communication. Execution times may be data-dependent or vary due to, e.g., the use of caches. For distributed systems with networked control loops where the sensors, controllers, and actuators reside on different physical nodes, the communication gives rise to latencies that can be more or less deterministic depending on the network protocols used. The result of all this is jitter in sampling intervals and non-negligible and varying latencies. The resulting temporal non-determinism can be approached in two different ways. The *hard real-time approach* strives to maximize the temporal determin-

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

The second, *soft* or *control-based*, approach instead views the temporal nondeterminism caused by the implementation platform as an uncertainty or disturbance acting on the control loop, and handles it using control-based approaches. Some example of techniques that can be applied are temporally robust design methods and measurement-based active compensation. The latter can be compared to traditional gain-scheduling and feed-forward from disturbances. In order to apply these techniques it is necessary to increase the understanding of how temporal nondeterminism affects control performance. This requires new theory and tools that now gradually is beginning to emerge. It is somewhat surprising, though, that the large robust control community not yet has focused on temporal robustness. A large amount of general theory and design methods have been developed. However, almost everything is developed for plant uncertainties, i.e., parametric or frequency-dependent uncertainties. Although parts of this carries over to temporal robustness it is likely that there is room for much more research here. The approach also requires language and/or operating support for instrumenting an application with measurement code.

An important issue that still is lacking is theory that allows us to determine which level of temporal determinism that a given control loop really requires in order to meet given control objectives on stability and performance. Is it necessary to use a time-triggered approach or will an event-based approach perform satisfactorily? How large input-output latencies can be tolerated? Is it OK to now and then skip a sample in order to maintain the schedulability of the task set? Ideally one would like to have an index that decides the required level of temporal determinism through a single quantitative measure. One possible name for such an index would be the *schedulability margin*. This measure would need to combine both a margin with respect to input-output latencies and a margin that decides how large sampling jitter the loop can tolerate. For constant input-output latencies the classical phase margin can be applied. The phase margin is based on a graphical frequency-domain representation. Recently new theory has been developed that uses the same graphical Bode-diagram representation, but which applies to systems with varying latencies [Lincoln, 2002b]. The stability criterion is based on the small gain theorem. The same theory can also be used to design dynamic latency compensation schemes [Lincoln, 2002a]. The approach assumes that the actual latencies can be measured and that a high-frequency model of the process is available. It does, however, not require any latency statistics information.

What is still missing in order to be able to define a reasonable analytical concept for a schedulability margin is a simple sampling jitter criterion. The criterion should ideally tell how large variations around a nominal sampling interval that the process could tolerate and still remain stable, alternatively maintain acceptable performance.

3. Feedback Scheduling

The objective of feedback scheduling is to increase flexibility and to master uncertainty with respect to resource allocation. Instead of pre-allocating resources based of off-line analysis the resources are allocated dynamically on-line, based on feedback from the actual resource utilization. In general the resources can be any computational resources. Here, we will however concentrate on the scheduling of the execution of real-time tasks, and in particular of the execution of real-time controller tasks.

3.1 Optimal Feedback Scheduling

Most of the suggested approaches to feedback scheduling have been more or less ad-hoc. Typically the aim has been to adjust sampling periods or execution time demands in such a way that the task set becomes schedulable or that the deadline miss ratio is at an acceptable level [Lu *et al.*, 2002].

However, in order for feedback scheduling to really become a realistic alternative it is necessary to take the application performance into account. For example to adjust the scheduling parameters in such a way that the global performance is optimized. To do this it is necessary to have performance metrics that are parameterized in terms of sampling intervals, latencies, and the jitter in these. When the applications are control loops there are certain possibilities do this. However, for more general applications this might not be so easy. In [Cervin, 2003] it was shown that a simple linear proportional rescaling of the nominal sampling periods in order to meet the utilization set-point is optimal with respect to the overall control performance under certain assumptions. It holds if the control cost functions $J_i(h_i)$, where h_i is the sampling period, are quadratic, i.e.,

$$J_i(h_i) = \alpha_i + \beta_i h_i^2$$

or if they are linear,

$$J_i(h_i) = \alpha_i + \gamma_i h_i,$$

and if the objective of the feedback scheduler is to minimize the sum of the control cost functions or a weighted sum of

