Co-design for Embedded Real-time Systems (CERTS'03)

(Satellite of the Euromicro Conference)

Porto, 2nd July 2003.

Chair — Iain Bate,
Department of Computer Science,
University of York
York, YO10 5DD, UK

iain.bate@cs.york.ac.uk

Organizing Committee — Karl-Erik Årzén
Joseph M Fuertes
Bob Madahar
Gerhard Fohler
Barrie Ross-Dow

Overview

Computing technology is quickly becoming a fundamental part of many commodity goods. While the demands for bigger and faster machines continue, a new wave of computing revolution is emerging: embedded computing. Previously, industry tailored applications to meet the capabilities of technologies, but now the time has come that technologies need to be tailored for applications. At the same time, the range of demands (e.g. power, dependability, cost etc.) have continued to grow. To best support these demands requires technologies that work across and correctly balance the different demands. A perceived weakness in the embedded real-time systems community is a shortage of events that cover multi-disciplinary topics such as control and scheduling, and hardware software co-design.

The aims of this workshop are to:

- identify other relevant cross-disciplinary topics to embedded real-time systems than the key ones listed above (hardware-software co-design, control-scheduling co-design),
- identify the state of the art, problems and open research areas related to embedded real-time systems.

Finally, we would like to thank all the people who helped with the event; the reviewers for their speed and their quickness, the organizing committee for their guidance, and the members of the IEE and Euromicro Technical Committee on Real-Time Systems who helped with the organisation.

Programme

HARDWARE SOFTWARE CO-DESIGN SESSION

09.15-10.00	Towards the Codesign of Large Complex Hard Real-Time Embedded Systems Neil Audsley, Real-Time Research Group, University of York
10.00-10.45	Design Space Exploration Approach for Reconfigurable Platforms K. Ben Chehida, University of Nice Sophia Antipolis
10.45-11.15	Coffee Break
11.15-11.45	**Discussion on hardware software co-design issues** *chair Neil Audsley*

GENERAL CO-DESIGN SESSION

11.45-12.30	Functionality/Dependability Co-design in Real-Time Embedded Software *Elisabeth A. Strunk, Department of Computer Science, University of Virginia*
12.30-12.45	A Configuration Manager for Embedded Real-Time Systems *Roman Gumzej, Faculty of Electrical Engineering and Computer Science, University of Maribor*
12.45-1.00	**Discussion on general co-design issues** *chair Elisabeth Strunk*
1.00-2.00	Lunch

CONTROL SCHEDULING SESSION

2.00-2.45	Resource-Constrained Embedded Control Systems: Possibilities and Research Issues *Karl-Erik Årźen*, Department of Automatic Control, Lund Institute of Technology
2.45-3.00	Real-Time and Closed-loop Control Co-design and Delay-dependent Feedback Scheduling *Daniel Simon, INRIA Rhône-Alpes*
3.00-3.15	Impact of Scheduling Policies on Control System Performance *Henrik Schiøler, Department of Control Engineering, Aalborg University*
3.15-3.30	Modelling Self-Triggered Tasks for Real-Time Control Systems *Manel Velasco, Industrial Informatics and Automatics Department, Technical University of Catalonia*
3.30-4.00	Coffee Break
4.00-4.15	Using Jitterbug to Derive Control Loop Timing Requirements *Anton Cervin, Department of Automatic Control, Lund Institute of Technology*
4.15-4.45	**Discussion on control scheduling co-design issue** *chair Karl-Erik Årźen*
4.45-5.15	**Demonstrations**

Towards the Codesign of Large Complex Hard Real-Time Embedded Systems

Neil C. Audsley

Real-Time Systems Research Group,
Department of Computer Science, University of York, York, UK
Neil.Audsley@cs.york.ac.uk

Abstract. The development of long lifetime hard real-time systems is becoming increasingly difficult, due to increased system complexity and pressure to reduce development times. This paper considers the use of codesign techniques for the development of future hard real-time systems. A three phase process is outlined. Firstly, non-functional design decisions are captured and structured in a manner that enables trade-offs between different non-functional properties to be considered (primarily time). Secondly, system functions are generated by use of high-level modelling tools (eg. Matlab) to reflect the trend towards these technologies for increased automation and higher levels of abstraction within the development process. Thirdly, low-level implementation performs relatively conventional hardware software codesign to map the functions generated onto a platform whilst meeting the non-functional requirements.

1 Introduction

Real-time embedded systems are becoming increasingly complex, in terms of their functional and non-functional properties, so making their design and implementation evermore difficult. However, systems need to be developed in shorter times, due to business requirements to reduce the time-to-market. Such conflicting pressures are often addressed by increasing the automation within the development process, e.g. by utilising high-level modelling tools (UML, Matlab, MatrixX etc) and utilising the automatic software generation facilities within those tools for system software production. Effectively, the abstraction level at which most of the system is developed is raised from the software level to a modelling level.

This general approach of high-level specification and greater process automation is extremely attractive in order to reduce system time-to-market. It can be seen in much hardware-software codesign research, which enables automatic derivation of a hardware architecture and application software from a high-level specification [1–3]. Such approaches are limited in terms of the scale of the system that can be developed (usually small uniprocessor or multiprocessor based systems rather than large distributed systems); limited traceability from specification to final design (due to automation); limited ability to change / update parts of the system with ease at some later date (rather the entire modified system has to be re-generated with no guarantee that the new hardware architecture will be identical to the original).

Many real-time embedded systems are developed for domains that have additional constraints than those assumed by "traditional" codesign work. Current research at York is concentrating upon the codesign of complex long-lifetime hard real-time systems. These systems have a number of important requirements. Firstly, timing predictability is key – failure of the system to meet timing requirements (eg. process deadlines) can result in catastrophic failure of the system. Thus, it is important to be able to show that all timing requirements of the system are met prior to run-time. Secondly, the system must be shown to be fit-for-purpose prior to use [4]. Often some regulatory authority (eg. aerospace, nuclear, medical) requires documentary evidence that both the development process and the system are sufficiently robust and correct before the system can be used. Finally, the system must be amenable to change / upgrade. This must be carried out in manner that minimises the impact of the change upon the rest of the system, to simplify the

process of convincing the regulators that the changed system has not introduced any unexpected problems.

This paper outlines an approach currently being developed that seeks to take advantage of increased automation for long-lifetime hard real-time embedded systems, whilst ensuring system timing predictability and amenability to change.

2 Overview of the Development Process

The process is broken into three phases:

1. *High-Level Design and Timing Optimisation*
   Captures design choices in a structured manner to aid traceability (and provide supporting evidence of the system being fit-for-purpose), whilst providing automatic optimisation of key system non-functional properties (including timing) to ensure that non-functional requirements will be met in the final system. This phase develops constraints (in terms of time, allocation, resource usage etc.) that are placed upon a subsequent implementation of the system.
2. *System Function Generation*
   Utilises appropriate modelling techniques (eg. UML, MatrixX, Matlab) for the modelling of desired functional behaviours. Software to implement these functions can be automatically generated from these tools. Where the modelling techniques available are not sufficient to express required functions, manual software development can occur (using a suitably rigorous software development process).
3. *Low-Level Implementation*
   Produces a hardware architecture that supports the functions generated whilst meeting the non-functional constraints generated by the high-level design phase. Restricted codesign techniques for automatic hardware and software production are used.

The process is illustrated in Figure 1. It is assumed that outline requirements of intended system behaviour are available at the start of this process. It is also assumed that new, changed or clarified requirements can become available during the development process (or after initial development, when system change or upgrades are required).

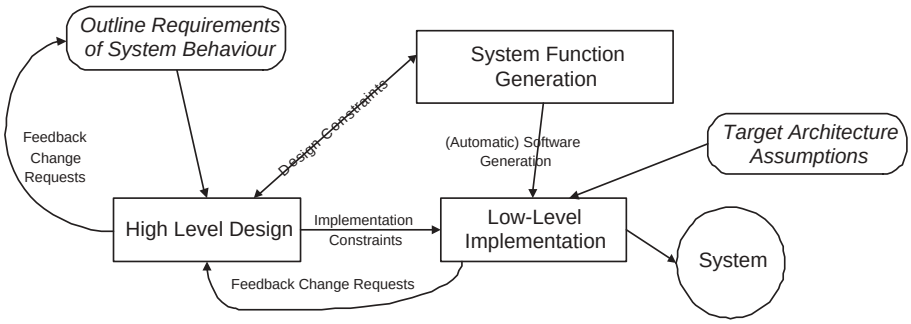

Fig. 1. Overview of Process.

Non-functional requirements are considered by the high-level design process to order to establish a set of constraints and requirements that must be met by the low-level implementation. This process can feed back any recommendations for change to the requirements process to the high-level modelling process if it finds contradictions or inconsistencies.

Figure 1 shows the system function generation phase that automatically generates the application functions in a manner that can then be taken by the low-level implementation phase

(usually expressed in a high level programming language). This does not include any required OS functionality, which is generated during the low-level implementation phase).

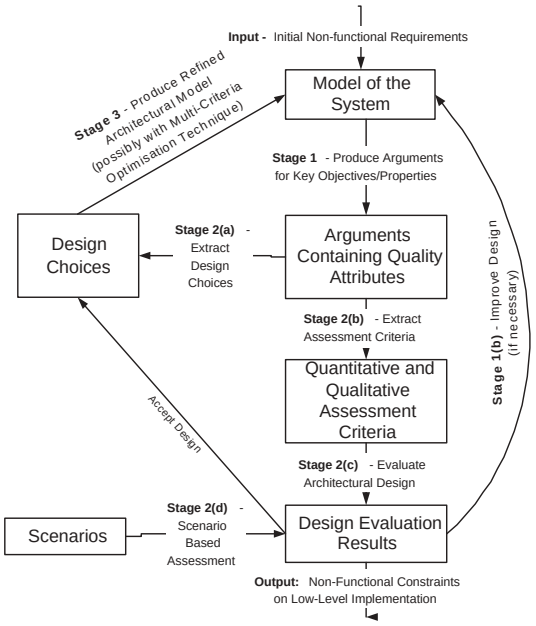

Fig. 2. High-Level Design Process.

2.1 High-Level Design and Timing Optimisation

Figure 2 provides a diagrammatic overview of the high-level process. It assumes the availability of non-functional system requirements. The process allows the derivation of design choices, identifying where different solutions are available for satisfying a key system requirement, managing the sensitivities / dependencies between components and design decisions. The process also identifes the constraints that must be placed upon functional component design, such that the overall objectives of the system are met. Such constraints are passed to the system function generation process as they are found. In practice, this will occur whenever design decisions are committed, rather than merely contemplated.

The high-level design process is also the recipient of constraints from the system function generator. These constraints include the functional properties that must be considered by the high-level design process during the development of the non-funcitonal design. For example, the number of processes and / or functions that must be accounted for during timing analysis.

Finally, the high-level design process collects all the design rationale (ie. the design choices, sensitivities, dependencies and design decisions) into a repository to aid traceability. This is particularly important if changes to the system need to be made during the lifetime of the system, eg. for planned updates or major revisions sometime after the system has been initially deployed.

The iterative nature of the high-level design process shown in Figure 2 is used to develop the design, in terms of further decomposition of the design. Eventually, when sufficient design development has occurred, the low-level implementation phase can be utilised.

A key aspect of the high-level design process is that many decisions made will constrain the eventual system implementation and architecture. For example, if during high-level design it is

determined that redundancy is required (to meet fault-tolerance requirements), then this will be specified to the low-level implementation phase.

The high-level design process is largely manual. However, in the generation of timing constraints, the high-level process uses an automatic optimisation process.

Further details of the high-level process are given in [5].

2.2 System Function Generation

The system function generation phase encompasses the mapping of funcitonal requirements to implementation. This is usually achieved using appropriate modelling tools (eg. UML, Matlab, MatrixX), that permit automatic generation of an implementation, as represented in a high-level language such as C, Ada, VHDL etc. The resultant "programs" can be passed to the low-level implementation phase. Whilst the generation of the program is automatic, the use of the tools themselves is manual. Many of the modelling tools include model-level testing and simulation of the model (ie. model execution) which aids verification that the model is meeting functional requirements.

The current realisation of the system function generation phase is limited to tools that can produce Ada (including UML, Beacon, MatrixX, Matlab). It is noted that the limitations on language are largely imposed by the current scope of the low-level implementation.

A key part of this phase is the interation of contraints with the high-level design phase, as described above. This phase is responsible for identifying functions that need to be executed (and perhaps upper and lower bounds on some timing properties), the high-level design phase is responsible for assigning execution times etc.

Ada for Hard Real-Time Systems Ideally, the system functions are generated in Ada, as the Ada language [6] facilitates the programming of real-time systems. It contains facilities for programming-in-the-small (ie. sequential programming), facilities for programming-in-the-large (ie. data abstraction and packages), together with facilities for concurrent programming (ie. tasks and inter-task communication). In addition, subsets of Ada have been developed that effect restrictions upon Ada that enable conforming programs to be statically analysed for timing, resource and functional properties.

The SPARK subset of Ada [7] restricts the sequential part of the language. Conformant programs can be proved (partially) correct. SPARK does not contain any dynamic constructs, including concurrency (and synchronisation), the access (pointer) type, variant records (hence no object-oriented capabilities). Sub-programs are no longer allowed to recurse, nor can procedure pointers be used. These restrictions make all subprogram call trees known at compile-time, and all variable references resolve to only one instance. The SPARK Ada subset is consistent with the requirements for real-time system timing analysis in that all conforming programs are statically analysable for their worst-case properties.

The Ravenscar tasking profile[8] is a statically analysable tasking subset. Unlike full Ada, Ravenscar compliant code is predictable in its timing behaviour and resource usage. The Ravenscar profile makes no comment on the sequential part of the language. The definition of Ravenscar is effectively included in the Ada standard, being part of Annex H (Safety and Security) which comments on applicability of Ada language features for use in safety-related systems.

A SPARK / Ravenscar conformant Ada program consists of a number of concurrent tasks, that interact via protected objects. These objects enforce mutual exclusion over some procedures and associated data within the object. Interaction with other devices is achieved by representation clauses, which associate a specific memory location with a program variable, so achieving a memory mapped programming model. Also, conformant programs are analysable for timing (and other statically determinable) properties.

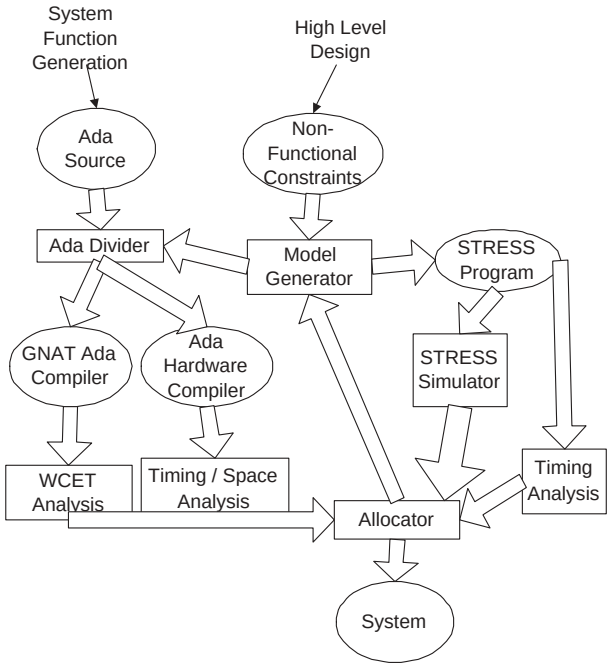

Fig. 3. Overview of Low-Level Implementation Process.

2.3 Low-Level Implementation

The low-level implementation phase allows the mapping of functions expressed in Ada (developed by the system function generation phase), constrained by the non-functional requirements established by the high-level design phase, into an actual hardware and software implementation. Although a number of design decisions have been taken during the high-level design phase, there is still considerable freedom for the low-level implementation phase to search a wide range of potential solutions.

Currently, the physical target architecture assumed is that of a single Field Programmable Gate Array (FPGA) [9], coupled to a number of RAM banks. Clearly, limiting the target architecture to a single FPGA restricts the solution space. However, the physical size of current high-end FPGAs is large, ensuring that substantial functionality can be achieved on a single device. Also, the presence of the RAM banks ensures that (parts of) the FPGA can be used for softcore CPUs, further extending the size of the functionality that can be implemented upon the target.

The low-level implementation phase follows a timing analysis driven approach, as motivated earlier in this paper. The timing characteristics of an Ada program are modelled sufficiently for analytical timing analysis to occur. The actual implementation of the system is then checked against the assumptions of the model. If the assumptions still hold (eg. that the WCET of a software task is no more than some value), then the full implementation will meet its timing requirements.

The method is illustrated in Figure 3. It consists of an iterative process with two main parts:

1. Modelling and simulating the timing and interaction properties of the software.
2. Compilation to hardware circuit and CPU instructions of a given allocation of the software.

These stages provide feedback in terms of timing characteristics of the actual software (eg. WCET of software tasks, or circuit speed and size of an FPGA task); analytical timing analysis; and simulation of the system. This is sufficient for the system configuration, in terms of the allocation

of tasks to hardware or software, to be evaluated. As a consequence, a new allocation can be determined to further improve the system.

The low-level implementation phase takes as inputs the constraints and requirements established by the high-level phase, together with the application software (it is assumed that the high-level modelling tools are able to generate (automatically) application software). This phase is able to analyse the software for timing characteristics (eg. worst-case execution times) and find a suitable platform on which the application software can execute to meet its timing requirements. Note that the low-level implementation phase is not necessarily restricted to implementations on CPU, it can also consider direct mapping to hardware (if permitted by the target architecture assumptions). The low-level phase is entirely automatic.

Compilation The compilation of Ada to binary (ie. the software route) utilises the GNAT Ada compiler [10]. The compilation to hardware is achieved using the hardware Ada compiler[11, 12].

Conventional compilation of Ada to CPU instructions follows the normal compilation path [10]. Note that the concurrent features of Ada require a run-time (or kernel) to be present at run-time. One function of the run-time is to provide scheduling between the different application tasks. Given the restricted concurrency model of Ravenscar conformant programs, the run-time required for such programs is simple – indeed, a simplistic run-time was one of the prime motivations for the Ravenscar subset.

SPARK / Ravenscar conformant Ada programs are ideal for direct compilation to hardware circuit. In[11, 12] an Ada compilation process is described for such programs. Essentially, concurrency within Ada can be represented on hardware as truely parallel tasks. In terms of the Ravenscar tasking subset, the main implication is that task scheduling is no longer required – indeed, no run-time is required at all. The sequential language used within a task is relatively straightforward to compile to hardware, as the restrictions of the SPARK subset ensure that no dynamic statements are present in a task.

Protected objects enforce mutual exclusion over some procedures and associated data. Hardware compilation does not remove the need for mutual exclusion, so protected objects remain. When contention exists over access to a protected object, the default locking policy of Ada is used, that is ceiling protocol [13], where ceiling priorities are defined in terms of the priorities of the tasks that

2.4 Meeting Timing Requirements

A dominant theme throughout the process is an emphasis upon ensuring that the system will meet any non-functional requirements, in particular timing. Static offline timing analysis is used to drive many of the decisions taken in both the high-level and low-level phases [14]. The analysis proceeds by extracting a model of the key timing properties of the system then calculating the worst-case timing behaviour of the system. If the timing properties of the system are met in the worst-case, the system will meet its timing requirements at run-time (assuming that the implementation does not invalidate any assumptions made in the model). Note that extensive testing of an implementation does not necessarily cover the worst-case.

The use of static offline timing analysis enables a correctness by construction approach to be used to develop the system. Essentially, the high-level design phase uses timing analysis to generate many of the constraints given to the low-level implementation phase. This then generates an architecture that meets the contraints, so ensuring that the system will meet its timing requirements.

2.5 Comparison with Codesign Approaches

It is appropriate at this juncture to compare the overall process outlined above with those of conventional hardware-software codesign, as typified by the approaches presented in [15].

Codesign approaches assume that a complete specification is available prior to system generation. To some degree, this is also seen in the process given above, where a reasonably complete set of requirements is required prior to the commencement of design. However, in realistic large hard real-time system developments, the precise specification is often not readily available until late in the development. Normally, the high level design process and the modelling has started before a total specification is available.

Codesign approaches usually assume a single process for development. This is not usually the case for large hard real-time systems, where parts of the design and implementation are sub-constracted to different companies. It is important that the overall process described above is amenable for use by a subcontractor building part of a system (eg. a sub-system), a prime contractor assembling the entire system, or a sub-contractor contributing either software (eg. by some system model).