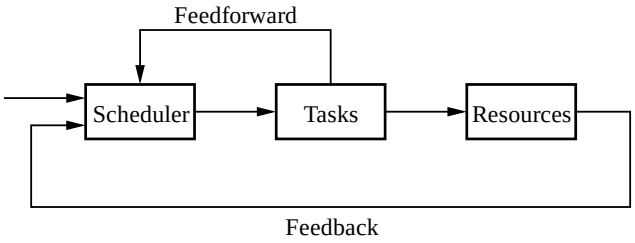

Figure 1 A general feedback scheduling structure. The resources are distributed among the tasks based on feedback from the actual resource use. The tasks can be use feedforward to notify the scheduler about changes in their resource demands.

the control cost functions. The advantage of this is a simple and fast calculation that easily can be applied on-line. The linear rescaling also has the advantage that it preserves the rate-monotonic ordering of the tasks and, thus, avoids any changes in task priorities in the case that fixed priority scheduling is used. Linear or quadratic cost functions are also quite good approximations of true cost functions in many cases. It is also possible to add more constraints to the optimization problem and still retain a simple solution. For example, one can use the nominal sampling periods as minimal sampling periods and use these whenever the utilization is less than the utilization set-point. However, the linear rescaling property does not hold in all cases. If the task set includes both tasks with quadratic cost functions and tasks with linear cost functions, the solution is not as simple, although it is still computable.

It is also possible to assign maximum sampling periods to certain tasks. This leads however to an iterative computation (LP-problem) in order to find the total rescaling of all the tasks. This is equivalent to the calculations needed in the elastic task model [Buttazzo *et al.*, 1998] when the tasks (springs) have constraints on how much they may be compressed. However, it should be noted that the the cost functions above only concern the task periods and not the input-output latencies.

3.2 Feedback Scheduling Structures

Different structures are possible in feedback scheduling. A pure feedback scheme is reactive in the sense that the feedback scheduler will only remove a utilization error once it is already present. By combining the feedback with feedforward a pro-active scheme is obtained. The feedforward path could be use to allow controller task to inform the scheduler that they are changing their desired amount of resources, e.g., changing their execution times or nominal sampling periods, and to give the scheduler the possibility to compensate for this before any overload has occurred. The feedforward path can be also be used for dynamic task admission. A block diagram of the feedback-feedforward structure is shown in Fig. 1.

It is also possible to consider a layered or cascaded control structure. The outer layer would consist of a feedback scheduler that, based on a desired set-point for the overall utilization, generates as outputs the desired utilization for each controller task. Associated with each controller task is then a local feedback scheduler that is responsible for adjusting the timing parameters of the task in order to fulfill the desired utilization. The utilization assigned to each controller task can be viewed as its share of the total resource, e.g., the total CPU capacity. This approach can be combined with reservation-based scheduling in order to provide temporal protection for the individual tasks [Mercer *et al.*, 1993]. Each task can then be seen as if it is executing on its own virtual CPU. One example of a reservation-based scheduling scheme is the constant-bandwidth server (CBS) [Abeni and Buttazzo, 1998]. Analysis of a reservation-based feedback scheduler is presented in [Abeni *et al.*, 2002]. The control server computational model for controller tasks is based on reservation-based scheduling and feedback scheduling [Cervin and Eker, 2003].