Codesign approaches utilise an automatic partitioning of functionality between hardware and software implementation. This is adopted in the low level implementation phase of the process outlined above. Here, functions expressed in a high level language (eg. software language such as Ada) are mapped to a combination of logic and CPU, utilising hardware compilers that map programs in a high-level language such as Ada, to circuit (ie. FPGA) [11, 12].

Codesign approaches assume a co-verification phase as part of the iterative search during system generation. In the overall process described above, verification occurs in many areas. As part of the high level design process, key non-functional requirements are verified as part of iteration towards a design solution, eg. timing. As part of the system function generation phase, functional properties will be verified. This occurs at the model level where appropriate. During low level implementation, further verification of properties (both functional and non-functional) is performed during the iterative search for an implementation solution.

The key part of codesign that is adapted throughout the overall process given above is the automatic trade-off of design choices, particularly in the non-functional domain. This is seen in the high-level design phase where timing properties (amongst others) can be optimised via trade-off analysis to provide a good technology independent design. The low-level implementation phase automatically finds a solution to meet the non-functional and functional designs generated by the high-level design process and system function generation phases respectively. In many ways, this low-level implementation phase is closest to the normal codesign approaches.

3 Conclusions

This paper has described a process which utilises codesign techniques within the development process for complex hard real-time systems. The motivations for inclusion of codesign techniques includes the structured capture of non-functional design decisions in a more structured and integrated manner than current practice suggests; technology independent design is encouraged, which postpones decisions regarding the target technology until late in the development process. Combined with automatic mapping of system functions to a target architecture within the constraints imposed by the non-functional requirements (and related design decisions), this provides a better process for complex hard real-time system development. The process ensures that key non-functional properties are met by the design and eventual implementation. Importantly, the process is driven from a timing analysis perspective, closely integrating static timing analysis within the process. This imposes a correctness by construction approach, rather than the build and test approach seen often in practical developments.

Further work presently being undertaken is seeking to expand the process and methods described in this paper in a number of ways. Firstly, more non-functional properties are being considered, including safety, reliability, size, cost and power. Secondly, potential implementation architectures are being expanded to include fully distributed systems with associated network controllers and hardware.

References

1. Suzuki, K., Sangiovanni-Vincentelli, A.: Efficeint Software Performance Esimation Methods for Hardware / Software Codesign. In: Proc. Design Automation Conference. (1996)
2. D. Gajski, F. Vahid, S.N., Chong, J.: System-Level Exploration with SpecSyn. In: Proc. Design Automation Conference. (1998) 812–817
3. Henkel, J., Ernst, R.: A Hardware/Software Paritioner Using a Dynamically Determined Granularity. In: Proc. Design Automation Conference. (1997) 691–696
4. Yeh, Y.C.: Dependability of the 777 primary flight control system. Proceedings 5th IFIP Working Conference on Dependable Computing for Critical Applications (1995)
5. Bate, I., Audsley, N.: Architecture trade-off analysis and the influence on component design. In: Proc. Workshop on Component Based Software Engineering. (2002)
6. Taft, S., Duff, R., eds.: Ada 95 Reference Manual: Language and Standard Libraries, International Standard ISO/IEC 8652:1996(E). Volume Lecture Notes in Computer Science 1246. Springer-Verlag (1997)
7. Barnes, J.: High Integrity Ada: The SPARK Approach. Addison-Wesley (1997)
8. Burns, A., Dobbing, B., Romanski, G.: The Ravenscar Tasking Profile for High Integrity Real-Time Programs. In: Reliable Software Technologies, Proceedings of the Ada Europe Conference, Uppsala. Volume 1411., LNCS, Springer-Verlag (1998) 263–275
9. Xilinx Corporation: Xilinx Product Information : http://www.xilinx.com/products. (2003)
10. Ada Core Technologies: GNAT Ada Compiler : http://www.gnat.com. (2001)
11. Ward, M., Audsley, N.C.: Hardware Compilation of Sequential Ada. In: Proceedings of CASES 2001. (2001) 99–107
12. Ward, M., Audsley, N.C.: Language Issues of Compiling Ada to Hardware. In: Proceedings of Ada Europe 2002. (2002)
13. Sha, L., Rajkumar, R., Lehoczky, J.: Priority Inheritance Protocols: An approach to real-time synchronisation. IEEE Transactions on Computers **39** (1986) 1175–1185
14. Audsley, N., Burns, A., Richardson, M., Wellings, A.: Hard Real-Time Scheduling: The Deadline Monotonic Approach. In: IEEE Workshop on Real-Time Operating Systems and Software, Atlanta, GA, USA (1991) 133–137
15. DeMicheli, G.: Readings in Hardware / Software Codesign. Morgan-Kaufman (2001)

Design Space Exploration Approach for Reconfigurable Platforms

K. Ben Chehida
I3S, University of Nice Sophia Antipolis, CNRS
Les Algorithmes/ Euclide B,2000 route des Lucioles
BP 121, 06903 Sophia-Antipolis Cedex
Tel. 0334 92942788
ben_cheh@i3s.unice.fr

M. Auguin
I3S, University of Nice Sophia Antipolis, CNRS
Les Algorithmes/ Euclide B,2000 route des Lucioles
BP 121, 06903 Sophia-Antipolis Cedex
Tel. 0334 92942777
auguin@i3s.unice.fr

ABSTRACT
This paper presents a Genetic Algorithm (GA) based approach for design space exploration targeting an architecture composed of a processor and a dynamically reconfigurable datapath (FPGA). From an acyclic task graph and a set of Area-Time implementation trade off points for each task, our GA performs HW/SW partitioning and scheduling such that the global application execution time is minimized. The efficiency of our GA is established through its application to a motion detection application with hard real time constraints.

Categories and Subject Descriptors
J.6 [**Computer Applications**]: Computer-Aided Engineering - *Computer-aided design (CAD).*
C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems - *Real-time and embedded systems.*

General Terms
Algorithms, Design, Performance.

Keywords
Genetic algorithm, HW/SW partitioning, clustering, dynamic reconfiguration, codesign.

1. INTRODUCTION
The recent improvements in size, flexibility and reconfiguration speed of FPGAs make this technology very attractive for low cost and high speed embedded system design. Connecting a reconfigurable device to a programmable processor in a single chip **[1, 2, 3, 4, 5],** constitutes a very flexible and efficient architecture that can be used in a wide variety of embedded devices (for example, intelligent terminals or sensors such as a networked camera **[6]**). Rapid development of embedded systems using this software/reconfigurable technology suffers from lack of advanced system level design tools which exploit efficiently the parallelism and the dynamic reconfiguration capabilities of the architecture.
The aim of the project EPICURE[1] is to introduce a design methodology for dynamically reconfigurable computing platforms composed of a general purpose processor (*CPU*) and a dynamically reconfigurable datapath (*FPGA…*). From performance/cost estimations of the functions of the application on the processor and on the reconfigurable circuit, we have developed a partitioning tool which provides a mapping and a schedule of the tasks on the architecture.
The organisation of this paper is as follow. In Section 2 we formulate our problem to match the application and the architecture models. The description of our partitioning approach based on a genetic algorithm is provided in Section 3, and in Section 4 are outlined some results on a motion detection application example. We conclude with Section 5.

[1] This project is supported by the French Ministry of Research and Education through the *Réseau National des Technologies Logicielles*. The partners of the project are CEA, Thales, Esterel Technologies, LESTER - Université de Bretagne Sud and I3S - Université de Nice Sophia Antipolis/CNRS.

2. PROBLEM FORMULATION
The dynamic reconfiguration technology is investigated by numerous research groups (e.g. **[7]**,**[8]**) and would be very attractive for commercial products. Exploiting dynamic reconfiguration requires rather a coarse grain parallelism to reduce the relative cost of reconfiguration and data transfers.
The partitioning problem imposes to specify a model of the target architecture and one of the target application. A precise definition of these models gives more realistic behaviors but a precision excess may drastically increase the partitioning time. Compromises have to be made. The next paragraphs will present the models retained in our approach.

The Architecture Model
Our target architecture is composed of a Processor connected to a Dynamically Reconfigurable Processing Unit (DRPU) as depicted in figure 1. We consider an embedded processor with a relatively deterministic behavior (WCET…) and a reconfigurable data path with Logic Cells (LCs) and Dedicated Cells (DCs). This kind of chips tends to be generalized, so we adapted our model to handle these evolutions. The LCs are functional elements used to synthesize logic and operators to build tasks and the DCs are more elaborated preset blocks (e.g. Block-RAMs, Multipliers).
Our partitioning approach takes into account the partial reconfiguration of the FPGA: the reconfiguration time depends on the number of LCs involved in the function realization. Complete reconfigurations of the circuit can be considered as well. However, we made the realistic assumption that there is no overlapping between partial reconfiguration and treatments on the FPGA. The data transfers between the processing units is done throw a double port memory (figure 1) situated in the interface and connected to the processor throw its data bus (bus

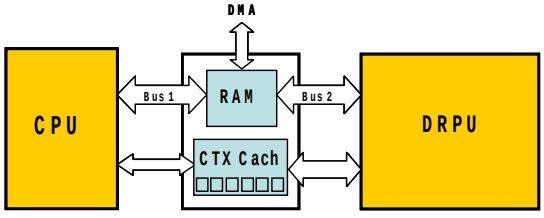

Figure 1. The Target Architecture

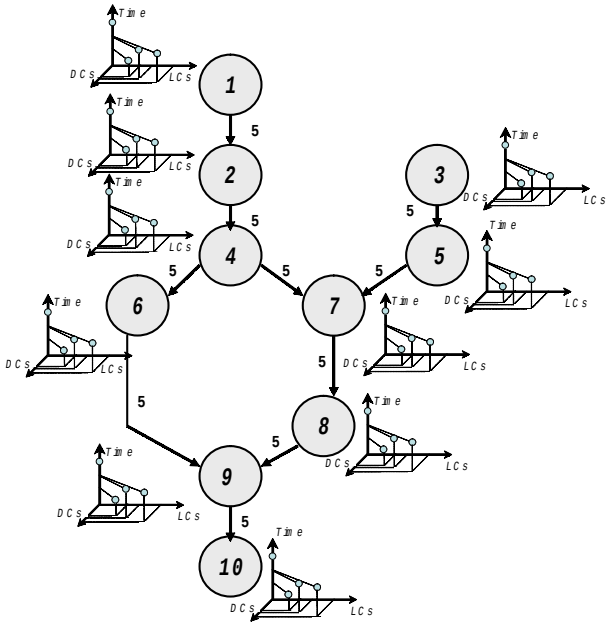

Figure 2: Task Graph and Area/Time trade off points

1) and to the FPGA throw a specific bus (bus 2). A Second assumption is that the communication on bus 1 are blocking for the CPU and the one on bus 2 are not for the treatment on the FPGA.

The communication time between two tasks mapped to SW is set to zero. Let ρ_i be the number of bytes on edge e_i and λ_l be the number of bytes per packet supported by bus l. Let τ_l be the communication time of a packet on l and Ω_l be the access time per packet on that bus. The communication time on edge e_i between a task τ_i mapped to SW (respectively to HW) and a task τ_j mapped to HW (respectively to SW), [9] is set to :

$$T_{com}(e_i) = \left\lceil \frac{\rho_i}{\lambda_1} \right\rceil . (\alpha_1 + \Omega_1) + \left\lceil \frac{\rho_i}{\lambda_2} \right\rceil . (\alpha_2 + \Omega_2) \qquad (1)$$

and the communication time on edge e'_i between two tasks mapped to HW is temporarily set to worst case communication time :

$$T_{com}(e'_i) = 2. \left\lceil \frac{\rho_i}{\lambda_2} \right\rceil . (\alpha_2 + \Omega_2) \qquad (2)$$

and will be updated once the HW contexts defined. For clarity reasons, we set all the communication times to 5 units in fig. 2.

The application model

The application model considered is a function or task level data flow graph specification. From this task graph, the goal of partitioning is to select whether to put each task into *SW* or *HW* such that the whole execution time is minimized.

Each node of the acyclic data flow graph denotes a task that can be mapped to the *SW* or the *HW*. The amount of data (bytes) that must be transferred between two connected tasks is associated with each edge. A task can begin its execution when all its parent tasks and incoming edges have completed their executions. *SW* and *HW* runtimes of each task are estimated in terms of Area–Time trade off points. *SW* runtime performance is estimated through profiling and *HW* (*FPGA*) performance/area estimations are performed at the behavioral level. The number of implementation points can differ for each task depending on the exploitation of the available parallelism in the task **[10]**. Figure 2

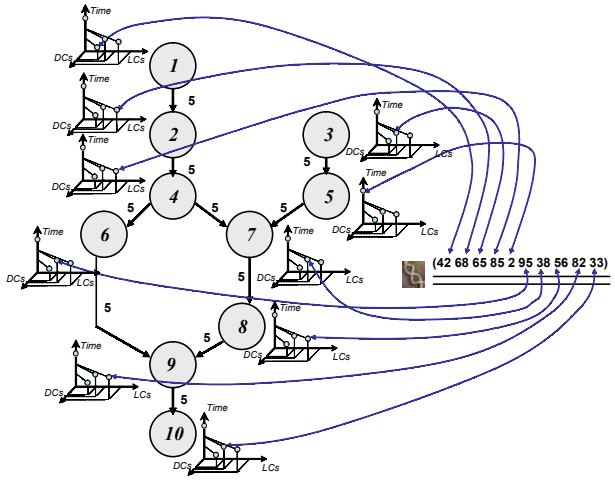

Figure 3: Chromosome encoding

shows an example of a task graph and the Area-Time implementation points for each task.

The partitioning approach is based on a genetic algorithm that realizes a design space exploration by generating different mappings of the tasks on the processor and the FPGA. Evaluation of the execution time of the architecture for each mapping requires to define a schedule of the tasks including reconfigurations for context switching and data transfers between tasks. This evaluation is performed with a clustering heuristic inspired by the COSYN method **[9]**.

3. HW/SW PARTITIONING USING A GENETIC ALGORITHM

We model and solve our partitioning problem through a Genetic Algorithm (GA). This kind of algorithms is based on five main steps : the *Encoding*, the *Evaluation* (in term of a cost function), the *Selection*, the *Generation* and the *Renewal* steps.

Chromosome Encoding

The encoding of any solution corresponds to the binding of each task to an implementation point. Our encoding method codes a chromosome *C* with an array of genes of length *N* where *N* is the number of tasks. Each gene *C(i)* is an integer representing a percentage. The maximum 100% value that can take *C(i)* is associated with the most *LCs*-based expensive implementation of task *i.* The selected implementation point is the nearest point to *C(i)* on the *LC's* axis. If there is more than one implementation point having the same *CLs* number, we compare the *DCs* picking also the nearest point to *C(i)* on that axis, and so on. All the solutions delivered by this encoding method are viable.

The chromosome example presented in Figure 3 assigns only task 5 to a *SW* implementation and all the others to *HW*. Tasks mapped to *HW* have to be grouped into Contexts (or *Clusters*) to finally evaluate the effectiveness of the individual.

Chromosome Evaluation

The fitness of every chromosome (solution) delivered by GA is evaluated allowing its ranking onto the current population. A solution is evaluated by its overall execution time including the reconfigurations for context switching and data transfers between tasks.

Contexts definition (Clustering):

We use a Clustering approach as addressed in [9] to group tasks in contexts. We first assign priority levels to tasks, starting from the graph's leafs. The priority level of a task is the longest path

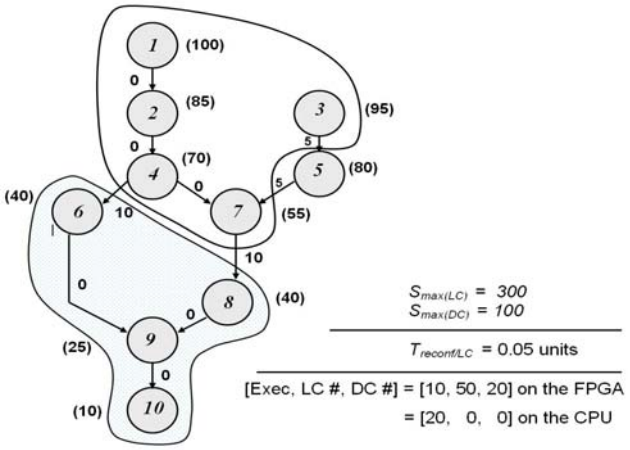

Figure 4 : Clustering and communication time update

from the task to a leaf evaluated as computation and communication costs (Fig. 4). To reduce the schedule length, we need to decrease the length of the longest path by *clustering* tasks along it in order to reduce the communication costs along the path. The priority P_i of task i is computed considering the priority of its successors j and the communication time between i and j according to equation (3):

$$P_i = T\_exec(\tau_i) + Max_{(j)}\,(P_j + T\_com(\tau_i, \tau_j)) \quad (3)$$

The cluster size $S_{max(LC)}$ is limited to the maximum FPGA size in terms of *LCs* (in practice, 80 to 85 % of the total number of *LCs*) and $S_{max(DC)}$ is the maximum size for the corresponding *DC*.

Initially, all the tasks are sorted in the decreasing order of their priority levels. We pick the unclustered task $\tau_i(t_i, S_{i(LC)}, S_{i(DC)})$ with the highest priority level, where t_i is the execution time and $S_{i(LC)}$ (respectively $S_{i(DC)}$) the number of *LCs (*respectively *DCs)* defined by the implementation pointed by *C(i)* in the chromosome, and mark it clustered. The available resources of the current cluster $Ress(C_{curr})$ (initially to *Smax*) are decreased by the corresponding S_i. This context building is iterated with tasks $\tau_j(t_j, S_{j(LC)}, S_{j(DC)})$ assigned to *HW* while:

$$S_{j(LC)} \le Ress_{(LC)}(C_{curr}) \;\&\&\; S_{j(DC)} \le Ress_{(DC)}(C_{curr}) \quad (4)$$

Else, a new cluster is created and the process is repeated until all the HW tasks are assigned to clusters. The reconfiguration time depends on the quantity N_k of *LCs* needed for mapping the context k on the FPGA. Let T_F be the time for a full reconfiguration then the reconfiguration time per LC is given by:

$$T_{reconf/LC} = \frac{T_F}{S_{\max}} \quad (5)$$

We evaluate the reconfiguration time of the context k by:

$$T_{reconf}\ (k) = N_k\ .T_{reconf/LC} \quad (6)$$

Once the contexts are defined, the algorithm updates the *intra-Context* (within a context) and *inter-Contexts* (between different contexts) communication times. *Intra-Context* communication times are set to zero.

In figure 4 and for simplicity reasons, we have fixed the execution time, the number of LCs and DCs for each task depending on the correspondent allocation.

Let $E_i(k)$ and $E_o(k)$ be respectively the incoming and outgoing edges of context *k.* for each edge $e_j \in E_o(l) \cap E_i(k)$ of contexts *l* and *k.* The communication time is updated by:

$$T_{com}(e_j) = Max\ (\ t(e_j),\ T_{reconf}(k)) \quad (7)$$

Where $t(e_j)$ is the communication time computed using (2). Hence, after updating communication times (see fig. 4), the global execution time is computed starting from the roots of the DFG and considering the ASAP execution time of each task. This global execution time (which is the cost of this solution) is the maximum ASAP execution time among all the leaves.

Chromosome selection:

Selection of solutions by GA is performed by the *Tournament* technique. A number ($N_{parents}$) of tournaments are performed, each one opposes a given number of individuals randomly chosen in the current population to finally select the fittest to be one of the parents allowed to reproduce.

Chromosome generation

Genetic operators are used on the $N_{parents}$ individuals selected by the *Tournament* technique to generate the $N_{children}$ solutions representing the new offspring.

Mutation operators:

Mutation randomly selects a gene (or a set of genes) and changes its value. The mapping of a task can change from a *SW* to a *HW* implementation, *HW* to *SW,* or the task may remain in *HW* but using a different implementation point. Five mutation operators are used in our algorithm. That permits large jumps in the solution space assuring a pure exploration process.

Crossover operators:

Two parent's chromosomes are cut at the same offsets (randomly set) from their starting points and the portions following the cut are swapped. Two crossover operators are used in our algorithm: A simple point (*1p-Cross)* and a double point *(2p-Cross)* crossover operator.

Renewal

After generation of the new offspring, the renewal of the population is performed according to the *elitism* principle. *Clones* are not allowed in our renewal procedure because they can invade the whole population leading to a genetic drift. When a number of generations N_{gen} has passed without improvements of the best individual, GA halts and displays the best encountered solution.

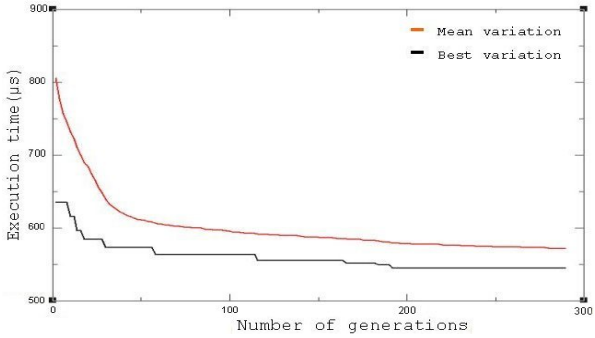

Figure 5: Best individual's cost and the mean cost values

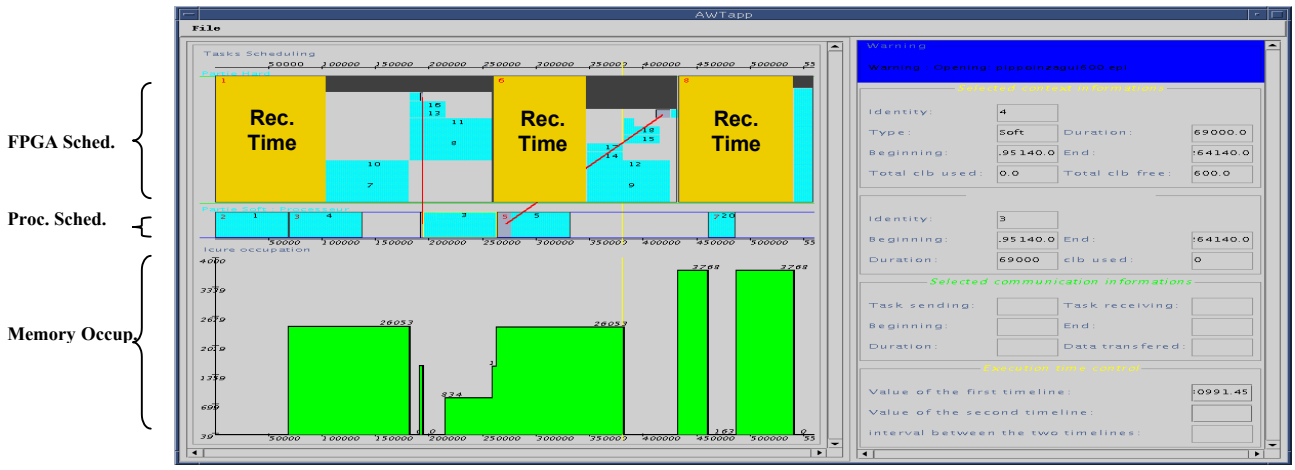

Figure 6: A partitioning solution

4. EXPERIMENTAL RESULTS

In this section we present results of the genetic algorithm for HW/ SW partitioning. Our partitioning algorithm is implemented in C++ on an Ultra Sparc 5 Unix workstation.

The benchmark used to evaluate the result quality of our partitioning algorithm is motion detection (MD) application that performs object labelling with a real time constrain of 40 *ms* per image. Considering a coarse grain decomposition of the application, we set the number of C procedures to 16.

A *SW* only implementation on an ARM 922 processor leads to an execution time of 76.44 *ms*. So we need to accelerate some portions of the application on *HW* to fit the deadline constraint. Given the execution time estimations on the ARM922 processor and the Xilinx Virtex-E family FPGAs (with a reconfiguration time per LC of 5µs), we fix the buses speed and width (bus 1 between the processor and the interface: $\lambda_1 = 128$, $\tau_1 = 10$ *ns;* bus 2 between the FPGA and the interface: $\lambda_2 = 256$, $\tau_2 = 15$ *ns*) and the memory access time $\Omega = 2$ *ns*.

GA is executed with an initial population size N_{Indiv} of 600 and an offspring size $N_{children}$ of 200. The GA terminates when N_{gen} = 100 generations have passed without improvements of the best solution. Towards the end of the run, a convergence is observed as displayed in figure 5. This figure shows the evolution of the best individual's cost and the mean cost over several generations. The CPU run time on the Ultra 5 workstation of the GA on the MD application is in the range of 4 to 6 *minutes*.

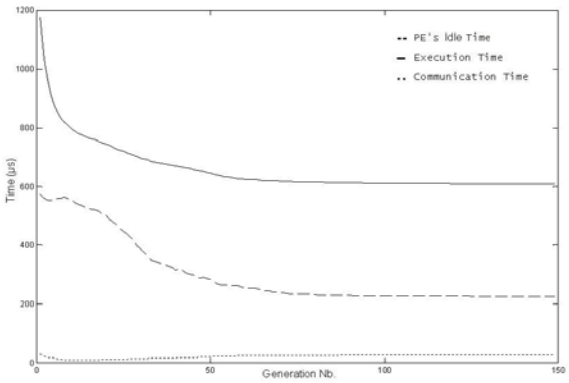

Figure 7: The PE's idle time, the mean exec. time and the mean com. time

The GA gives a total execution time of 28.1 *ms* for the hole application with 43200 *LCs* and 150 *RAM blocks* that can fit on a Virtex XCV2000E. Note that these results are obtained for a fixed granularity of *LCs*. Considering an FPGA with a different granularity requires new estimations and partitioning.

We must notice here that the behaviour of the clustering/scheduling algorithm in the GA consists in exploiting the available *LCs* and *DCs* in the FPGA to parallelize and to speed up executions of tasks. However, the allocation of a new task to a HW context, in the case of a partial reconfiguration, leads to delay the executions of the other tasks already allocated in that context since the reconfiguration time of the context is augmented.

As communications play an increasing role in today's SOC components, it is also interesting to see the variation of the mean communication time over several generations. Figure 6 shows that, after a short decrease, the mean communication time increases as the overall execution time is dropped. That means that the refinement procedure of the GA tries to exploit at best the available parallelism between the Processing Elements (PEs) leading to extra communication times that remain 'reasonable' comparing with the overall execution time. Figure 7 shows also the variation of the mean PE's Idle time which is the mean over the individuals of a given generation of the Idle times on the two PEs (the FPGA and the processor). This mean time decreases drastically as the GA proceeds, which is also due to the refinement procedure capability to use at best the available gaps in the timing charts of the two PEs.

These timing charts are presented in figure 6 where we can distinguish three FPGA contexts (notice that we have considered partial reconfiguration in this example: the reconfiguration time blocs are of different sizes), the scheduling on the processor and the memory occupation.

5. CONCLUSION

The scheduling/clustering process in the fitness evaluation step of the genetic algorithm is a greedy algorithm that must be tuned to take into account the delay introduced in the executions of tasks in a context due to the allocation of a new tasks in that context. In the genetic algorithm allocation and scheduling are separated: allocation is included in the design space exploration while scheduling allows the evaluation of each solution.

The genetic approach for HW/SW partitioning with a dynamically reconfigurable unit is really effective; it provides an efficient assistance to the designer in the investigation of a balanced architecture and allows various parameters of the architecture to be optimized, such as the number of available *LCs* and *DCs* in the reconfiguration unit, the reconfiguration time per *LC*, the data transfer rates on the buses, the relative speeds of the processor and the reconfiguration unit.

6. REFERENCES

[1] O. Brosch, J. Hesser. ATLANTIS – A Hybrid FPGA/RISC Based Reconfigurable System, Reconfigurable Architectures Workshop, Cancun; Mexico May 2000.

[2] J. R. Hauser and J. Wawrzynek, Garp: A MIPS Processor with a Reconfigurable Coprocessor, Proc. FCCM '97, April 1997.

[3] H. Singh, M. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, E. M. C. Filho, MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications. IEEE Transactions on Computers 49(5): 465-481 (2000).

[4] Excalibur backgrounder, Altera Corporation, June 2000.

[5] Virtex-II Pro data sheet, Xilinx Inc. January 2002.

[6] P. Six, Designing Reconfigurable Networked Appliances using C++, IMEC, Vendor presentation at DAC 2001, june 18-20, Las Vegas.

[7] C. Ebeling, D. Cronquist and P. Franklin. Configurable Computing: The Catalyst for High-Performance Architectures, Proceedings of IEEE International Conference on Application-specific Systems, Architectures and Processors, pp. 364-72, July 1997.

[8] H. Singh, M. H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh and E. M. C. Filho. MorphoSys : An Integrated Reconfigurable System for Data-Parallel Computation-Intensive Applications. University of California, Irvine, CA 1999.

[9] B.P. Dave, G. Lakshminarayana, N. Jha. COSYN: hardware/software co-synthesis of embedded systems, Design Automation Conference, Anaheim, 1997.

[10] S. Bilavarn, G. Gogniat, J. L. Philippe. Area Time Power Estimation for FPGA Based Designs at a Behavioral Level, ICECS, Beyrouth, December 2000, Kaslik, Lebanon.

Functionality/Dependability Co-design in Real-Time Embedded Software

Elisabeth A. Strunk John C. Knight
Department of Computer Science
University of Virginia, USA
{*strunk*, *knight*}*@cs.virginia.edu*

Abstract

As embedded processors become more powerful, demand for quantity and complexity of real-time embedded function increases. This increase in function is in direct opposition to system dependability requirements, since dependability is harder to achieve in larger or more complex applications. While tradeoffs must be made in designing systems to achieve both these ends, system architectures have the potential to ameliorate the problem. In this paper, we advocate the use of survivability as a mechanism for maintaining dependability while attaining the functionality desired by users and discuss research directions needed to realize its benefits.

1. Introduction

The more computing power that embedded processors provide, the more functionality application designers desire to include in real-time embedded systems. This functionality, while offering many possibilities in terms of convenience and safety, can quickly become much more complex than that which humans are able to comprehend. Lack of comprehension introduces opportunities for error, and in systems that must be dependable, such as those described as *safety-critical*, those errors could easily have unacceptable consequences. Furthermore, the resources required to ensure dependability of conceptually simple but extensive functionality might be more than a customer is willing to provide. This sort of tradeoff suggests that functionality/dependability co-design will become an issue of increasing importance.

Limiting the functionality included on a processor is an infeasible option. Not only does it seem a poor choice economically, in practice it is likely to be ignored. Introducing additional complex safety checks can add its own risks due to an increasing incomprehensibility of the overall design [8]. We therefore need technologies that enable designers to include functionality without compromising the critical dependability properties of the system.

Sha has proposed the use of simplicity in dealing with complexity [10], and has shown how this works in control systems. In this paper we introduce the notion of applying *survivability* to embedded real-time systems, extending Sha's concept to the more general framework of arbitrary embedded systems while combining it with principles from the field of critical networked information systems. The goal in designing a survivable embedded system is to develop the system in such a way that it provides crucial functionality during operation even if it is not able to provide non-crucial functionality. By doing so, different dependability requirements can be associated with different functional elements, and, provided the system is designed appropriately, crucial system properties, such as safety, can be maintained even if desirable though non-crucial functionality cannot.

In practice many safety-critical systems are built this way, although with an ad hoc approach. We propose a general, comprehensive approach based on a rigorous definition of survivability. This approach permits a tradeoff between the degree to which functionality is maintained and the cost of system development. Within an application, it also provides a feasible route to the ultradependable implementation of crucial services without demanding the ultradependable implementation of the entire application, a goal that is often technically infeasible using more traditional methods.

The remainder of this paper is organized as follows. Section 2 discusses why survivability is a good strategy for addressing the functionality/dependability co-design issue. Section 3 gives background on survivability from other disciplines and defines it for embedded real-time systems. Section 4 gives a brief example of what this might mean in terms of avionics system regulations. Sections 5 and 6 enumerate future research directions needed to realize the potential of this strategy, and Section 7 concludes the work.

2. Why Survivability?

In order to understand why survivability might be helpful in the context of dependable systems, we first explain

what we mean by dependability. Avizienis, Laprie, and Randell have defined dependability as a collection of six properties [2], and this definition has become a *de facto* standard as well as a *de jure* standard in progress through IFIP Working Group 10.4 [4]. The six properties are:

- **availability**: readiness for correct service,
- **reliability**: continuity of correct service,
- **safety**: absence of catastrophic consequences on the user(s) and the environment,
- **confidentiality**: absence of unauthorized disclosure of information,
- **integrity**: absence of improper system state alterations;
- **maintainability**: ability to undergo repairs and modifications [2].

These properties are in some sense orthogonal to the function specified for the system. For instance, availability is the probability that the system will be able to provide service at time t, i.e., that its functional and real-time requirements are met with a certain probability at the time when it is called. Similarly, meeting reliability, safety, integrity, and confidentiality requirements might be contingent on the system's meeting real-time requirements. Maintainability is chiefly an offline characteristic—and should not be required to be performed in real time even if it is online—and so it will not be addressed here.

Many techniques have been developed to support the engineering of dependable systems. In a broad sense, these techniques fall into three primary areas: fault avoidance, fault elimination, and fault tolerance. Combined with analysis techniques such as fault-tree analysis, event-tree analysis, and failure-modes-and-effects analysis, these approaches to dealing with faults permit useful dependability predictions to be made about specific designs. However, none of these techniques effectively address the dependability problems arising from the growing complexity of embedded software. For example, showing that an entire modern avionics system designed for a commercial air transport meets the FAA's mandated safety goal (see below) is, in general, beyond the present state of the art.

In many cases, much of the functionality included in a system is not directed primarily at system safety. For example, the autopilot system on a commercial air transport could contribute to an accident, but while it is a significant part of the safety case for the aircraft, cessation of its function is unlikely to have catastrophic consequences. This suggests that such a system does not need to be *ultradependable* as much as it needs to be *fail-stop* [9]. Provided the autopilot either works correctly or stops and alerts the pilot, the aircraft is unlikely to come to harm because of it. The complete avionics system for the aircraft needs to be able to operate without the autopilot (and other similar subsystems) so that safety is not compromised even though an emergency landing might be required if the autopilot fails. Such a strategy also reflects the practice of *safe programming* as advocated by Anderson and Witty [1]. In an informal sense, such an avionics system is *survivable* rather than ultradependable.

The notion of survivability has been discussed extensively in the area of networked information systems, and numerous informal definitions of the term have appeared. Knight, Strunk, and Sullivan have presented a more rigorous definition based on the idea that a survivable system is one that complies with its *survivability specification*, a structure that defines the dependability requirements that must be met for different sets of system functionality [4]. We claim that this rigorous definition can be applied in a straightforward manner to the domain of real-time embedded systems with significant benefits, including the provision of a precise framework for functionality/dependability co-design.

Defining survivability in terms of a specification offers significant advantages in systems engineering, in comprehensibility of the necessary dependability properties of a system, and in demonstration of those dependability properties. In terms of systems engineering, the specification permits domain experts to define precisely what functionality is crucial to system dependability and what sorts of timing guarantees must hold on that functionality before the software is designed. This enables system designers to make appropriate tradeoffs. Specifications allow experts to see software at a high level of abstraction, aiding them in understanding what the system as a whole is required to accomplish. The specification can require overall dependability properties of the system as well, defining fault conditions under which those properties must hold. Some formal systems offer considerable benefits through design verification of such dependability properties. Finally, being able to provide crucial functionality in the presence of certain classes of faults means that those faults do not have to be tolerated by the entire system, and demonstrating that only a part of the system tolerates those faults is not only less expensive but also a much more tractable problem.

3. Survivability In Embedded Systems

3.1 Survivability in Critical Information Systems

Survivability is an established research discipline in the realm of critical information systems. The general idea of survivability is that a system will "survive" (i.e., continue some operation), even in the event of damage. The operation it maintains may not be its complete functionality, or it may have reduced dependability properties. It will be some useful functionality that provides value to the users of the

system, including possibly the prevention of catastrophic results due to the system's failure.

Like many terms used in technologies that have not yet matured, however, survivability is not defined with the rigor we need in order to use the concept in reference to specific systems. It has roots in other disciplines that begin to indicate what it should mean in our field; for instance, the telecommunications industry defines survivability as:

> **Survivability**: A property of a system, subsystem, equipment, process, or procedure that provides a defined degree of assurance that the named entity will continue to function during and after a natural or man-made disturbance; e.g., nuclear burst. Note: For a given application, survivability must be qualified by specifying the range of conditions over which the entity will survive, the minimum acceptable level or [*sic*] post-disturbance functionality, and the maximum acceptable outage duration [9].

The network survivability community has attempted to come up with a more directly applicable description, resulting in definitions such as Ellison's:

> **Survivability**: The ability of a network computing system to provide essential services in the presence of attacks and failures, and recover full services in a timely manner [4].

The sundry definitions of survivability vary considerably in their details, but they share certain essential characteristics. One of these is the concept of service that is essential to the system. Another is the idea of damage that can occur, and responding to that damage by reducing delivered function.

Definitions such as these are inadequate because they do not give system developers criteria for determining whether a system is survivable. One cannot determine whether a system is survivable if one is unsure exactly what survivability means. Also, knowing exactly what survivability means in general does not ensure that a straightforward implementation of survivability exists for a particular system. A precise definition is necessary in order to make survivability a meaningful system property.

Knight *et al.* give a definition based on specification: "A system is survivable if it complies with its survivability specification" [4]. They draw on the properties mentioned above and present a specification structure that tells developers what survivability means in an exact and testable way. When followed, this structure will cause them to document what it means for their system to be survivable. It is this perspective we take when defining survivability in embedded systems.

3.2 Requirements of a Survivability Specification

Embedded real-time software has certain similarities to and differences from large networked systems. It has a certain level of intellectual manageability stemming from its less distributed nature. However, it still rarely possesses the qualities of what could be considered a stand-alone application. Embedded systems generally receive input from and send output to other devices; this is the purpose for which such systems are built. These devices can fail just as network nodes can, and such failures must be considered in order to build software that is safe.

Adding to the problem is the inherent functional complexity of many embedded systems. Networked survivable systems are designed to deal with the failure of software on individual nodes; but, when dealing with embedded systems, that software may be in the logical central node. Input and output devices generally are not designed to compensate for failure of the embedded software, and so the software must be designed to survive internal failures.

Furthermore, safety-critical embedded systems are likely to have hard real-time requirements, their dependability requirements are likely to be much tighter than those for networked information systems, and the allowance for duplication much smaller. If one ATM fails, a banking customer can use another; if it takes longer than expected on occasion, this is merely irritating. Only large numbers of such failures can cause significant problems. The smaller scale of embedded systems aids in their analysis, but it makes them more tightly coupled and thus necessitates deeper rigorous analysis.

Finally, many embedded systems require some minimal level of function to ensure safety. For example, software controlling aircraft flight cannot simply terminate in mid-air; there must be some basic level of operation that it is guaranteed to maintain. Networked systems are likely to see a more gradual degradation, with the boundary between effectiveness and ineffectiveness being blurred.

Functionality/dependability co-design decisions, then, must make some compromise between the functionality a user desires to see in a system and the minimal functionality a system must maintain to be considered dependable. We are proposing essentially a framework where the former is the primary function and the latter the backup. This is an incomplete view, however, for three reasons:

- *User expectation.* The user is likely to expect some minimum probability that the full function is provided. Operating exclusively in backup mode is almost certain to be unacceptable.
- *Multiple functionalities.* Usually, there will be more than two major classes of function. If the system must degrade its services, some services are likely to be more valuable than others even if they are not essential for

dependable operation, and the system should continue to provide those services if possible.

- *Value as a function of state.* What is essential for dependable function usually depends on prevailing conditions. In other words, the functionality that is determined to be crucial by domain experts will usually depend upon operating circumstances. As an example, consider an automatic landing system. It could halt and simply alert pilots of its failure if it were not in use (i.e., in standby mode), but if it were controlling an aircraft it would have to ensure that pilots had time to gain control of the situation before halting.

These concepts are used extensively in industrial software development, but the survivability framework puts them on a rigorous footing. This enables them to be analyzed to determine whether they do in fact satisfy the user's needs.