3.3 Direct Feedback Scheduling

Most of the feedback scheduling approaches proposed for control applications are indirect. By adjusting the task parameters, e.g., period and execution time, one makes sure that the task set is schedulable and has certain timing properties (latencies and jitter). These timing properties will then indirectly determine the performance of the application. The problem with this is the relationship between the timing parameters and the cost/performance. In most cases the relationship only holds in stationarity and in a mean-value sense. In direct feedback scheduling the idea is to base the decision of which task to execute on the instantaneous cost. This cost will grow the longer the control loop executes in open loop and decrease when a control action is issued. The instantaneous cost could then be used as a dynamic priority similar to the task deadline in EDF. One example of how this approach could be implemented is the following. Each controller consists of two parts. The first part contains the sampling of the measurement signal and evaluation of the instantaneous cost function. The second part is the actual control algorithm, which then would be optional. The total control system would execute at a constant short sampling period. The first part could be implemented by using interrupt handlers and be executed for all the controller tasks in the beginning of each sampling period. The scheduler would then select and execute the controller part of the control task with the largest instantaneous cost.

The resulting system would be a special case of an aperiodic event-triggered sampled system. Although time-triggered sampling is adequate for many simple control loops, there are a lot of control problems where it is more natural to use event-triggered sampling, e.g., control of combustion engines. Another 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 one typical example, [Dodds,

1981]. Systems with pulse frequency modulation [Skoog and Blankenship, 1970], [Sira-Ramirez, 1989], and analog or real neurons whose outputs are pulse trains, see [Mead, 1989] and [DeWeerth *et al.*, 1990] are other examples.

Analysis of systems with event based sampling is related to general work on discontinuous systems, [Utkin, 1981], [Utkin, 1987], [Tsypkin, 1984] and to work on impulse control, see [Bensoussan and Lions, 1984]. Much work on systems of this type was done in the period 1960–1980. Analysis of event-based sampled systems is considerably harder than for time-based sampled systems. This is due to the fact that sampling is no longer a linear operation. There are several papers that treat special system setups, such as observers for linear system with quantized outputs, [Sur, 1996], [Delchamps, 1989] many of which use classical ideas from Kalman observer design. In [Åström and Bernhardsson, 1999] it is shown that event-based sampling can be more efficient than equidistant sampling. 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. However, we are still very far from a general theory for aperiodic event-triggered sampled systems.

In spite of the lack of theory it is possible to derive different heuristic versions of direct feedback scheduling. A question is then how the instantaneous cost function should look like. It would be quite natural to include the controller error in the function. The larger the error the more critical the loop is in general. One could also consider including the error derivative. The motivation for this would be to be able to judge the decision whether to execute a controller on a prediction of the error rather than on the actual error. A loop with a large but decreasing error would be less urgent than a loop with an increasing error of the same magnitude. One could also consider the past history of the error signal, i.e., include an integral term in the cost. One possibility would be to judge the decision of which controller to execute on a performance measure such as the IAE (Integrated Absolute Error) or the ISE (Integrated Square Error), possibly in combination with some forgetting factor. Interestingly enough an instantaneous cost function of this kind shows strong similarities with the well-known PID controller. Another useful term to add to the function would be a term that increases the longer the loop has been running in open loop.

3.4 Scheduling Overhead

In order for feedback scheduling to be practically useful it is crucial that the overhead associated with the feedback scheduler itself is small compared to the dynamics and the time intervals of the task set that is being controlled. Hence, simple techniques such as linear rescaling is preferable over methods involving more complex calculations.

Changing the task parameters in an indirect feedback scheduling scheme gives rise to a mode change transient. Although the task set may be schedulable both before and after the mode switch, it is not at all sure that all task deadlines are met during the transient. The necessary analysis in order to guarantee this is still not completely worked out, e.g., [Tindell *et al.*, 1992], [Pedro and Burns, 1998], and [Buttazzo *et al.*, 1998].