3.3 Defining Survivability

The criteria above are vague, and using them informally will not enable developers to determine whether a system meets them. We therefore must define what we mean by a survivability specification in some rigorous but general way so that, when a specification is built using the framework, it can be analyzed to determine whether it possesses all the necessary information. The form of survivability specification we will use for embedded systems has six elements:

S**: the set of functional specifications of the system.* This set includes the *preferred* specification defining full functionality. It also includes alternative specifications representing forms of service that are acceptable under certain adverse circumstances (such as failure of one or more system components). Each member of ***S is a full specification, including dependability requirements such as availability and reliability for that specification.

E**: the set of characteristics of the operating environment* that are not direct inputs to the system, but affect which form of service (member of ***S) will provide the most value to the user. For example, when developing an aircraft automatic landing system, whether the aircraft has reached decision height (the height below which it is committed to land) might be a member of ***E***. Above decision height it might be safer for the system to pull the plane up before relinquishing control, while below decision height it would leave the aircraft on the course to land. Each characteristic in ***E*** will have a range or set of possible values, in this case *above decision height* or *below decision height*. These values also must be listed.

D**: the set of assignments of value to members of **E** that the system might encounter.* This is essentially the set of all modes (i.e., collection of states) the environment can be in at any particular time. Each element of ***D is some predicate on the environment. ***D*** will not necessarily be equal to the set of combinations of all values of elements in ***E***; for example, if *stage of flight* were also a member of ***E***, then {*below decision height, enroute*} could not be a member of ***D*** because it is an unreachable state of the environment.

V**: matrix of relative values each specification provides to the user.* Each value will be affected both by the functionality contained in the specification and the environmental conditions for which that specification is appropriate. For example, the primary specification might have value 5 under all members of ***D, the alternative of pulling up value 2 when above decision height and 0 below decision height, and the alternative of continuing on current course value 2 when below decision height and 0 above decision height. Quantifying these values is impossible, but using relative values (as is done in economic utility theory) gives the ordering a developer needs to implement the system.

T**: the valid transitions from one functional specification to another.* Each member of ***T represents a transition from one specification to another. It includes the specification from which the transition originates (source specification), the specification in which the transition ends (target specification), and a member of ***D*** defining the environmental conditions under which that transition may occur (the transition guard). The guard enables a specifier to define which transitions are valid under certain circumstances, and the developer can then use ***V*** to decide which target specification is most valuable under those conditions.

P**: the set of probabilities on combinations of specifications*. Each member ***P will be a set of specifications mapped to a probability. The set of specifications is the specifications that provide approximately the same level of functionality, under different environmental conditions. The probability is the probability of a failure occurring in the system when the system is in compliance with one of those specifications (or the single specification, if there is only one in the set for that probability). The probabilities serve to provide a lower-bound guarantee of system operation.

4. An Avionics Example

As an example of how survivability can be applied to embedded real-time systems, we show how it might be used with the current dependability requirements for U.S. commercial avionics systems put forth by the US Federal Aviation Administration (FAA). The FAA categorizes aircraft functionality into three major levels of criticality according to the potential severity of its failure conditions [6]:

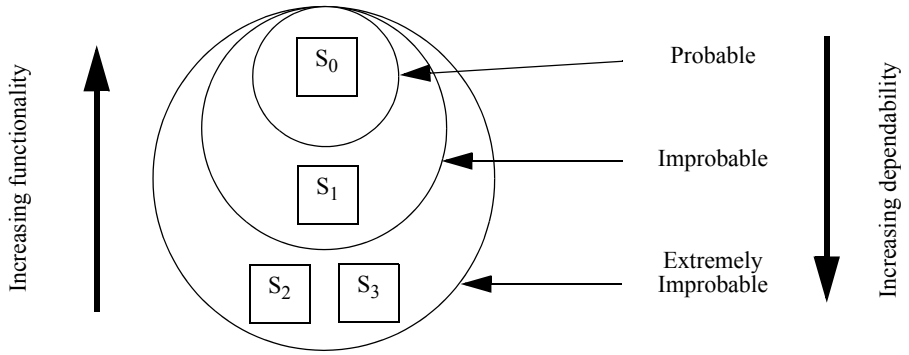

Figure 1. Function and dependability in a survivability specification

Minor: Failure conditions which would not significantly reduce airplane safety, and which involve crew actions that are well within their capabilities...

Major: Failure conditions which would reduce the capability of the airplane or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example, (i) A significant reduction in safety margins or functional capabilities, a significant increase in crew workload or in conditions impairing crew efficiency, or some discomfort to occupants; or (ii) In more severe cases, a large reduction in safety margins or functional capabilities, higher workload or physical distress such that the crew could not be relied on to perform its tasks accurately or completely, or adverse effects on occupants.

Catastrophic: Failure conditions which would prevent continued safe flight and landing.

Failure conditions must have probabilities of not occurring proportional to the potential consequences of their occurrence. "(1) Minor failure conditions may be probable. (2) Major failure conditions must be improbable. (3) Catastrophic failure conditions must be extremely improbable" [6]. "Probable" is defined as "anticipated to occur one or more times during the entire operational life of each airplane"; "improbable" as "not anticipated to occur during the entire operational life of a single random airplane"; and "extremely improbable" as "so unlikely that [the failure condition is] not anticipated to occur during the entire operational life of all airplanes of one type" [6]. Quantifying these definitions leads to probabilities that can be extremely small. "Extremely improbable", for example, corresponds to a failure rate of 10^{-9} per hour of operation.

In our automatic landing system example, we will assume four functional specifications, as shown in Figure 1. The first, *primary*, specification (S_0) will have all of the functionality the user desires for the system. The consequences of any failures will be minor because, if they have the potential to be more severe, the system can transition to one of the other three specifications. Therefore, any failure in the primary specification may be "probable".

The first alternative specification (S_1) will have much of the functionality desired by the user, but some desirable yet unnecessary functionality removed. For example, the system might have to follow the step-down altitude clearances for the runway to descend at the proper rate rather than using the glideslope. All failures in this specification must be "improbable"; its functionality is important enough that frequent interruptions could have adverse consequences. However, none of it need be "extremely improbable" because any failures with potentially catastrophic consequences will cause a transition to a different alternative specification (S_2 or S_3).

S_2 and S_3 are the specifications that have very high dependability requirements. We will let S_2 be the specification requiring the aircraft to pull up and alert the pilot on system failure and S_3 be the specification requiring that the system continue on its current course when alerting the pilot if the system fails. They contain the minimum functionality necessary to maintain safe operation of the system. Any non-masked failure of either of these specifications—such as failure to alert the pilot that the system has malfunctioned and the pilot is now in control—must be "extremely improbable", as they are designed to include only the system functionality whose failure could have catastrophic consequences.

A major factor that the FAA guidelines for flight systems does not address is changes in what system dependability requirements might be based on environmental circumstances. Whether the system transitions to S_2 or S_3 on a failure of S_1 depends on whether the aircraft is above or below decision height at the time of the transition. The

new probability requirement, then, would be that a failure of S_2 above decision height is "extremely improbable", and a failure of S_3 below decision height is "extremely improbable". In some cases the environmental conditions might change, and a transition between specifications appropriate to different conditions must occur in order to keep the system operating with the optimal functionality.

Finally, it is possible that the system could recover from a failure that forced it to transition to a lower level of functionality. For instance, the aircraft might operate under specification S_1 while the glideslope transmitter is reset to recover from a transitory error, then transition back to specification S_0. A specification structure for a survivable system should provide for this as well.

In considering this example, it is important to note the difference between this discussion and the normal approach that is taken by regulating agencies such as the FAA. The example illustrates the use of survivability and the association of different dependability requirements with different functionalities within the survivability specification. This contrasts with the current situation in which the entire system would be assigned a certification level by the FAA even though much of the functionality would not require that level of dependability.

5. Research Challenges in Functionality/ Dependability Co-design

The notion of functionality/dependability co-design is complex yet potentially very fruitful. There are numerous remaining issues that need to be resolved before the concept as manifested in the notion of survivability applied to embedded systems can be used routinely. In this section, we present some of the research issues.

- *Defining confidence for different software dependability levels.* Current software practice defines measurements for determining confidence in software, but these measurements tend to be based on intuition and can vary widely across application domains and regulatory agencies. The research community has argued that engineers cannot be confident that current software will function at ultradependable levels. The aim of functionality/ dependability co-design of software is to make this a tractable problem. Powerful but potentially expensive mathematical analysis such as model checking and putative theorem proving can be applied to critical parts of a system while leaving most of a company's standard development processes intact. Testing might be able to show statistically that certain functionality meets its less stringent dependability requirements. Determining whether ultradependable functionality meets its requirements still is not, however, a statistical problem [3]. The computing research community is now presented with the challenge of deciding what metrics or processes give us confidence that software conforming to them indeed achieves stated dependability goals. This includes determining what sorts of analysis apart from testing can show in a defensible way that real-time constraints are met.
- *Validation of survivability specifications.* Potentially, the most valuable benefit from building survivable specifications is a significant reduction in complexity for the most dependable portions of a software system. This reduced complexity can facilitate inspection and validation of the system. It can also help application domain experts determine what properties are required in order to ensure the system is dependable in the common sense, i.e., that its users can depend on its performance. The composite survivability structure, however, is more complex than any individual specification. Inspection and other validation methods are needed to ensure experts' understanding of the overall function of the system.
- *Analysis of transitions between specifications.* The powerful advantages gained through applying survivability require that certain properties of transitions between specifications be guaranteed. The specifier must be able to show that the system is able to transition to an alternative specification without violating the system's overall dependability requirements if he is to claim that the alternative specification can stand in for the primary specification in terms of dependability analysis. A major part of this is determining what the real-time requirements on the transitions themselves must be, and how they relate to the individual specifications under which the system will operate.
- *Determining criteria for violation of dependability requirements.* In typical dependable systems, violation of dependability requirements is strictly disallowed and so determining when a violation occurs is unnecessary. Because survivability permits some leeway in this aspect, decidable criteria for these properties must be defined. Availability and reliability, for example, are defined probabilistically, but their probabilities are defined over some period of time that in the current framework is unbounded. In practice, real-time requirements impose a bound whose interaction with other requirements must be assessed in order to determine when dependability properties are violated.
- *Establishing composite system properties.* A survivable system is broken into separate survivability specifications with separate probabilities, but these specifications come together to form the overall system specification. It is the properties of this overall specification in which the user is interested for purposes of

determining delivered value. For example, proving that a failure of the automatic landing system to alert pilots when it malfunctions is extremely improbable does not show that the autopilot will function as desired while enroute. It is these properties that must be studied to optimize decisions in functionality/dependability co-design.

6. Research Challenges in Hardware/ Software Co-design of Survivable Systems

Survivability as a system concept has impact beyond functionality/dependability co-design. It is quite possible for it to be exploited to assist in the process of hardware/ software co-design because it provides a much more flexible software architecture than is found in current non-survivable designs. Also, effective hardware/software co-design can magnify the benefits of survivable systems. We discuss in this section several challenges in the area of hardware/software co-design as it relates to survivability.

- *Distribution of survivable software over available hardware.* Faults in the software in a survivable system can originate in damage to its underlying hardware, and reconfiguration can be initiated by hardware failure. We have presented a specification framework aimed at aiding functionality/dependability co-design of the software portion of the system. Further research is needed in areas such as determining where critical pieces of function should reside; choosing how best to overlay the software on available hardware, including analyzing what sort of capability the hardware has in terms of carrying out software function in real time; and determining what sort of code replication is most efficient in terms of space and time requirements.
- *Analysis of tradeoffs between hardware cost and software function in survivability levels.* Survivability can be employed to avoid certain hardware-based solutions to dependability issues. Since convenient but noncritical functionality does not have to be dependable, a simpler but more fragile hardware implementation of it could be built. This would reduce development cost of the system, but at the expense of user convenience or satisfaction. Deciding what function belongs in each specification, which then dictates the dependability level at which that function must be implemented (including what sort of timing characteristics must be guaranteed of it), is an area that holds great potential for economic research in system development.
- *Refining software's role in system risk analyses.* Attempting a more rigorous definition of what confidence means for software at certain dependability levels implies that a new definition of what confidence in a system means is also needed. For instance, if testing is to give a certain amount of confidence in software, one might question what hardware configuration is necessary to carry out that testing. Alternatively, if certain classes of software function are claimed to benefit from design diversity, hardware might play a practical role in this by providing options such as writing different versions of software for different hardware platforms. Finally, software might be implemented on hardware in such a way that the hardware prevents certain interactions between components. How to construct a risk analysis of the overall hardware/software system and what design decisions might facilitate that analysis are further promising future research areas.
- *Implementation feasibility.* Software specification languages can express functions not implementable in any programming language, and dependability levels can be required that are impossible to achieve with finite hardware. For example, it is possible to specify an oracle to the halting problem that has an availability of 1 and returns a result within some specified time bound. The question of whether a feasible implementation of a specification exists must be answered for any system, but survivable systems can encompass a broader range of functionality and dependability requirements since they allow the tradeoff to be made in a gradual manner. Research in efficient iteration between specification and design stages of hardware and software development could greatly increase the efficiency of processes for building survivable systems.

7. Conclusion

As embedded real-time systems become larger and more complex, issues in functionality/dependability co-design will become increasingly important. Addressing these issues becomes much easier with the observation that much of the desired functionality is not essential for dependable system operation. Separating safety-critical function and designing it for ultradependability while implementing the residual function in a less dependable way is a promising solution. This strategy is currently employed in the domain of networked information systems under the name *survivability.* We have discussed several definitions of survivability, outlined a rigorous definition as applied to embedded real-time systems, and presented future directions for the research community to explore in realizing its benefits.

Acknowledgments

We thank William Greenwell for his assistance in constructing our avionics examples. This work was funded in

part by NASA Langley Research Center under grants numbered NAG-1-2290 and NAG-1-02103, in part by the Defense Advanced Research Projects Agency under grant N66001-00-8945 (SPAWAR), in part by the U.S. Air Force Research Laboratory under grant F30602-01-1-0503, and in part by Microsoft. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the U.S. Air Force, or the U.S. Government.

References

[1] Anderson, T., and R. W. Witty. "Safe programming." *BIT*, 18:1-8, 1978.

[2] Avizienis, A., J. Laprie, and B. Randell, "Fundamental Concepts of Computer System Dependability," IARP/IEEE-RAS Workshop on Robot Dependability: Technological Challenge of Dependable Robots in Human Environments, Seoul, Korea, May 2001.

[3] Butler, R. W., and G. B. Finelli. "The Infeasibility of Experimental Quantification of Life-Critical Software Reliability." ACM SIGSOFT '91 Conference on Software for Critical Systems, New Orleans, LA, December 1991.

[4] See http://www.dependability.org/wg10.4/

[5] Ellison, B., D. Fisher, R. Linger, H. Lipson, T. Longstaff, and N. Mead. "Survivable Network Systems: An Emerging Discipline." Technical Report CMU/SEI-97-TR-013, Software Engineering Institute, Carnegie Mellon University, November 1997.

[6] Federal Aviation Administration Advisory Circular 25.1309-1A, "System Design and Analysis."

[7] Knight, J. C., E. A. Strunk and K. J. Sullivan. "Towards a Rigorous Definition of Information System Survivability." DISCEX 2003, Washington, DC, April 2003.

[8] Perrow, C. *Normal Accidents: Living with High-Risk Technologies*. Princeton University Press: Princeton, New Jersey, 1999.

[9] Schlichting, R. D., and F. B. Schneider. "Fail-stop processors: An approach to designing fault-tolerant computing systems." *TOCS* 1(3):222-238.

[10] Sha, Lui. "Using Simplicity to Control Complexity." *IEEE Software* 18(4):20-28.

[11] U.S. Department of Commerce, National Telecommunications and Information Administration, Institute for Telecommunications Services, Federal Standard 1037C.

A Configuration Manager for Embedded Real-Time Applications

Roman Gumzej
University of Maribor
Faculty of Electrical Engineering and Computer Science
Smetanova 17, SI-2000 Maribor, Slovenia
roman.gumzej@uni-mb.si

Wolfgang A. Halang
Fernuniversität
Faculty of Electrical and Computer Engineering, 58084 Hagen, Germany
wolfgang.halang@fernuni-hagen.de

Abstract

Configuration management in co-design of distributed real-time systems and applications is considered. In particular, an idea for modeling the dynamical behaviour of embedded real-time applications with collections of timed hierarchical state diagrams is indicated. By translation to program code, schedulable objects are formed, representing the applications' semantics. During execution, the real-time operating system, being a part of a configuration management program, must ensure that the applications' system requests are serviced and their timing restrictions are met. A distinction is made between active and passive objects. Active objects communicate among each other and manage the passive objects and their co-operation. An active "super" object, called Configuration Manager, is foreseen for configuration management at each node in a distributed system, and to handle the active objects and their co-operation.

1. Introduction

In the last ten years the co-design discipline has moved from an emerging discipline to a mainstream technology [13]. Contemporary co-design methodologies have emerged partly from previous projects (e.g.: [2, 3, 4]) and as part of the development of a novel standard method for designing embedded (real-time) systems – UML [5]. For verification of the designs partly formal methods and proofing are used (e.g.: [10]), whereas on the other hand mostly co-simulation is used for their validation. Additions of domain-specific layers in system design (e.g.: security related issues [11]) indicate that the completeness and maturity of co-design methods has not yet been reached. Especially the target platform independence and fault tolerance options seem to be a problem due to the large variety of hardware components and existing applications, used in embedded systems. In the scope of the article a Configuration Manager (CM) is considered, whose function is amongst others to serve as a hardware abstraction layer.

Two types of objects, which represent the functional components of an automation application, have been identified, namely active and passive ones [1]. An active object communicates with other active objects and manages the resources and lifecycles of its passive objects. The passive objects contain functionality, but do not act unless triggered by an active object. In general, one may say that the active objects represent tasks, whereas the passive objects represent procedures and functions with separate data spaces.

Active objects may or may not be dependent on the simultaneous presence of an operating system. In environments without operating systems one of them must take over the duties of initialisation and resource management. Under the supervision of an operating system, on the other hand, the execution of active objects is scheduled on the basis of their scheduling conditions, and the resources are managed for them by the operating system. In this case, active objects have the responsibility to manage their passive objects, only.

In this article the separation of design concerns is done on three levels of software (architecture) modelling: software to hardware mapping, software architecture and program task modelling. Each of them is managed by another entity. The approach is part of the Specification PEARL methodology for co-design and co-simulation of embedded real-time systems [6].

In the main body of this article, the architectural mod-

elling is presented, followed by the description of the task model and the functionality of the configuration manager. To draw a conclusion, their rôle for building the executive model and for Co-Simulation is discussed.

2. System Architecture and Program Modeling

In the mentioned methodology, hardware and software architectures are described separately. A hardware architecture is composed of processing nodes, named stations, whose descriptions also mention their components with their properties. A software architecture is organised in the form of collections of program modules, the latter combining program tasks, functions and procedures of the application software. The collection is the unit of software mapped onto stations, i.e., at any time there is exactly one collection assigned to run on a station. Thus, the collection is also the unit of dynamic re-configuration.

The merge of different models with configuration manager and simulator structures is depicted in Fig. 1. To manage collections a Configuration Manager (CM) is required, which is situated between the real-time operating system, if any, and the applications. Its rôle is to function as (1) a hardware abstraction layer, (2) a hardware/software interface, and (3) as an "Inter-Collection" co-operation agent.

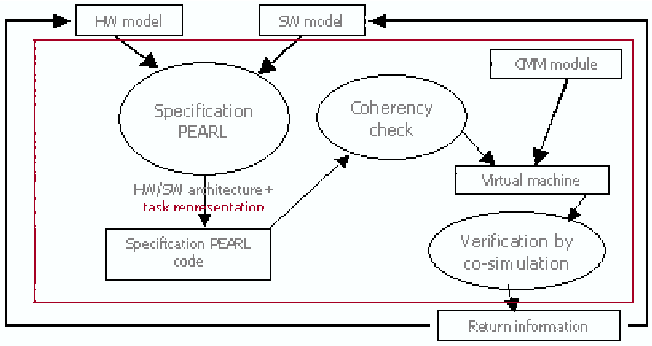

Figure 1. Interrelations of hardware/software models

2.1. Application Program Model

According to [8], the computational model of most applications running in real time can be written in form of the following "equation":

Real-time program model = Dataflow model + State automaton + Timing limitations

The tasks of a system represent the processes of the running system. Their main properties are trigger conditions and timing limitations as well as being part of a certain collection. This information is sufficient to build a coarse program model, but it is not enough to determine its functionality. Therefore, timed state transition diagrams have been introduced, representing their control and data flows as well as their synchronisation and inter-communication by calls to the configuration manager or real-time operating system of the station, resp., executing the tasks. They are a variation of the Shaw's Communicating state machines [12] and are described in detail in [7]. A Timed State Transition Diagram (TSTD) represents a single task, and has the semantics of a timed hierarchical state machine.