3.5 Anytime Controllers

A feedback scheduler can control the utilization of task set by either changing the task periods or by changing the task execution times. For controller tasks the first alternative is the most natural. However, there are examples when also the execution times can be changed, e.g., Model-Based Predictive Controllers (MPC), see e.g. [Garcia *et al.*, 1989; Richalet, 1993]. In an MPC, the control signal is determined by on-line optimization of a cost function in every sample. The optimization problem is solved iteratively, with highly varying execution time depending on a number of factors: the state of the plant, the current and future reference values, the disturbances acting on the plant, the number of active constraints on control signals and outputs, etc. For fast processes with dominating time constants in the same order as the execution time, the execution time also gives rise to an input-output latency that can effect the control performance considerably. The MPC strategy has won widespread industrial use in recent years, the main advantages being its ability to handle constraints and its straightforward applicability to large, multi-variable processes. However, because of the computational demands of the control algorithm, MPC has traditionally only been applied to plants in the process industry, with slow dynamics and low requirements on fast sampling. The industrial practice has been to run the MPC algorithm on a dedicated computer, and to decrease the complexity of the problem so that overruns are avoided.

In the terminology of [Liu *et al.*, 1991] MPCs can be viewed as anytime algorithms of the “milestone” task type. In each sample, the quality of the control signal is gradually refined for each iteration in the optimization algorithm, up to a certain bound. This makes it possible to abort the optimization before it has reached the optimum, and still obtain an acceptable control signal. Another nice feature of MPC is that it is not only *possible* to extract a real-world quality-or-service or cost measure from the controller, but the control algorithm is indeed *based* on the same measure. This enables a tight and natural connection between the control and the scheduling. MPC controllers also fit nicely with the imprecise computation model [Chung *et al.*, 1990]. Each MPC task has a mandatory part that consists of a search for a feasible solution that fulfills all the constraints, and one optional part that is the actual optimization, i.e., the gradual refinement of the feasible solution. The use of feedback scheduling for MPC controllers is reported in [Henriksson *et al.*, 2002b; Henriksson *et al.*, 2002a].

Another area where the task execution time may be varied is in visual servoing. A camera-based vision sensor can in a certain sense be viewed as an anytime sensor, especially

if the visual feedback is based on the extraction of image feature points. The more time available for the sensing the better estimates of the feature points can be derived.

3.6 Quality of Control

When applying feedback scheduling the control performance can be viewed as a quality parameter similar to the quality-of-service parameters used within multimedia and communication systems. Several important issues require attention. One issue is how performance specifications should be represented. Instead of specifying absolute values of different performance parameters, e.g., overshoot, steady state variance, etc, the designer needs to specify acceptable ranges of these values. An interesting issue is how these specifications should be expressed. One possibility is to use a minimum-maximum interval, i.e., in essence specify that the actual value of the performance metric should be uniformly distributed. Another possibility is to use general distributions.

Another important issue is how the run-time resource negotiation should be expressed. The exact nature of which negotiation scheme that is most appropriate is still open. One possibility is to use contracts that specifies how the control loop performance depends on the assigned resource level. Another possibility is to apply the Broker architectural software pattern.

4. Co-design Tools

In order for integrated control and real-time scheduling to be a reality it is necessary to have computers tools for simulation, analysis, and synthesis. During recent years a few such tools have emerged, e.g., the RTSIM scheduling simulator extended with a numerical simulation module [Casile *et al.*, 1998], the Ptolemy system with its recently included timed multitasking domain [Liu and Lee, 2003], and the simulation tool presented in [El-khoury and Törngren, 2001]. Here we will focus on the two Matlab/Simulink tools that have been developed in our group, Jitterbug and TrueTime [Cervin *et al.*, 2003].

4.1 Jitterbug

Jitterbug [Lincoln and Cervin, 2002] makes it possible to analyze the impact of latencies, jitter, lost samples, aborted computations, etc on controller performance. The tool can also be used investigate jitter-compensating, aperiodic, and multi-rate controllers. The basis of Jitterbug is the calculation of a quadratic performance criterion. The main contribution of Jitterbug is the packaging that it provides of the theory for linear quadratic Gaussian systems and jump-linear systems in a user-accessible way and on a format that suits the analysis of controller timing issues. However, the tool also has a number of limitations that it is important to be aware of.