2.2. Active and Passive Objects

The tasks, modeled in a top-level TSTD description, form active objects, which are assigned global properties such as trigger conditions, priorities or deadlines, whereas the sub-level super-state defining diagrams represent passive objects, which share the global task properties of their native tasks.

The idea, shown in Fig. 2, to represent timed state transition diagrams as tasks is the following: the semantics of a TSTD diagram are represented by the Main() method of a task object (considering the existing mechanisms for the translation of state charts to program code). Then, the constructor/destructor methods represent initialisation/finalisation actions.

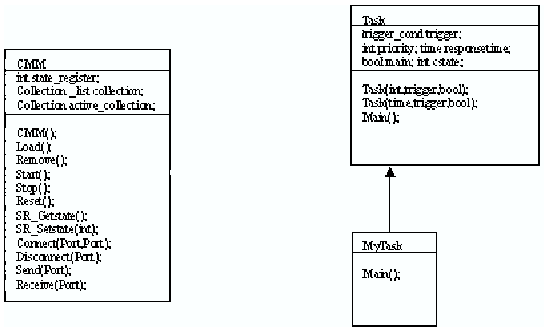

Figure 2. Proposal of the configuration manager (CM) object

Hence, from this emanates the idea to represent "tasks" or "threads" as "schedulable objects", which are executed under supervision of a configuration manager. Its functionality is detailed in the next section. The transitions between top-level states are performed in the Main() method, while the sub-charts of composite states are represented by Main()

methods of its sub-classes, having the same names as the composite states. After being instantiated by the super-class, their execution continues within the local enumeration of states, while the global enumeration is continued upon returning to the same point after the corresponding end state (return statement) is reached.

The "schedulable object" classes form class libraries named "collections". The configuration manager (CM) class constitutes a separate library, being loaded in full or partially to each station (processing node). The CM object initially loads and manages the collection objects, containing schedulable objects (tasks), activating the collection objects, which correspond to the station state. It also establishes the necessary communication channels (ports) for inter-collection/inter-task communication and co-operation. The CM object is considered an active object, located above the other active objects contained in collections.

3. Configuration Management

The executive program at each station is the configuration manager object, combining the functionality of a configuration manager and a real-time operating system. Initially, it loads the collections of task objects and activates the initial collection by triggering the latter's initialisation task objects. In stations without a real-time operating system this functionality is provided by the configuration manager, whereas otherwise the CM represents a front end to the operating system functions, i.e., an abstraction of a real-time operating system.

Besides local execution, the CM is also responsible for communication with other stations, and for co-operation among the tasks of the same collection. Hence, it must establish port-to-port connections through the interfaces of the station, and appropriate local port connections for inter-task communication. Synchronisation and system service requests are serviced on the same station, in case the station is configured to run the real-time operating system part of the CM. Otherwise, these requests are forwarded to the appropriate station.

The CM has information on station parameters and SW/HW mapping. It represents an interface to the hardware of the system and, since programmed in C, can be translated for various types of processors and peripheral devices. Application programs consider the CM as an abstraction layer of hardware and operating system, while accessing the configured hardware devices. Hence, the functionality of the CM must provide the following services:

- initial loading of program collections,
- memory management (allocation, access, de-allocation),
- inter-station communication through their interfaces (also supporting rôle-dependent high-level communication protocols),
- internal station-state monitoring and reaction to state changes (configuration management), and
- controlled shut-down of stations.

3.1. Functions and Properties of the Configuration Manager

As indicated in Fig. 3, the CM has to react to three different kinds of stimuli: (1) messages between stations, (2) service calls to a real-time operating system, and (3) proper CM requests.

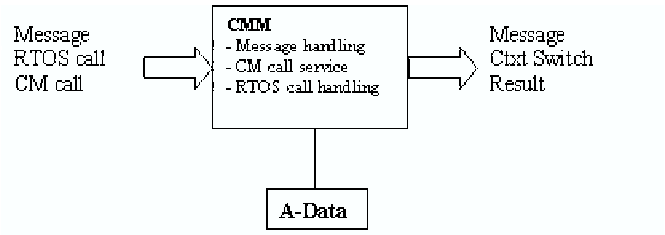

Figure 3. The configuration manager as a black box

The application programming interface (API) of the CM has the following functions:

Configuration management:

- Cm$Init – to load the initial software configuration,
- Cm$Load – to add a collection to the current software configuration,
- Cm$Remove – to remove a collection from the current software configuration,
- Cm$Start – to start execution,
- Cm$Stop – to shut down a station, and
- Cm$Reset – to re-start a station with an initial configuration.

Station state monitoring:

- Cm$SR_Getstate – to retrieve the current state of a station, and
- Cm$SR_Setstate – to change the current state of a station.

Inter-station communication:

- Cm$Port_Connect – to create a port (establishing point-to-point connection),
- Cm$Port_Disconnect – to disconnect an established connection,
- Cm$Port_Send – to transmit a message through a connection, and
- Cm$Port_Receive – to receive a message through a connection.

The configuration manager's API is independent on the station type, but its message routing is affected by the rôle a station plays in a distributed system which is determined by the type of the station. Hence, in order to be able to correctly "understand" and process the messages it receives, the CM has to "know" the rôle its station plays in the overall system architecture.

The connections are established through ports of the SW architecture and associated devices of the HW architecture. The attributes of ports represent the communication parameters (smallest package, protocol, etc.) and routing parameters (VIA/PREFER). The routing affects the way the HW communication devices are used. VIA determines the exact line to use, while the PREFER attribute is usually assigned to the most trusted line in a list.

In asymmetrical architectures direct calls to CM and real-time operating system functions are not possible. Hence, an additional function, *Cm$System(parameters)*, is needed for appropriate system request messages to the CM or the real-time operating system, respectively. If a station is configured with no real-time operating system, the parameters of system requests are routed to the appropriate station. The parameters of such system requests are translated to system calls in accordance with the CM's API.

The properties, specified for the hardware and software architectures, form the base of an architecture data structure, through which the CM (its configuration management and real-time operating system parts) is parameterised. This information is taken into account when executing the system services, listed above.

Between the hardware and the software architecture parameter structures exist the following relations:

- for a station one or more states are defined, and a collection is associated with each of them,
- each station knows its rôle in the distributed system it belongs to (based on its type),
- each collection knows the station it belongs to, and
- each collection knows its members and interfaces to other collections and within the collection.

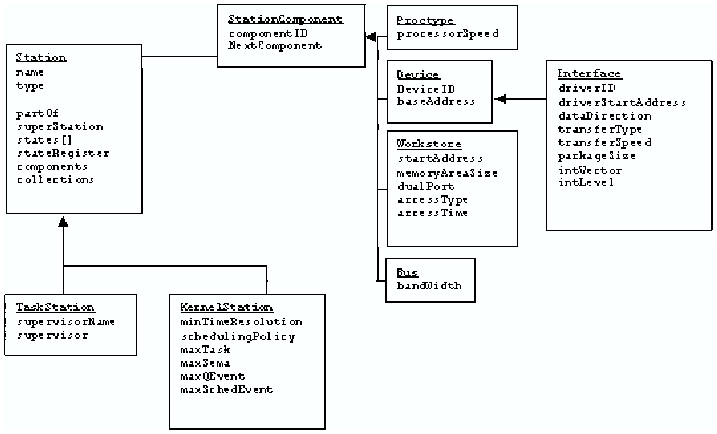

Figure 4. Hardware architecture parameters

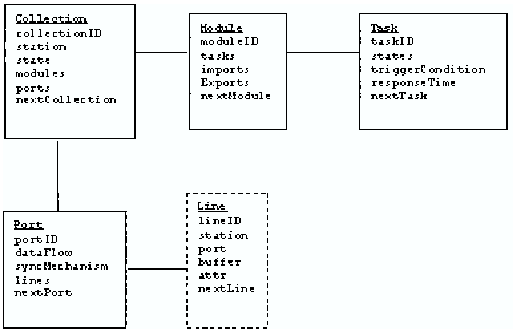

Figure 5. Software architecture parameters

The hardware and software architecture properties (cp. Fig. 4 and 5) are used as "architecture parameters" to parameterise the configuration manager. Fig. 6 presents the structure of the entire CM module.

4. The Rôle of the CM in Co-simulation

Since the co-design methodology Specification PEARL also includes model checking by co-simulation, the computation model for the co-simulation also addresses CM functions. In this case the global framework of the system is built around the CM, based on the system specification.

Our simulation method is based on co-simulation with EDF (Earliest-Deadline First) scheduling and time boundaries. It is primarily meant for checking the timing properties of the modelled system in order to determine its feasibility.

The designed system model is transformed into an internal representation (parameterised by the architecture pa-

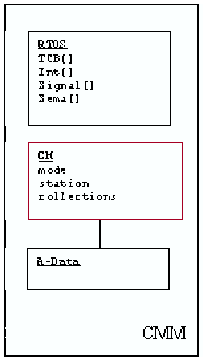

Figure 6. Structure of the configuration manager

rameters in Fig. 4 and 5) for co-simulation, whose primary result is a successful execution or failure, whereas the secondary result is its execution trace, from which additional information on bottlenecks, unreachable states and resources, etc. in the designed prototypes, can be extracted.

For a successful simulation run, it is assumed that the system model is complete and consistent. Intermediate checks must be done during the design of the system architecture to ensure this:

- Completeness check (all components, which are referred to, are present and fully described);
- Range and compatibility check (some parameters of components may be range checked to discover obvious mistakes and to discover possible incompatibilities in the parameters of the associated components);
- Software to hardware mapping check (every collection must be mapped onto a stations' state considering its and / or its supervisors' resources; e.g.: the number of tasks, which may be handled by a single RTOS processing node is limited).

Co-simulation is done based on the following presumptions:

- there is only one global simulation clock in the system and all Real-Time Clocks (RTCs) are synchronised with it,
- the time events relate to the corresponding STATIONS' RTC,

The CM has an important role in the co-simulation (Fig. 7). It maintains the RTC of the station and represents the HW abstraction layer for the executing application during the co-simulation as well as during the execution on the target architecture.

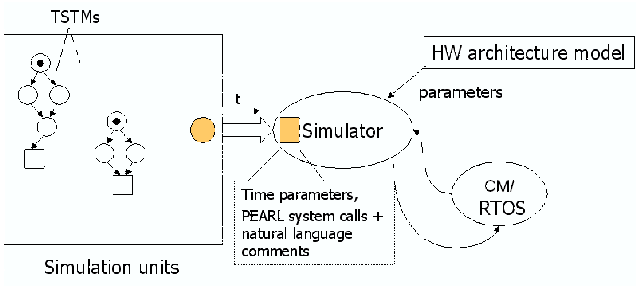

Figure 7. The course of simulation

5. Conclusion

To address the complexity of distributed real-time embedded applications, the Specification PEARL methodology has been devised. By employing a CASE tool and semantic methods for model checking and feasibility estimation, the methodology has gained the desired practical usability. Since it incorporates constructs and mechanisms to ensure safety and fault tolerance, it stands out among contemporary design methods for embedded real-time systems.

The clear definition of the configuration manager (active) object's responsibilities and functions is crucial for the Specification PEARL methodology, its syntax and semantics, to produce executable results. Without such a configuration management program it would merely represent another description and specification language and design method. With its help, and by ensuring portability of the configuration manager's code, the desired range of hardware target platforms can be reached, on which real-time application programs, written in PEARL or C++, using CM real-time operating system service calls, are to run.

6. Acknowledgements

This article presents the results of the research project Holistic Embedded Control Systems Design (Z2-3493), being financed in part by the Slovenian Ministry of Education, Science and Sport.

References

[1] G. Agha. The Structure and Semantics of Actor Languages. J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, Foundations of Object-Oriented Languages, pp. 1-59, Springer Verlag, 1991.

[2] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki and B. Tabarra.

Hardware-Software Co-Design of Embedded Systems: The POLIS Approach, Kluwer Academic Publishers, 1997.

[3] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vinticelli. Metropolis: An Integrated Electronic System Design Environment. IEEE Computer, Vol. 36, No. 4. April 2003.

[4] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi, and M. Poncino. SystemC Cosimulation and Emulation of Multiprocessor SOC Designs. IEEE Computer, Vol. 36, No. 4. April 2003.

[5] B. P. Douglas. Real-Time UML, Addison Wesley, 1998.

[6] R. Gumzej. Embedded System Architecture Co-Design and its Validation. Doctoral thesis, University of Maribor, Slovenia, 1999.

[7] R. Gumzej, Matjaž Colnarič. The Representation of PEARL Tasks as Timed State Transition Diagrams. Proceedings of WRTP'03 27^{th} IFAC/IFIP/IEEE Workshop on Real-Time Programming, Zielona Gora, Poland, May 2003.

[8] A.K. Mok. Towards Mechanization of Real-Time System Design. Foundations of Real-Time Computing: Formal Specifications and Methods. Kluwer Academic Publishers, 1991.

[9] Full PEARL (DIN 66253, Part 2). 1982.

[10] K. Richter, M. Jersak, and R. Ernst. A Formal Approach to MpSoC Performance Verification. IEEE Computer, Vol. 36, No. 4. April 2003.

[11] P. Schaumont and I. Verbauwhede. Domain-Specific Codesign for Embedded Security. IEEE Computer, Vol. 36, No. 4. April 2003.

[12] A. C. Shaw. Communicating real-time state machines. IEEE Trans. Software Engineering, Vol. 18, No. 9, pp. 805-816, 1992.

[13] W. Wolf. A Decade of Hardware/Software Codesign. IEEE Computer, Vol. 36, No. 4. April 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

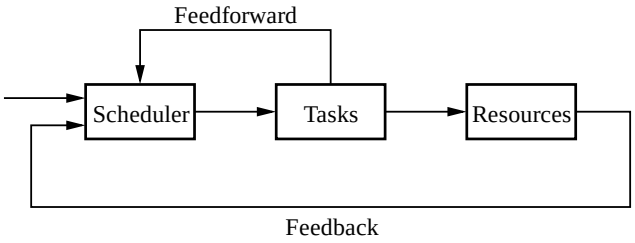

Figure 1 A general feedback scheduling structure. The resources are distributed among the tasks based on feedback from the actual resource use. The tasks can be use feedforward to notify the scheduler about changes in their resource demands.

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 $\rho(\Phi_1) \leq$

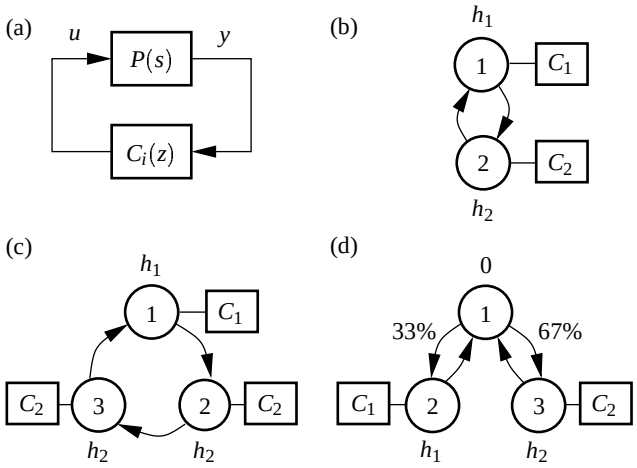

Figure 2 Jitterbug model of system with varying sampling intervals: (a) signal model, (b) timing model with repeating intervals h_1, h_2, (c) timing model with repeating intervals h_1, h_2, h_2, and (d) timing model with random sequence of sampling intervals.

$1, \rho(\Phi_2) \leq 1$, 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.

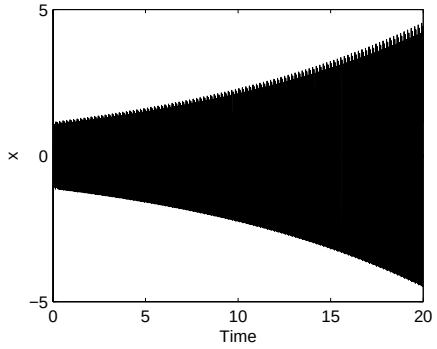

Figure 3 Simulation of with switching sequence $h_1 h_2 h_2$. The result is an unstable system.

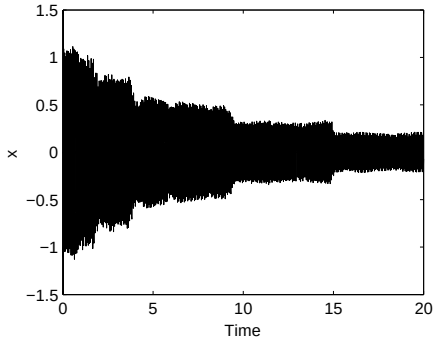

Figure 4 Simulation of with a random two-point distribution. The result is a stable system.

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.

Real-time and delay-dependent control co-design through feedback scheduling

Daniel Simon, Olivier Sename[1], David Robert and Olivier Testa
INRIA Rhône-Alpes, Équipe POP ART, 655 avenue de l'Europe, 38330 Montbonnot, France
{Daniel.Simon, Olivier.Sename}@inrialpes.fr

Abstract

Control systems are often designed using a set of cooperating periodic modules running under control of a real-time operating system. As a correct behaviour of the closed-loop controller requires that the system meets timing constraints like periods and latencies, the structure and timing parameters of the controller must be set according to control requirements. To take into account timing uncertainties, e.g. due to preemption, a delay-dependent feedback loop has been designed; the scheduling controller regulates the resource utilisation according to the estimated execution times. The aim is here to control the Quality of Service consisting of computing resources utilisation and quality of control contribution (evaluated here through the tracking error and the robustness w.r.t to control delay). The actuators are the tasks periods and a H_∞ control approach provides robustness w.r.t. modelling errors. A simulated example with two pendulums enlightens the approach.

1 Introduction

Most feedback control systems are essentially periodic, where the inputs (reading on sensors) and the outputs (posting on actuators) of the controller are sampled/hold at a predefined rate. While basic digital control theory deals with systems sampled at a single rate, it has been shown, e.g. [19], that the control performance of a non-linear system like a robot can be improved using a multi-rate controller : some parts of the control algorithm, e.g. updating parameters or controlling slow modes, can be executed at a pace slower than the one used for fast modes. In fact, a complex system involves sub-systems with different dynamics which must be further coordinated. Therefore the controller must run in parallel several control laws with different sampling rates inside a hierarchy of more or less tightly coordinated layers.

Digital control systems are often implemented as a set of tasks running on top of an off-the-shelf real-time operating system (RTOS) using fixed-priority and preemption. The performance of the control, e.g measured by the tracking error, and even more importantly its stability, strongly relies on the values of the sampling rates and computing latencies[1] (delays) [2]. Therefore it is essential that the implementation of the controller will respect an adequate temporal behaviour to meet the expected performance. However, control science and real-time computing most often evolved independently leading to misconceptions and lack of efficiency in the implementation of real-time control algorithms.

This paper deals with real-time scheduling and control co-design. First, as in [5], we are interested in feedback scheduling, i.e. we aim to adjust on-line the sampling periods of the controllers in order to meet the computing resource requirements. Thus, the process control parameters are changed on-line according to the required sampling periods. Furthermore the process control design takes into account control delays that are unavoidable in real-time control [3]. We here provide a state feedback control law for discrete systems with control delay, the parameters of which depend on the sampling period of the controllers.

The outline of the paper is the following. Section 2 recalls the state of control theory w.r.t. sampling and delays while section 3 enlightens some deficiencies between popular real-time paradigms and control requirements. Section 4 provides guidelines for a static implementation of closed-loop control systems. Section 5 presents the considered architecture for feedback scheduling and the Quality of Control (QoC) specifications. An illustrative example of a multi-tasks control system is presented in section 6 where the advantage of the proposed methodology is emphasised. The paper ends with some concluding remarks and further research directions.

2 Control and timing

Once a control algorithm has been designed, a first job consists in partitioning it into tasks and then in the assignment of timing parameters, i.e. periods of tasks and I/O latencies, so that the controller's im-

[1] on leave from Laboratoire d'Automatique de Grenoble, ENSIEG-BP 46, 38402 Saint Martin d'Hères Cedex, FRANCE

[1] the latency is the delay between the instant of a measure q_n on a sensor and the instant when the control signal $U(q_n)$ is sent to the actuators.

plementation satisfies the control objective. Control theory for linear systems sampled at fixed rates, possibly with *fixed and known pure time* delays, has been established a long time ago. Some more recent results deal with varying or unknown delays or sampling rates in control loops, still in the framework of linear systems, e.g. [13], [9]. Unfortunately most real-life systems are non-linear. The extrapolation of timing assignment through linearising often gives rough estimations of allowable periods and latencies or even can be meaningless. Thus slicing the control algorithm and setting adequate values for the timing parameters rapidly falls into case studies based on simulation and experiments. Such case-studies may benefit from off-line control/scheduling co-design, e.g. [16] using off-line iterative optimisation, to compute an adequate setting of periods, latencies and gains resulting in a requested control performance according to the available computing resource and implementation constraints.

Control systems are often cited as examples of "hard real-time systems" where deadline violations are strictly forbidden. In fact experiments show that this assumption may be false for closed-loop control. Any practical feedback system is designed to obtain some stability margin and robustness w.r.t. the plant parameters uncertainty. This also provides robustness w.r.t. timing uncertainties: closed-loop systems are able to tolerate some amount of sampling period and computing delays deviations, jitter and occasional data loss with no loss of stability or integrity, but only disturbances. The hard real-time assumption should be changed for a "weakly hard" one, where absolute deadlines would be replaced by statistical ones, e.g. the allowable output jitter compliant with the desired control performance. Even if computing such statistics is out of the scope of current control theory, this intrinsic robustness of closed-loop controllers gives an additional degree of freedom which can comply with Quality of Service (QoS) computation and flexible scheduling design.

3 Traditional real-time models

Usually, real-time systems are modelled by a set of recurrent tasks assigned to one or several processors and a worst case response times technique is used to analyse fixed-priority real-time systems as initiated in [10]. Well known scheduling policies, such as Rate Monotonic for fixed priorities and EDF for dynamic priorities [23], assign priorities according to timing parameters, respectively sampling periods and deadlines. They are said "optimal" as they maximise the number of tasks sets which can be scheduled with respect of deadlines, under some restrictive assumptions. They are very popular but they must not be used blindly.

They hardly take into account precedence and synchronisation constraints which naturally appear in a control algorithm. The relative urgency or criticality of the control tasks can be unrelated with the timing parameters. Thus, the timing requirements of control systems w.r.t. the desired control goal expressed as a performance index do not fit well with scheduling policies purely based on schedulability tests. It has been shown through experiments, e.g. [3], that a blind use of such traditional scheduling policy can lead to an inefficient controller implementation; on the other hand a scheduling policy based on application's requirements, associated with a right partition of the control algorithm into real-time modules may give better results.

Another example of unsuitability between computing and control requirements arises when using priority inheritance or priority ceiling protocols to bypass priority inversion due to mutual exclusion, e.g. to ensure the integrity of shared data. While they are designed to avoid dead-locks and minimise priority inversion lengths, such protocols jeopardise at run-time the initial schedule which was carefully designed to meet control requirements. As a consequence latencies along some control paths can be increased to values far over their desired value, leading to a poor control performance or even instability.

Design and off-line schedulability analysis rely on a right estimation of the tasks worst case execution time. Even in embedded systems the processors use caches and pipelines to improve the average computing speed while decreasing the timing predictability. Another source of uncertainty may come from some pieces of the control algorithm. For example, the duration of a vision process highly depends on incoming data from a dynamic scene. Also some algorithms are iterative with a badly known convergence rate, so that the time before reaching a predefined threshold is unknown (and must be bounded by a timeout). Finally, in a dynamic environment some control activities can be suspended or resumed and control algorithms with different costs can be scheduled according to various control modes leading to large variations in the computing load. Thus real-time control design based on worst case execution and strict deadlines inevitably leads to a low average usage of the computing resource.

4 Control systems and off-the-shelf RTOS

The controller is most often implemented as a set of modules, each of them encoding a piece (function) of the control algorithm. At run-time the control modules are basically periodic and are scheduled using the basic features of the RTOS, i.e. priority based preemption and synchronisation primitives.

4.1 Priority assignment

A control system for a robot, and more generally for process control, can be split into several calculation paths [1] : the direct control path computes control set-points from tracking errors and must run with a small period and a low latency to ensure the process stability. The respect of the above timing constraints is critical w.r.t. the control performance, i.e. this part of the controller has a high relative urgency. Others urgent activities are critical tasks which monitor the system's activity and which are triggered by the detection of deviations from the nominal behaviour. In fact some of these error recovery procedures can be better triggered by interrupts rather than by periodic polling.

Other tasks are used to update slowly varying parameters of the non-linear plant model. These tasks are often data-handling intensive, e.g. using trigonometric functions or matrix inversion. Their duration can be far longer than the period assigned to the direct path, but delaying their ending instants has a weak effect on the controller performance, e.g. the control jitter or the system's stability. Thus they can be assigned a low priority so that their execution is preempted by every execution of the direct path calculation [19].

The whole system is usually run over a single CPU, or on a limited number of CPUs with a static partition of the tasks. *Priorities must be assigned to tasks according to their relative urgency*, so that a high criticality module can preempt the execution of a less urgent one when it becomes runnable. This also ensures that in case of overload the tasks which are serviced on time will be the most critical ones ; obviously provision must be made to avoid an operating system's crash in case of overload or repeated deadlines miss. As far as the control algorithm does not change, the relative urgency of modules does not change either so that priorities inside the control algorithm can be statically assigned. Anyway the relative urgency between tasks may change during a restructuring of the system, e.g. for admission of new control tasks ; in that case this is done at a rate imposed by the controlled system and environment's dynamics, not at the scheduler's rate like in EDF.

However, using only preemption is not enough to accurately specify the controller, in particular it cannot efficiently take into account the precedence constraints between subsets of the control algorithm.

4.2 Synchronisation and communication

A partial synchronisation of tasks allows for the specification of precedence constraints and thus improves the control performance by minimising the latency on critical paths.

Generally, the best data to be used in a *closed-loop* control algorithm is the last one produced, thus the buffers between modules may have only one slot, and the incoming data overwrites the old one. The following basic communication/synchronisation schemes are of particular interest to implement closed-loop controllers :

- Synchronising the starting of a module on the end of another module (data production) is of prime interest to specify data dependency and to ensure latency minimisation on some critical control paths [19]; such one way synchronisation with a system's clock is also a good way to provide for periodic tasks.

- Asynchronous communication must be used between modules with unrelated sampling rates. As mutual exclusion on shared data using mutex introduces side effect run-time synchronisation (priority inversion), lock-free shared data protection mechanisms, e.g. [20], must be preferred;

- Strong synchronisation (the well known rendezvous) must be sparingly used, and its appropriateness w.r.t. the application requirements must be carefully considered as it is a very efficient dead-lock generator.

Some comprehensible rules, based on control algorithm analysis, layers of static priority and careful synchronisation, allows for the design of rather complex controllers taking into account the control requirements; they also lead to off-line analysis independently of the priority assignment, e.g. [18].

However, the design and schedulability analysis of systems based on static values of the scheduling parameters assumes the *a priori* knowledge of the worst case execution time of the tasks [15]. This is always difficult to measure, and can lead to a severe under-use of the computing power when the computing load has large variations, e.g. in vision-based control. Another source of timing uncertainty arises from variable communication delays when the controller is distributed over a local area network, leading to measuring jitter. Perturbations on the system's computing load also arise during restructuring due to admission or cancellation of control activities upon occurrence of events coming from a dynamic environment. Adaption against timing uncertainties could be provided by more flexible scheduling policies.

5 Further results on feed-back scheduling

It is expected that an on line adaption of the scheduling parameters of the controller may increase its overall efficiency w.r.t. timing uncertainties coming from the unknown controlled environment. Also we know from control theory that closing the loop may increase performance and robustness against disturbances when properly designed and tuned (otherwise it may lead to instability). Thus the idea of feed-back scheduling recently arose both from the control side

[6, 5] and from the real-time computing side [22, 11]. Anyway the design of efficient feed-back schedulers must start with a safe design of a real-time implementation based on control requirements as previously described (i.e. algorithm partition, precedence constraints and priority assignment).

Figure 1 gives an overview of a feed-back scheduler where an outer loop (the *scheduling controller*) has been added to the process controller to adapt in real-time the scheduling parameters ; it takes as input measures taken on both the controlled process output (e.g. the tracking error) and on the computer's activity (e.g. the computing load). Besides this controller working periodically at a rate which is itself to be determined, the system's structure may evolve along a discrete time scale upon occurrence of events, e.g. for new tasks admission or exception handling. These decisional processes are handled by another real-time task, the *scheduling manager*, which is not further detailed in this paper. Preliminary studies and experiments show that the tools needed to handle the measurements and actuation tasks, e.g. precise time-stamping of events, already exist in most off-the-shelf RTOS ; thus these housekeeping tasks can be built in a middleware layer between the kernel and the application tasks and we do not need to patch an existing real-time kernel.

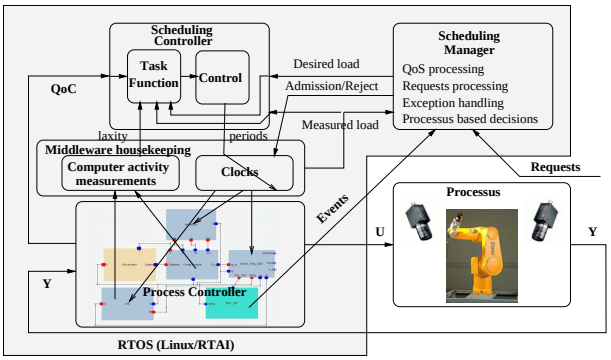

Figure 1: Feed-back scheduler structure

The problem can thus be stated as QoC (Quality of Control) optimisation under constraint of available computing resources. During experiments an estimate of the current utilisation may be computed as:

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

where C_i is the estimated execution time of the task i, and h_i the sampling period assigned to task i.

Preliminary studies [6] suggest that a direct synthesis of the scheduling regulator as an optimal control problem leads, when it is tractable, to a solution too costly to be implemented in real-time[2]. Practical solutions will be found in the available control toolbox or in enhancements and adaptation of current control theory. For instance the calculation of the new task periods can be done by the rescaling [4]:

$$h_i^{new} = h_{i_{nominal}} \frac{U}{U_{sp}}$$

where U_{sp} is the utilisation set-point.

The feedback scheduler then controls the processor utilisation by assigning task periods that optimise the overall control performance.

5.1 A new feedback scheduling architecture

In order to adjust on-line the scheduling parameters of the control tasks, a control-scheme should be established for the scheduler, as done in [5]. As any control design problems, the important issues are therefore the specification of control inputs, measurement outputs and control structure.

5.1.1 Control structure: Feedback scheduling is a dynamic approach allowing to better use the computing resources, in particular when the workload changes e.g. due to the activation of an admitted new task. We propose in figure 2 a hierarchical control structure. The feedback scheduler controls the CPU activity according to the computing resource availability (measured through some computing load metric) by adjusting the periods of the tasks used in the process controller(s).

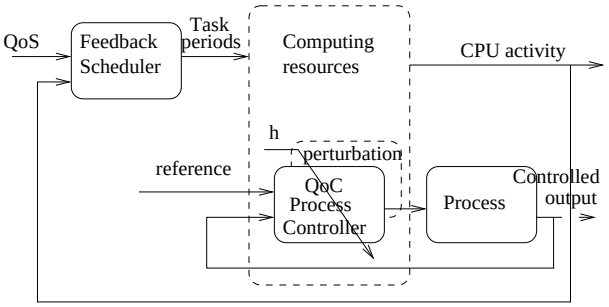

Figure 2: Hierarchical control structure

On the other hand the internal process controller is here designed to take into account timing uncertainties, e.g. due to preemptions which are unavoidable in real-time control and difficult to accurately predict in a dynamic environment. Indeed unknown input-output latencies can deteriorate the process performances and stability; thus the considered Quality of Control measure is composed of the usual tracking error and the robustness w.r.t to control delays.

Note that in this preliminary study this structure is simplified compared with the one of figure 1 : ideally the QoC measured from the controlled process would be also fed back to the scheduler and be taken into account in the QoS computation.

[2] recall that the feed-back scheduler will be itself a real-time task loading the shared computing resource...

5.1.2 Sensors and actuators: As stated in section 4, priorities must be assigned to control tasks according to their relative urgency ; this ordering remains the same in the case of a dynamic scheduler. Dynamic priorities, e.g. as used in EDF, only alter the interleaving of running tasks and will fail in adjusting the computing load w.r.t. the control requirements measured by the QoC. In consequence we have elected the tasks periods to be the main actuators of the system running on top of a fixed priority scheduler[3]. Anyway, to be compliant with control requirements the control algorithms must be first adequately sliced into sets of ordered synchronised and communicating real-time tasks as in the static case.

Our aim is to adjust on-line the sampling periods of the controllers in order to meet the computing resource requirements. The control inputs are then the periods of the control tasks. The measured output is the CPU utilisation, estimated through the execution time, in a similar way as in [5]. Thus, for each period of the scheduler h_S, the CPU utilisation is estimated from job execution-time measurements of the control tasks, as:

$$\begin{aligned} \overline{U}(kh_S) &= \sum_{i=1}^{n} \frac{\overline{c}_i(kh_S)}{h_i((k-1)h_S)} \\ \hat{U}(kh_S) &= \lambda \hat{U}((k-1)h_S) + (1-\lambda)\overline{U}(kh_S) \quad (1) \end{aligned}$$

where, for each period of the scheduler, $h_i(kh_S)$ is the sampling period currently assigned to the control task i, and $\overline{c}_i(kh_S)$ is here the mean of the measured execution-times of the control task i during each period of the scheduler. Samples for the measured output (i.e. CPU utilisation) are taken here at the period of the scheduler to be controlled, which is usual for identification purpose. In (1)λ is a forgetting factor, chosen as $\lambda = 0.3$, which ensures a fast estimation.

5.1.3 Control design and implementation: Our aim is here to provide a new control scheme for the feedback scheduler. First one should note that, if the execution times are constant, then the relation, $U = \sum_{i=1}^{n} C_i f_i$ (where $f_i = 1/h_i$ is the frequency of the task) is a linear function (while it would not be as a function of the task periods). For the considered application with two control tasks, illustrated in section 6, we obtain the following static map. This linear characteristic of the static behaviour of the scheduler implies that we can consider a linear model for the scheduling controller design. Now, using (1), the estimated requested CPU load is:

$$\hat{U}(kh_S) = \frac{(1-\lambda)q^{-1}}{1-\lambda q^{-1}} \sum_{i=1}^{n} \overline{c}_i(kh_S) f_i(kh_S) \quad (2)$$

where q stands for the shift operator. For the control design, we have chosen to consider a "normalised"

[3]Possible secondary actuators are variants of the control algorithms, with different QoS contributions to the whole system. Such variants should be handled by the scheduling manager working on a discrete events time scale

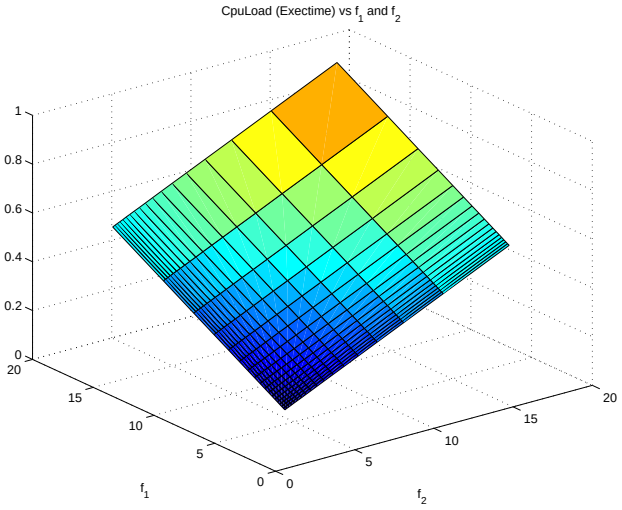

Figure 3: CPU utilisation vs control tasks frequencies

control model (i.e independent on the execution time) as $H(z^{-1}) = \frac{(1-\lambda)z^{-1}}{1-\lambda z^{-1}}$. As illustration, in the two control tasks system presented in the following section, the control scheme is therefore as in figure 4 where the estimated execution-times are used on-line to adapt the gain of the controller for the original CPU system (2).

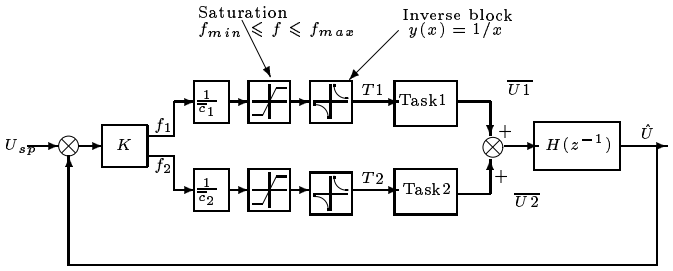

Figure 4: Control scheme for CPU resources

According to this control scheme, the design of the controller K can be made using any advanced control methodology. For the considered application (see section 6), we have chosen the H_∞ control theory which can lead to a robust controller w.r.t modelling errors (see [21] for details on H_∞ control). Moreover it provides good properties in the presence of external disturbance, as it is emphasised in the illustrative example.

5.2 A QoC criterion: robustness w.r.t. delays

In most of computer-controlled systems, the computer must do in each period: sampling of the process output, executing of the control algorithm, sending the new control signal to the process. This implies that the control task is supposed to have a fixed period and that the input-output latency (i.e the control delay) is small and without jitter. If not, this should be considered in the control design. In fact such delays may be of two-fold: first the communication be-

tween the sensor and the controller, and between the controller and the actuator, as well as the computational delay corresponding to the control computation cost. The latter is generally less than a sampling period. However when the control task is preempted by higher priority tasks, this may lead to delays larger than a sampling period. In this framework, these delays may be lumped in a single input delay. Hence, we will consider in the sequel discrete time-delay systems as:

$$\Sigma_d : \quad x(k+1) \quad = \quad Ax(k) + Bu(k-d) \quad (3)$$

where $x(k) \in \mathbb{R}^n$ is the state vector assumed to be measured, $u(k) \in \mathbb{R}^r$ is the control input vector, d is the positive *unknown* delay value, A and B are real matrices of appropriate dimensions. Let us recall that in this formulation, d is the input unknown delay.

Remark 1 *Control theory for linear systems sampled at fixed rates, including fixed and known delays, has been much studied for ten years. Let us cite [2] where an augmentation approach is used for sampling a system with time-delay and obtain an equivalent free delay representation. However if the delay vary the dimension of the non-delayed system will vary accordingly, which is not acceptable. Since the delay is here assumed to be unknown, such an approach cannot be applied. This motivates the need in real-time control, to consider discrete-time representations with time-delay. On the other hand specific methods to time-delay systems could be used to derive a discrete-time representation with delays [7] from a continuous-time one.* □

For systems (3), the following family of state feedback control laws is considered:

$$u(k) = Kx(k) \quad (4)$$

Using the control law (4), the closed-loop system is then:

$$x(k+1) = Ax(k) + BKx(k-d)$$

Since the closed-loop system is a state-delayed system, specific method to study the stability of such systems must be used in order to design the feedback gain K. Two kinds of stability results can be obtained: either delay-independent, or delay-dependent ones. For discrete-time systems let us cite the papers of [9, 12] as they consider systems with unknown delay, with uncertainties, and eventually a disturbance. In the considered framework of real-time control with delays, experiments may allow delay measurements (see for instance [14]). Even if the exact delay values are unknown, it can be possible to estimate a bound on the control delay, i.e. a maximal delay. We then focus here on delay-dependent methods for discrete time-delay systems, that ensure stability for a maximum allowable delay. In [17] an Linear Matrix Inequality approach has been used to design a memoryless delay-dependent state feedback control law of the form (4) that ensures asymptotically stability for the closed-loop system (3-4) for any time-delay d satisfying $0 \leqslant d \leqslant \bar{d}$.