Linear Systems: Jitterbug only applies to linear systems. Although linear theory often provides a very good approximation of non-linear systems there are a lot of situations when non-linear issues are important. For example, all actuators have limited range, i.e., they saturate. During actuator saturation the control loop is effectively cut-off and the controlled process runs in open loop. In order to avoid that unstable controller states, e.g., the integrator state, explode (wind-up) during saturations all practical controllers must be equipped with an anti-windup scheme. This can not be analyzed using Jitterbug. The fact that Jitterbug is not applicable to non-linear systems is, however, not surprising. Non-linear discrete-time systems is a very undeveloped field. For example, there is not even a commonly accepted sampled-time representation of general non-linear systems yet.

Stationarity: The quadratic cost calculated by Jitterbug is a measure of the controller performance during stationarity. This is well suited for regulatory control systems where the objective is to keep the controlled variables at constant set-point values during the presence of stochastic noise. However, for servo-control systems where the main objective is tracking of non-constant set-point signals and rejection of deterministic disturbances it is the performance during transient conditions, e.g. overshoot and rise time, that is most important. Although this is closely related to the type of performance measures that Jitterbug calculates, Jitterbug is in general not ideally suited for these types of control problems.

Statistical measure: The output of Jitterbug is a statistical measure, i.e., an expected value. Latencies and jitter are modeled using statistical distributions. A result of this is that Jitterbug can never be used to formally prove that, e.g., the cost function for a certain timing scenario in actual case will have a certain result. The results only hold in a mean-value sense. Another effect of the statistical nature of Jitterbug is that timing situations that have probability zero will be disregarded in the analysis. A case where this can be important is for systems with switching-induced instability. Consider the following example from [Schinkel *et al.*, 2002]. The process to be controlled is modeled by

$$\dot{x} = Ax + Bu$$

where

$$A = \begin{bmatrix} 0 & 1 \\ -10000 & -0.1 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The system is stable with poles in $p_{1,2} = -0.05 \pm 100i$. The process has been discretized with $h_1 = 0.002s$ and $h_2 = 0.094s$. The two discrete-time systems are represented by

$$x_{k+1} = \Phi_i x_k + \Gamma_i u_y$$
$$i \in \{1, 2\}$$

where $\Phi_i = e^{Ah_i}$, $\Gamma_1 = \int_0^{h_i} e^{As} B ds$. Both discretizations lead to stable discrete systems with the spectral radius $\rho(\Phi_1) \leq$

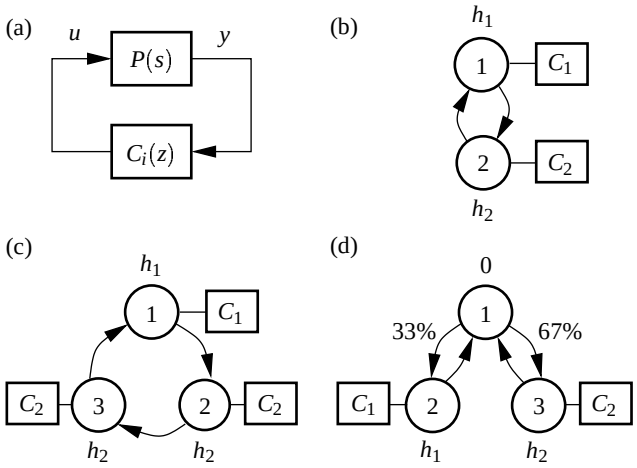

Figure 2 Jitterbug model of system with varying sampling intervals: (a) signal model, (b) timing model with repeating intervals h_1, h_2, (c) timing model with repeating intervals h_1, h_2, h_2, and (d) timing model with random sequence of sampling intervals.

$1, \rho(\Phi_2) \leq 1$, where $\rho(\Phi_i)$ gives the largest eigenvalue of Φ_i. For each discrete-time system a LQ-controller has been designed to minimize the continuous-time cost function

$$J = \lim_{t\to\infty} \frac{1}{t} \int_0^T \left(x^T(s) Q_1 x(s) + u^T(s) Q_2 u(s) \right) ds$$

where

$$Q_1 = \begin{pmatrix} 20000 & 0 \\ 0 & 20000 \end{pmatrix}, \quad Q_2 = 50$$

The resulting state feedback law is represented by $u = -K_i x$.