6 Illustrative example

In this part, we describe our methodology for the feedback-scheduling of control tasks in the case of two linearised pendulum systems presented in [6], i.e.

$$\left\{ \begin{array}{lcl} \dot{x}(t) & = & Ax(t) + Bu(t) \\ y(t) & = & Cx(t) \end{array} \right. \quad (5)$$

where $A = \left[\begin{array}{cc} 0 & 1 \\ \omega_0^2 & -2\xi\omega_0 \end{array} \right]$, $B = \left[\begin{array}{c} 0 \\ \omega_0/g \end{array} \right]$ and $C = \left[\begin{array}{cc} 1 & 0 \end{array} \right]$, with $\xi = 0.2$, $g = 9.81$, $\omega_0 = 3.77$ for the first pendulum, $\omega_0 = 4.08$ for the second one. y is the controlled output and x is here assumed to be measured. The corresponding system is the pendulum in the upright position (i.e an unstable open-loop system).

According to the choice of a sampling period h, and including a control delay d, the following discrete model is obtained:

$$x(k+1) \quad = \quad A_h x(k) + B_h u(k-d) \quad (6)$$

where A_h and B_h are the corresponding system matrices.

In this case, the nominal sampling period is $0.1sec$, and is assumed to be changed between $0.02s$ and $0.5s$ by steps of $0.02s$. Note that this is larger than required by the usual criterion for the choice of sampling period [2]: $0.1 \leqslant \omega h \leqslant 0.6$. The corresponding saturation block in figure 4 is then such that $f_{min} = 0.02$ and $f_{max} = 0.5$. To this model, the control delay must be added to lead to the considered system representations (3).

6.1 Process control design

The methodology described in section 5.2 has been used to design delay dependent state feedback control laws for system (6). This methodology is compared to a classical pole placement control law, designed assuming there is no control delay.

As precised in [5], on-line recalculations of the control law are often too costly, particularly here where convex optimisation (LMI) is used. The considered solution is then to calculate off-line the parameters of the controller for a range of sampling periods, and store them in a table, as we have done for the considered range of sampling period, i.e. $[0.02 - 0.5]$ with a step of $0.02s$.

6.2 Feedback scheduling design

The feedback scheduler is here implemented as an application task that runs in parallel with the control

task, with a higher priority. It executes as a periodic task, with a period h_S, larger than the sampling period of the control task, in order not to change two often the sampling period of the control tasks, i.e. the control parameters. We have chosen here $h_S = 2s$.

As precised before in section 5.1, The H_∞ control theory is here applied . Such a control method uses some weighting functions that have to be chosen to satisfy the performance specifications, i.e.here mainly a closed-loop system with a rise time of $4s$ and a module margin higher than 0.5 for robustness. Using the classical methods available in control design softwares, the solution of the H_∞ control problem gives the controller $K = [K_1 \quad K_2]$ with $K_1(z) = K_2(z)$ and:

$$K_1(z) = \frac{-0.1938622 + 0.4519570z + 0.6475011z^2}{0.8412522 - 0.1573669z + z^2}$$

6.3 Simulations

Simulations have been performed using Truetime, a Matlab/Simulink toolbox for real-time control [8]. Here the first pendulum has an higher priority than the second one. On figure 6 the presented results are then only given for this lower priority pendulum. Moreover we assume here that the control delay is half a nominal sampling period, i.e. $0.05s$.[2mm] The scenario used here is the following. At $t = 0$ the control tasks begin. At $t = 2$, the scheduling controller is switched on, and the feedback control of the computing resources is realised around 60% of utilisation (which is the nominal CPU load). At $t = 25s$, a reference step input is sent, representing an decrease of 30% of ressource availability, leading to a increase of the control sampling periods. In both cases (classical and delay dependent control laws) the performances are few affected by this change, due to the adaptation of the control parameters (stored in tables). At $t = 50$, the set-point of resources availability is set back to 60%. At $t = 70s$, a disturbance task, with an higher priority than the control tasks, but for which we cannot measure its execution time, appears. As seen on figure 5, this task implies important preemptions in the lower pendulum control task, leading to an increase of the control delay. As shown in figure 6 the classical pole placement control law, design without account for delay, becomes unstable. On the contrary, the memoryless delay-dependent control law ensures robust stability of the system, despite the presence of unknown and varying control delay. This generates some temporal uncertainties in the execution of the control task These results point out the interest of feedback scheduling, allowing an adaptation of the control law parameters under resource availability, as well as the importance of taking into account the control delay in the design of the process control law, as done in part 5 in the process modelling and control steps. If not the process can become unstable due to such temporal uncertainties.

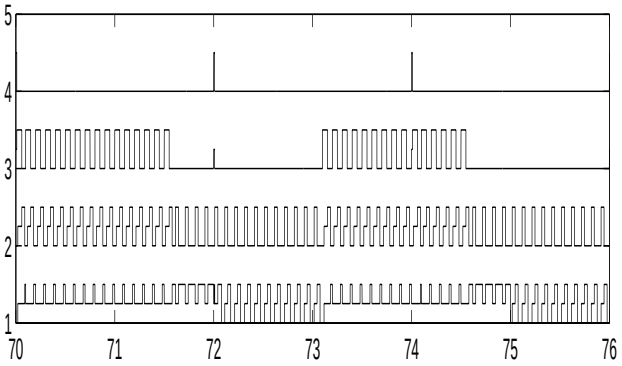

Figure 5: Zoom on process control behaviours

7 Conclusion

In this paper, a new methodology for scheduling-control co-design is proposed. We have provided a new architecture and design method for feedback scheduling. Indeed the control synthesis of the feedback scheduler has been provided using the H_∞ control theory, and the gain of the controller is adapted online using the measured execution times of the control tasks. An integrated control-scheduling approach is proposed, where the state feedback control law has been designed according to a range of sampling periods (for adaptation under computer resources requirements), and taking into account some control delay (mainly due here to the preemption in the controllers). Some simulation results have been given, which emphasises the interest of this approach.

Obviously simulations are not accurate enough to model a real system, and ongoing experiments running on top of an actual RTOS must be further developed to better assess this new approach; in particular the computing overload due to such method must be carefully evaluated and must be integrated in the overall QoS computation. However even simulations show that setting the scheduling controller period, estimating the CPU load and choosing the QoC criterion are not trivial tasks. Finally, besides control related aspects, the role and structure of the scheduling manager must be detailed to efficiently integrate exception handling and control modes in a safe and flexible control system.

References

[1] K.-E. Arzén, A. Cervin, J. Eker, and L. Sha. An introduction to control and scheduling co-design. In *39th IEEE Conference on Decision and Control*, Sydney, Australia, december 2000.

[2] K.J. Astrom and B. Wittenmark. *Computer-Controlled Systems.* Information and systems sciences series. Prentice Hall, New Jersey, 3rd edition, 1997.

[3] A. Cervin. Towards the integration of control and real-time scheduling design. Technical Report Li-

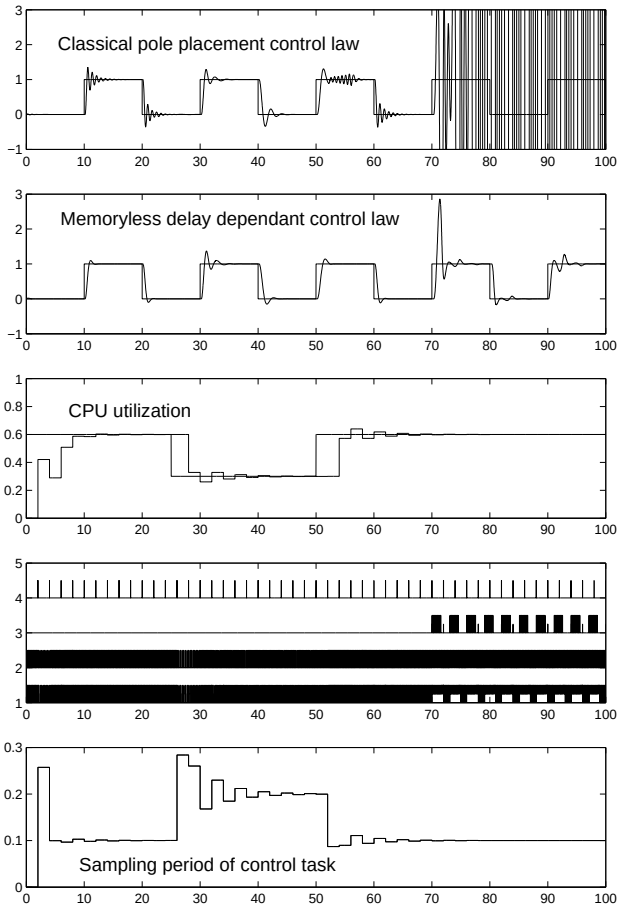

Figure 6: Feedback scheduling and process control behaviours

centiate thesis ISRN LUTFD2/TFRT–3226–SE, Department of Automatic Control, Lund Institute of Technology, Sweden, May 2000.

[4] A. Cervin and J. Eker. Feedback scheduling of contol tasks. In *Proceedings of the 39th IEEE Conference on Decision and Control*, Sydney, Australia, December 2000.

[5] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén. Feedback-feedforward scheduling of control tasks. *Real Time Systems*, 23(1):25–54, 2002.

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

[7] A. Fattouh, O. Sename, and J.-M. Dion. Pulse controller design for linear time-delay systems. In *IFAC Symposium on System Structure and Control*, Prague, 2001.

[8] D. Henriksson, A. Cervin, and K.-E. Årzén. Truetime: Simulation of control loops under shared computer resources. In *Proceedings of the 15th IFAC World Congress on Automatic Control*, Barcelona, Spain, July 2002.

[9] Y. S. Lee and W. H. Kwon. Delay-dependent robust stabilization of uncertain discrete-time state-delayed systems. In *IFAC 15th World Congress*, Barcelone, Spain, 2002.

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

[11] C. Lu, J.-A. Stankovic, G. Tao, and S.-H. Son. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real Time Systems*, 23(1):85–126, 2002.

[12] M. S. Mahmoud. Robust h_∞ control of discrete systems with uncertain parameters and unknown delays. *Automatica*, 36:627–635, 2000.

[13] P. Marti, J. Fuertes, G. Fohler, and K. Ramamritham. Jitter compensation for real-time control systems. In *22nd IEEE Real-Time Systems Symposium*, London, UK, 2001.

[14] J. Nilsson. *Real-Time Control Systems with Delays*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, January 1998.

[15] P. Puschner and A. Burns. Guest editorial : A review of worst-case execution-time analysis. *Real Time Systems*, 18(2-3):pp 115–128, 2000.

[16] M. Ryu, S. Hong, and M. Saksena. Streamlining real-time controller design: from performance specifications to end-to-end timing constraints. In *IEEE Real Time Systems Symposium*, 1997.

[17] O. Sename, D. Simon, and D. Robert. Feedback scheduling for real-time control of systems with communication delays. *submitted to IEEE International Conference on Emerging Technologies and Factory Automation, Lisbon, Portugal*, september 2003.

[18] D. Simon and F. Benattar. Design of real-time periodic control systems through synchronisation and fixed priorities. Technical Report RR4677, INRIA, december 2002.

[19] D. Simon, E. Castillo, and P. Freedman. Design and analysis of synchronization for real-time closed-loop control in robotics. *IEEE Trans. on Control Systems Technology*, 6(4):445–461, july 1998.

[20] H.R. Simpson. Multireader and multiwriter asynchronous communication mechanisms. *IEE Proceedings-Computer and Digital Techniques*, 144(4):241–244, 1997.

[21] S. Skogestad and I. Postlethwaite. *Multivariable Feedback Control: analysis and design.* John Wiley and Sons, 1996. http://www.chembio.ntnu.no/ skoge/.

[22] J. Stankovic, C. Lu, S. H. Son, and G. Tao. The case for feedback control real-time scheduling. In *11th Euromicro Conference on Real-Time Systems*, York, England, 1999.

[23] J.A. Stankovic, M.Spuri, M. Di Natale, and G. Buttazzo. Implications of classical scheduling results for real-time systems. *IEEE Computer*, 28(6):16–25, 1995.

IMPACT OF SCHEDULING POLICIES ON CONTROL SYSTEM PERFORMANCE

Henrik Schiøler, Anders P. Ravn** and Jens Dalsgaard Nielsen**

Aalborg University,
*Department of Control Engineering
**Department of Computer Science

ABSTRACT

It is well known that jitter has an impact on control system performance, and this is often used as an argument for static scheduling policies, e.g. a time triggered architecture. However, it is only completion jitter that seriously disturbs standard linear control algorithms in a way similar to the delay inherent in a time triggered architecture. Thus we propose that standard control algorithms are scheduled dynamically, but without preemption. Analysis of this policy is contrasted with a corresponding time triggered architecture and is shown to have better impulse response performance both in the deterministic case and under white noise disturbances. The conclusion is that under very reasonable assumptions about robustness of control algorithms, they are insensitive to relase jitter, albeit strongly sensitive to completion jitter, thus priority based scheduling without *preemption is may be preferable for such systems.*

1. INTRODUCTION

Control systems are almost exclusively implemented on multitasked computers, where task scheduling is determined by a real-time scheduling policy. The overall performance of a system is therefore dependent on cooperation between the disciplines of control engineering and software engineering of hard real-time systems. Highly optimized solutions may be found by merging the disciplines [16, 17]; but the solutions are usually highly specialized, and requires a development effort by highly skilled people with a knowledge spanning both areas. In most development organizations, control and scheduling are done by different specialists, thus a smooth cooperation depends on a clear division of responsibilities - a contract. The basis of a standard contract may be phrased as follows:

> The control engineers will deliver a collection of tasks to be executed periodically. Each task is characterized by its period T and its worst case computation time C.
>
> The software engineer promises to execute each task periodically with the given period and with sufficient computation resources for the worst case computation time.

When both parties are conservative, i.e. the control engineers use very robust algorithms and the software engineers load the computers moderately, the cooperation works well, and there is no need to elaborate the contract. However, when we want efficient solutions, the fine print of the contract becomes important. What does it mean to execute a task periodically?.

The interpretation of the control engineer is that the task is released and reads its inputs from sensors at times r_k for $k = 0, 1, \ldots$, where the release times are equidistant $r_{k+1} - r_k = T$, and where the first release r_0 happens within the first period of time. Furthermore, the tasks are assumed to be completed at times $c_k = r_k + C$, when outputs to actuators has been done. In some cases, controls are developed assuming that computations take no time, i.e. $C = 0$; but this assumption is clearly the responsiblity of the control engineer and does not influence the common contract.

The software engineer, who applies a scheduling policy, has another interpretation. A periodic task shall be executed once within each of the periods $[0, T], [T, 2T], [2T, 3T], \ldots$ (Here we make the usual assumption that the deadline for a task is the same as its period.) Tasks may thus actually be released and completed at times $r_i', c_i' \in [T \cdot i, T \cdot (i+1)]$. The differences $J_i^r = r_i - r_i'$ and $J_i^c = c_i - c_i'$ are respectively the *release* and the *completion jitter*. Generally a maximum release jitter may be guaranteed, i.e. $J_i^r \leq \delta$ as well as a maximum delay D between release and completion, i.e. $D_i = c_i - r_i \leq D$.

The fine print of an extended contract will explain, how jitter is handled. For the control engineer, several compensation mechanisms are possible [11]. They are typically based on per sample modification of the applied control law, which requires knowledge of the actual jitter. The software engineer may also choose scheduling disciplines that makes jitter predictable. When we consider popular scheduling policies, we have the following characteristics:

Static scheduling: With a preplanned schedule, there is no jitter, or at most a small, amount due to asynchronous interrupts.

A special case is the time triggered architecture [19, 3], where, similar to PLC-controllers, input is read at the start of a period and output is produced at the end. Both operations are assumed to take no time, which in general is reasonable. One can say that in some sense, the release jitter is minimized at the cost of maximized delay.

Dynamic scheduling: Here tasks are assigned priorities according to some criteria, see e.g. [7, 4, 5] for deadline montonic, rate monotonic or other policies. When there are

only periodic tasks and when computation times do no vary, there is no more jitter than in a static schedule; but in most cases, where dynamic scheduling is used, because it adapts better to situations with aperiodic tasks or server tasks, jitter may become difficult to control. However, if scheduling is done without preemption, delay almost is eliminated, at the cost of introducing blocking and in turn increasing release jitter for higher priority tasks.

When a system is developed, there are thus the following options:

1. Use static scheduling or the time triggered approach and pay the price in the form of complex analysis of the concrete configurations.
2. Use dynamic scheduling and complicate real time control by jitter and or delay compensation.

The thesis of this paper is that we can avoid the complexities for standard proportional control systems by using dynamic scheduling without preemption. In this very common case, our analysis shows that control performance is almost unaffected by release jitter, and that the control performance is better than the one that can be expected from a comparable time triggered scheduling policy.

Overview

The following section surveys related work, while Section 3 analyses the effect of release jitter on simple first order proportional control systems, as well as the effect of delay, as introduced in time triggered architectures. Jitter and delay are analysed from deterministic and stochastic viewpoints. Section 4 provides a generalization to higher order systems of the approach presented in the previous section. Section 5 concludes and indicates directions for future research.

2. RELATED WORK

The study of sampled control systems is by no means new. However the special case for irregular sampling due to real time scheduling seem to gain much interest recently. The paper [8] and the report [9] provide overviews of the problem and surveys of related work within real time systems and control engineering. The article [10] considers sampling delay and jitter and a rational model for varying delay is presented along with an accompagning robust controller design based on μ-synthesis [15]. The effect of limited sampling jitter is illustrated by an example of a double integrator. In [11] an online jitter compensation is introduced; it uses per sample recalculation of control law parameters based on timestamps. Optimal ressource distribution to control tasks is investigated in [12] and [1]. In [1] the time triggered approach is adopted as a basis for an optimization scheme yielding optimal sampling periods under various preemptive scheduling disciplines. Optimality is defined on the basis of an appropriate objective function reflecting system robustness w.r.t. stability margins. The paper [18] presents a technique to bound release jitter, based on the modification of task temporal parameters in DM and EDF scheduling. In [13] we find stability analysis for sampled systems with non-ideal sampling, where both jitter and delay is considered. The sampling proces is considered for 3 cases: constant, periodic and general. In the general case a simple common one sample criterion is assumed for the transition matrix.

In comparison, our work provides analytical results for the impact of jitter on linear control system comprising first and higher order continuous time systems equipped with state proportional discrete time controllers. We advocate a simplistic approach allowing release jitter whereas delay is considered tightly bounded since non preemptive scheduling is assumed. As in [1] we consider neither redesign nor on line adaptation to accomodate for sampling irregularities. We choose the time triggered approach from [1] as a paradigm for comparison and provide robustness analysis for jittered sampling similar to [13], however, our analysis of the sample process is more general than presented in [13].

3. FIRST ORDER CONTROL SYSTEMS.

Consider a first order continous time dynamic system evolving in time according to differential equation (1)

$$\dot{x} = A \cdot x + B \cdot u \tag{1}$$

A ZOH-equivalent continuous to discrete transformation [6] produces the following discrete time system

$$x_{k+1} = F_T \cdot x_k + G_T \cdot u_k \tag{2}$$

wher F_T and G_T are given by

$$F_T = exp(AT) \tag{3}$$

$$G_T = \frac{B}{A} \cdot (exp(AT) - 1) \tag{4}$$

and T is the nominal time between samplings k and $k+1$. Assume that a discrete time state space control law K is derived yielding a closed loop system

$$x_{k+1} = F_T \cdot x_k - G_T \cdot u_k = (F_T - G_T \cdot K) \cdot x_k = Q_T \cdot x_k \tag{5}$$

where Q_T is given by

$$Q_T = exp(AT)(1 - \frac{KB}{A}) + \frac{KB}{A} \tag{6}$$

As an example let $A = -0.02, B = 1, T = 1$ and the nominal pole placement be $Q_T = 0.8$ then $K = 0.18$. Thus by proportional feedback the bandwidth of a stable first order system is increased approximately be a factor 10 which seems reasonable.