By looking upon the spectral radius for $\Phi_i - \Gamma_i K_i$ it is easy to verify that the closed loop system is stable for both sampling intervals. However, for the repeating switching cycle $h_1 h_2 h_2$ the system is unstable. This can be verified by looking upon the spectral radius of the resulting system $\rho\left((\Phi_2 - \Gamma_2 K_2)^2 (\Phi_1 - \Gamma_1 K_1)^1\right) \geq 1$.

Using Jitterbug it can easily be verified that the closed loop system is stable for both the two sampling intervals. It is also possible to use Jitterbug to verify that the switching sequence $h_1 h_2 h_2$ is stable. However, if it not known beforehand that this sequence yields an unstable system, one may easily be fooled. For example, if the sampling interval is modeled as a two-point distribution with 33% probability for h_1 and 67% probability for h_2 then Jitterbug gives a result that indicates that the closed loop system is stable. Using a feedback scheduler that dynamically adjusts the sampling periods it is not impossible that a situation like this could arise.

The Jitterbug models for different timing sequences situations are shown in Fig. 2. The simulated time sequences for the switching sequence $h_1 h_2 h_2$ is shown in Fig. 3 and for the random two-point distribution is shown in Fig. 4.

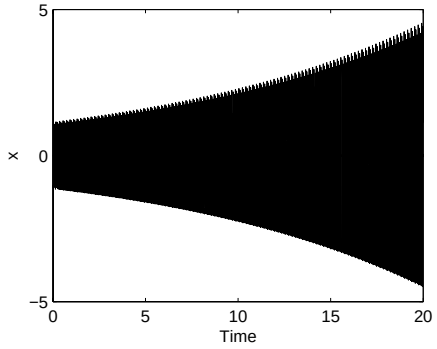

Figure 3 Simulation of with switching sequence $h_1 h_2 h_2$. The result is an unstable system.

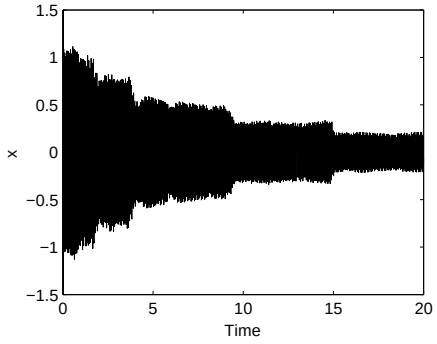

Figure 4 Simulation of with a random two-point distribution. The result is a stable system.

4.2 TrueTime

TrueTime, [Henriksson *et al.*, 2002c], allows co-simulation of distributed computer-based control loops and the controlled plant. This is achieved by simulating the temporal behavior of multitasking real-time kernels and network protocols in Matlab/Simulink. TrueTime was primarily developed as a co-design tool. However, TrueTime can also be used as an experimental platform for research on feedback scheduling. Using TrueTime it is possible to implement user-defined scheduling policies, it supports deadline overrun and execution-time overrun handling, measurements of execution time is straightforward, and the application tasks can be interfaced with the kernel level. Used in this way TrueTime can be used to evaluate and test different new real-time kernel features before they are implemented for real. The possibility to run TrueTime in real-time makes this even more interesting. Interfacing the computer to a real process using A-D and D-A converters the setup can be used to emulate a slow, multitasking computer controlling a real plant.

Extensions: Although TrueTime is a quite powerful already today there are certain areas that could be extended. The network block only supports data-link protocols and only a single network segment may be present. In current work we are extending this by also implementing transport layer protocols, e.g., TCP with acknowledgment messages, buffering,

congestion control, and flow control. We also allow multiple network segments within the same model. This is necessary in order to be able to model networked control loops over switched Ethernet, a technology that is becoming increasingly popular for real-time communication. Another area where TrueTime needs improvement is execution time estimation. In TrueTime it is the user that assigns the time it takes to calculate every code segment or interrupt handler. This time should ideally match the actual time that it would take to execute the equivalent code on the target platform under consideration. However, assigning these times can in practice be very difficult. It would be interesting to combine TrueTime with an execution time analysis tool. However, exactly how that should be done is still unclear.

5. Conclusions

Control systems constitute an important subclass of embedded real-time systems. Control systems have traditionally been relatively static systems. However, technology advances and market demands are rapidly changing the situation. The increased connectivity implied by Internet and mobile device technology will have a major impact on control system architectures. Products are often based on commercial-off-the-shelf (COTS) components. The rapid development of component-based technologies and languages like Java increases portability and safety, and makes heterogeneous distributed control-system platforms possible. The evolution from static systems towards dynamic systems makes flexibility a key design attribute for future systems.

A key future challenge is to provide flexibility and reliability in embedded control systems implemented with COTS component-based computing and communications technology. Research is necessary on design and implementation techniques that support dynamic run-time flexibility with respect to, e.g., changes in workload and resource utilization patterns. The use of control-theoretical approaches for modeling, analysis, and design of embedded systems is a promising approach to control uncertainty and to provide flexibility. A related area is quality-of-service (QoS) issues and feedback scheduling approaches in control systems. In order to support this development it is important that the control community increases its efforts on development of control theory that it is aware of implementation-platform resource constraints. It is also important that the real-time computing community work hand in hand with the control community to develop models, methods, tools, and theory that match their respective requirements..

5.1 Acknowledgment

This work is supported by the the SSF/ARTES and SSF/FLEXCON research programmes on real-time systems and control, and by the LUCAS center for applied software research.

References

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

Abeni, L., L. Palopoli, G. Lipari, and J. Walpole (2002): “Analysis of a reservation-based feedback scheduler.” In *Proc. 23rd IEEE Real-Time Systems Symposium*.

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

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

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

Buttazzo, G., G. Lipari, and L. Abeni (1998): “Elastic task model for adaptive rate control.” In *Proc. 19th IEEE Real-Time Systems Symposium*, pp. 286–295.

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

Cervin, A. (2003): *Integrated Control and Real-Time Scheduling*. PhD thesis ISRN LUTFD2/TFRT--1065--SE, Department of Automatic Control, Lund Institute of Technology, Sweden.

Cervin, A. and J. Eker (2003): “The Control Server: A computational model for real-time control tasks.” In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*. Porto, Portugal. To appear.

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

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

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.

El-khoury, J. and M. Törngren (2001): "Towards a toolset for architecural design of distributed real-time control systems." In *Proc. 22nd IEEE Real-Time Systems Symposium*.

Garcia, C. E., D. M. Prett, and M. Morari (1989): "Model predictive control: Theory and practice – a survey." *Automatica*, **25:3**, pp. 335–348.

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

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

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

Henriksson, D., A. Cervin, and K.-E. Årzén (2002c): "TrueTime: Simulation of control loops under shared computer resources." In *Proceedings of the 15th IFAC World Congress on Automatic Control*. Barcelona, Spain.

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

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

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

Lincoln, B. (2002b): "A simple stability criterion for control systems with varying delays." In *Proceedings of the 15th IFAC World Congress*.

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

Liu, J. and E. Lee (2003): "Timed multitasking for real-time embedded software." *IEEE Control Systems Magazine*, **23:1**.

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

Lu, C., J. A. Stankovic, S. H. Son, and G. Tao (2002): "Feedback control real-time scheduling: framework, modeling and algorithms." *Real-Time Systems*, **23:1/2**, pp. 85–126.

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

Mercer, C. W., S. Savage, and H. Tokuda (1993): "Processor capacity reserves for multimedia operating systems." In *Proc. Fourth Workshop on Workstation Operating Systems*.

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

Richalet, J. (1993): "Industrial application of model based predictive control." *Automatica*, **29**, pp. 1251–1274.

Schinkel, M., W.-H. Chen, and A. Rantzer (2002): "Optimal control for systems with varying sampling rate." In *Proceedings of American Control Conference*. Anchorage.

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.

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.

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

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

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.