3.1. Deterministic analysis of jitter

Suppose sampling times vary over time, so that the time between samplings k and $k+1$ is now T_k, then the following solution to (5) is found

$$x_k = \Pi_{j=0}^{k-1} Q_{T_j} \cdot x_0 \tag{7}$$

where $T_j = r'_{j+1} - r'_j$. Assuming $J_i^r \leq \delta$ for some positive real number δ, we obtain

$$m\,T - \delta \leq \sum_{i=k}^{k+m-1} T_i \leq m\,T + \delta \quad \text{for all } k, m \geq 0 \tag{8}$$

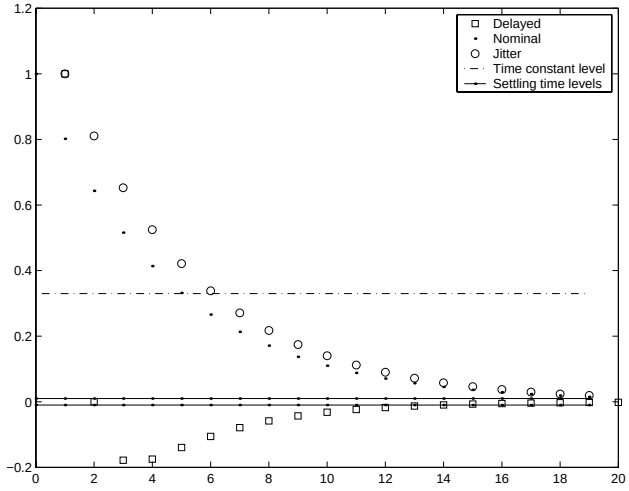

Figure 1: Nominal, delayed and jittered step error responses.

We consider error convergence in response to a reference value change at $t_0 > 0$. The first sampling instant following the change may ultimately be delayed δ. After the first sampling, convergence may be bounded as in inequality (9). Assuming $Q_{T_j} > 0$, Q_{T_j} is decreasing in T_j, so for the constraints (8) there is a symmetric interior maximum $T_j = T - \frac{\delta}{k}$ for x_k on the hyperplane $\sum_{j=0}^{k-1} T_j = kT - \delta$, i.e.

$$\begin{aligned} x_k &= x_0 \ \Pi_{j=0}^{k-1} Q_{T_j} \\ &= x_0 \ \Pi_{j=0}^{k-1} (exp(AT_j)(1 - \frac{KB}{A}) + \frac{KB}{A}) \\ &\leq x_0 \ \bar{x}_k \\ &= x_0 \ (exp(A(T - \frac{\delta}{k}))(1 - \frac{KB}{A}) + \frac{KB}{A})^k \end{aligned} \quad (9)$$

The above reasoning is based on the assumption that $Q_{T_j} > 0$ which however, is considered to be a rather mild restriction. For the simple example above, the bound for T_j ensuring $Q_{T_j} > 0$ would be $T_j \leq 5.26 \ T$, which is equivalent to occasional vacations of the sampling process lasting more than 5 times the nominal sampling period. The latter is clearly quite significant.
An overall upper bound for error convergence is found by the pairs $(t_0 + \delta + kT, x_0 \bar{x}_k)$ as shown in Figure 1, where $\delta = T$ is assumed for the above example to make it comparable with the delayed situation.

3.2. Deterministic analysis of time triggered control

Fixed priority scheduling along with the celebrated RMA priority assignment [2] as well as EDF schdeduling [2] provide conditions for completion jitter below task periods, i.e. T. When control dynamics are slow compared to the nominal sampling rate such an approach may work satisfactory, but it may also deteriorate even reasonably robust nominal designs significantly.
Introducing a one period in loop delay to the example from the previous section results in the second order system of equation (10)

$$\begin{aligned} x_{n+1}^1 &= x_n^2 \\ x_{n+1}^2 &= -K \ G_T \ x_n^1 + F_T \ x_n^2 \end{aligned} \quad (10)$$

with a resulting pole pair $(0.23, 0.73)$. Impulse responses for the nominal system and its delayed counterpart in Figure 1 show a significant overshoot of about 20 % introduced by delay.
Delay compensation can for fixed delay of one single or an integer number of sample periods be performed through the well known Smith predictor governed by the dynamics in equation [14].

$$\begin{aligned} \hat{x}_{k+1} &= F_T \cdot \hat{x}_k + G_T \cdot u_k \\ u_k &= -K(x_{k-1} - \hat{x}_{k-1} + \hat{x}_k) \end{aligned} \quad (11)$$

The result of compensation is also shown in Figure 1 revealing the expected closed loop response with an additional delay of 1 sample period. Ideally more advanced control designs should accompany application of the Smith predictor as pointed out in [14]. However we believe the rather simplistic approach above is justified for comparative reasons since we compare approaches of similar complexities.

3.3. Comparison for deterministic analysis

When $Q_{Tp} > 0$, no overshoot is introduced by release jitter. Thus convergence speed is simply stated in time constant or settling time terms, i.e. $t^* = \min\{\delta + kT \mid \bar{x}_k \leq \alpha\}$ where α assumes values 0.33 or 0.01 for time constant or settling time respectively. As seen in Figure 1 jitter increases the time constant from 5 to 6 sampling periods and settling time is approximately unchanged from the nominal case. It must be noted that the delayed impulse response is superior to both the nominal and the jittered ones when observing time constant and settling time alone. However in many cases overshoot is undesirable or even hazardous, as in positional controllers for mechanical systems, and in general overshoot may indicate robustness problems, i.e. the ability to maintain stability under system uncertainties. All together the jittered response is by most standards far closer to the nominal design than the delayed one. Application of the Smith predictor produces a result almost identical to the worst case jittered response.

3.4. Stochastic analysis.

Disturbances are incorporated into the state space model by restating equation (1) as a stochastic differential equation, i.e.

$$dx = A \cdot x + B \cdot u + dw \quad (12)$$

where w is a standard brownian motion. Let t_k and t_{k+1} be separated by T_k in time and define x_k by

$$x_k = x(t_k) \quad (13)$$

then the following stochastic discrete time model is obtained from a ZOH transformation

$$x_{k+1} = F_T \cdot x_k + G_T \cdot u_k + \int_0^{T_k} e^{A \cdot (T_k - \tau)} dw \quad (14)$$

The last term is readily shown to be an independent Gaussian random variable w_k with a variance $C_w(T_k)$ given by

$$C_w(T_k) = \int_0^{T_k} e^{2A\cdot(T_k-\tau)} d\tau \tag{15}$$

Introducing the discrete time state space control law K the following approximative closed loop model is obtained

$$x_{k+1} = Q_{T_k} x_k + \sqrt{C_w(T_k)} \cdot w_k \tag{16}$$

where w_k is a standard Gaussian variable. Computing variances yields

$$\sigma^2_{k+1} = Q^2_{T_k} \sigma^2_k + C_w(T_k) \tag{17}$$

In general, it is difficult to see which pattern of $\{T_j\}$ maximizes (17). We shall proceed less general. We assume $\delta = T$ and deduce results valid for the above example. In that case all sequences $\{T_1, .., T_n\}$ reside within the sets $D_n \subset R^n$ defined by

$$\begin{aligned} D_n &= \{(T_1, .., T_n) \mid (k-1)T \\ &\leq \sum_{j=i}^{i+k-1} T_j \leq (k+1)T \ \forall (i,k) \mid i,k \geq 1, i+k-1 \leq n\} \end{aligned}$$

The variance $\sigma^2_n = \phi(T_1, .., T_n) + \Pi_{j=0}^{n-1} Q^2_{T_j} \cdot \sigma^2_0$, where ϕ is independent of σ^2_0. Thus under stability assumptions for sequences $\{T_j\}$ where $(T_1, .., T_k) \in D_k$, $\lim_{n\to\infty} \sigma^2_n = \phi(T_1, .., T_n)$, i.e. the effect of initial conditions disappear. Next we define $\sigma^2_n(T_1, .., T_n)$ by

$$\sigma^2_n(T_1, .., T_n) = \phi(T_1, .., T_n) + \Pi_{j=0}^{n-1} Q^2_{T_j} \cdot M \tag{18}$$

where

$$M = \frac{C_w(2T)}{1 - Q^2_{2T}} \tag{19}$$

i.e. $\sigma^2_n = \sigma^2_n(T_1, .., T_n)$ for the case $\sigma^2_0 = M$. We define M_n by

$$M_n = \max_{(T_1,..,T_{2n}) \in D_{2n}} \sigma^2_{2n}(T_1, .., T_{2n}) \tag{20}$$

and likewise

$$g(x, y) = C_w(x) + Q^2_x \ C_w(y) + Q^2_x \ Q^2_y \ M \tag{21}$$

Assume $M_{n-1} = M$ and let the real sequences S_k be defined by $S_k \in R^{2k}_+$, $S_k(i) = 0$ for $i = 0, 2, .., 2(k-1)$ and $S_k(i) = 2T$ for $i = 1, 3, .., 2k-1$

Since $S_n \in D_{2n}$

$$M_n \geq \sigma^2_{2n}(S_n) = g(2T, 0) \tag{22}$$

By definition of g in (21)

$$M_n \leq \max_{(x,y)\in D_2} g(x, y) \tag{23}$$

It can be verified for the above example that

$$\max_{(x,y)\in D_2} g(x, y) = g(2T, 0) \tag{24}$$

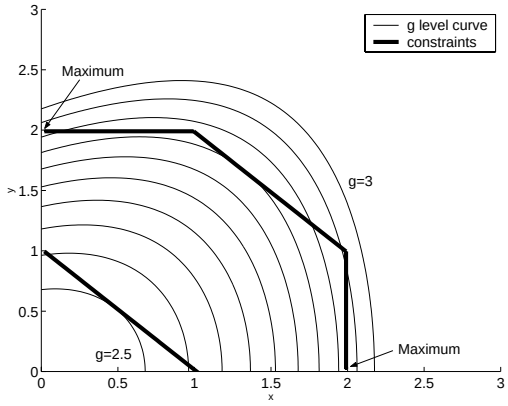

Figure 2: Maximizing the function g within D_2

as illustrated in figure (2). So by inequalities (24) and (22) $M_n = g(2T, 0) = \sigma^2_{2n}(S_n) = M$.

Since

$$\lim_{n\to\infty} \sigma^2_n = \lim_{n\to\infty} \sigma^2_n(T_1, .., T_n) = \phi(T_1, .., T_n) \tag{25}$$

M asymptotically bounds σ^2_n independent of the initial variance.

For the example above, the variance in the nominal case would amount to $\frac{C_w(T)}{1-Q^2_T} = 2.74$. The above analysis gives an assymptotic variance bound of $M = 3.05$ assumed for sampling period sequences $\{0, 2T, .., 0, 2T\}$ equivalent to the case where each second sample is released exactly one sample period to late or to a periodic sampling with a double period.

With time triggered feed back, analysis is carried out through standard matrix analysis of the second order system (26)

$$\begin{aligned} x^1_{n+1} &= x^2_n \\ x^2_{n+1} &= -K \ G_T \ x^1_n + F_T \ x^2_n + w_n \end{aligned} \tag{26}$$

where w_n is a normally distributed random variable with zero mean and variance $C_w(T)$ it can be found that the error variance amounts to 3.28 for the above example, which is a significant increase from the nominal case. In this example, jitter of one sample period increases error variance only half as much as a time triggered delay of one sample period. Performing similar computations for an equivalent system with Smith predictor, as described in equation (11), yields 3.7 for the stationary output variance. A simplistic use of the Smith predictor in conjuction with a time triggered sampling approach should therefore be strongly discouraged, whereas more advanced control designs may yield highly improved performance as pointed out in [14].

4. GENERALIZATION TO HIGHER ORDER SYSTEMS.

Analysing simple systems may yield insight and give guidelines for a more general approach. In this section we shall provide a generalization to higher order systems as well as a general approach to stochastic analysis with arbitrary jitter bound δ.

Consider a continous time dynamic system evolving in time according to differential equation (27)

$$\dot{x} = A \cdot x + B \cdot u \tag{27}$$

A ZOH-equivalent continuous to discrete transformation produces the following discrete time system

$$x_{k+1} = F_T \cdot x_k + G_T \cdot u_k \tag{28}$$

wher F_T and G_T are given by

$$F_T = exp(AT) \tag{29}$$

$$G_T = \int_0^T exp(A(T-t))dtB \tag{30}$$

T is the nominal time between samplings k and $k+1$ and $exp(\cdot)$ is the matrix exponential. Assume that a discrete time state space control law K is derived giving a nominal closed loop system

$$x_{k+1} = F_T \cdot x_k + G_T \cdot u_k = (F_T - G_T K) \cdot x_k = Q_T \cdot x_k \tag{31}$$

for $Q_T = F_T - G_T K$

4.1. Deterministic Analysis.

We shall proceed with the aid of the following first order approximation for Q_T

$$Q_T = I + A \cdot T - B \cdot K \cdot T = I + T \cdot (A - BK) \tag{32}$$

which is valid whenever $|AT| << 1$. Then the approximate eigenvalues of Q_T are found to be

$$\lambda_T = (1 + T \cdot \lambda) \tag{33}$$

where λ is a corresponding eigenvalue of $A - BK$. Up to a first order approximation eigenvalues of Q_T match those of $A - BK$, i.e. they are constant. Assuming K to be chosen so that $A - BK$ has distinct eigenvalues, a basis of eigenvectors $V = [v_1, v_2, .., v_n]$ exists. Defining principal outputs z by $z_k = V^{-1} x_k$ we obtain in correspondence to equation (7)

$$z_k^i = \Pi_{j=0}^{k-1} \lambda_{T_j}^i \cdot z_0^i \tag{34}$$

where $\lambda_{T_j}^i = (1 + T \cdot \lambda^i)$, i.e. a 1. st order approximation of the i th. eigenvector of Q_{T_j}. Thus a modulus bound on z_k is

$$|z_k^i| = \Pi_{j=0}^{k-1} |\lambda_{T_j}^i| \cdot |z_0^i| \tag{35}$$

For a well damped nominal design and low values of $T \cdot |\lambda^i|$, $|\lambda_T^i|$ is decreasing approximately affinely with T, so as for the one dimensional case, there is a symmetric interior

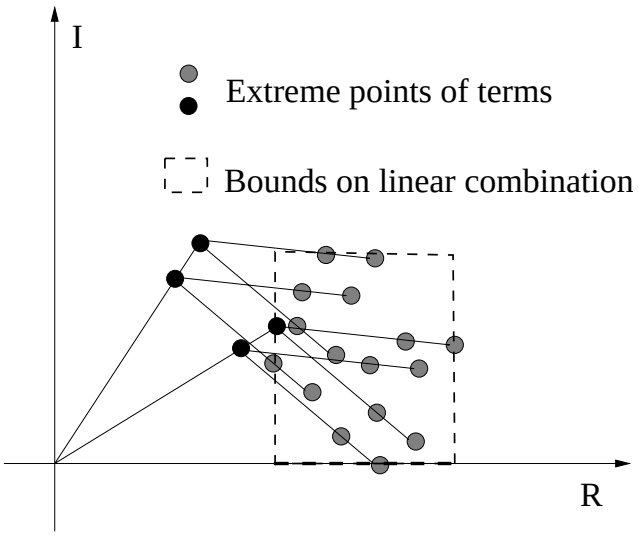

Figure 3: Computing bounds for complex linear combination.

maximum $T_j = T - \frac{\delta}{k}$ for $|z_k^i|$ on the hyperplane $\sum_{j=0}^{k-1} T_j = kT - \delta$ for the constraints (8). Thus a closed form modulus upper bound is given by

$$|z_k^i| \leq |1 + (T - \frac{\delta}{k}) \cdot \lambda^i| \cdot |z_0^i| = \bar{z}_k^i \tag{36}$$

Lower modulus bound are found by realizing that products attain minimum values in extreme points of the constraint set. So generally we have

$$|z_k^i| \geq |z^i(m\ T)|^{\lceil \frac{k}{m} \rceil} \tag{37}$$

for $(m-1)\ T \leq \delta \leq m\ T$. Inequality (37) expresses modulus bounds $m\ T, 0, .., 0, m\ T, 0, .., 0, m\ T, 0, ..$
For the phase ϕ_k^i of z_i^k we have

$$\phi_k^i = \sum_{j=0}^{k-1} \angle \lambda_{T_j}^i + \angle z_0^i \tag{38}$$

and a corresponding first order approximative bound

$$k\angle\lambda_T^i + \Lambda_T^i \delta + \angle z_0^i \leq \phi_k^i \leq k\angle\lambda_T^i - \Lambda_T^i \delta + \angle z_0^i \tag{39}$$

where Λ_T^i denotes the first order derivative of λ_T^i w.r.t. T, computed at the nominal design values. Upper and lower bounds for system states x_k are found by inspecting $x_k = V z_k$ for maximal and minimal values, as illustrated in figure (3). The required linear combination of complex numbers is performed every combination of extreme values of modulus and phase.

4.2. Example: Mechanical system.

Working cycles of mechanical automata are frequently defined by stepwise positional changes. Transitions should most often comply to specifications on rise time, settling

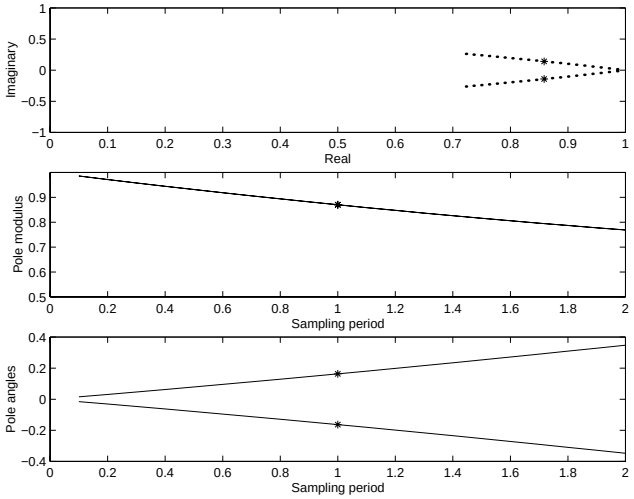

Figure 4: Root locus and corresponding moduli and angles for varying sample period in mechanical system.

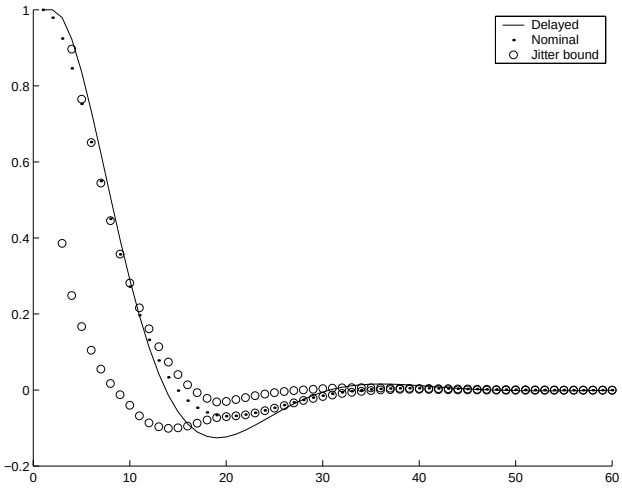

Figure 5: Nominal, delayed and jittered step responses of mechanical system.

time and overshoot. Between transitions reference positions should be maintained in the presence of mechanical disturbances in the shape of random forces acting on the system.

Our example system is defined in state space form by

$$\dot{x} = A \cdot x + B \cdot u + C \cdot w \tag{40}$$

where

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -0.2 \end{bmatrix} \tag{41}$$

and $B = C = [0\ 1]^T$. State vector components are position and velocity respectively. For nominal pole placements $(1 + 0.2 \cdot exp(-i \cdot 3/4 \cdot \pi), 1 + 0.2 \cdot exp(i \cdot 3/4 \cdot \pi))$ a feedback vector $K = [0.05\ 0.1]$ is obtained. Root (pole) locus and corresponding moduli and angles for sample periods varying over $[0, 2T]$ are shown in Figure 4, to validate the first order approximation conducted. In Figure 4 the nominal design is indicated with "*".

Step response error results for the nominal design, the time triggered approach as well as upper and lower bound for the jittered approach and $\delta = T$ are found in Figure 5.

As seen in Figure 5 jitter may increase the time constant and overshoot. Overshoot from one time delay is however even higher, though not significantly. Allthough the example is carried out for $\delta = T$ the approach presented remains generally valid within the validity domain of the first order approximations conducted.

4.3. Stochastic analysis.

We obtain the following recursion for the covariance matrices of a jittered system

$$\Sigma_{k+1} = \Sigma^{(1)}(\Sigma_k, T_k) = Q_{T_k} \Sigma_k Q_{T_k}^T + N(T_k) \tag{42}$$

where

$$N(T) = \int_0^T exp(A(T-t)) C C^T (exp(A(T-t)))^T dt \tag{43}$$

for which a close form expression is available by realizing the the eigenvectors of $exp(At)$ remain constantly identical to the eigenvectors of A and the eigenvalues of the matrix exponential are found by taking exponentials of the eigenvalues of A. We let M denote the set of, real symmetric, positive definite matrices of appropriate dimension, i.e. $\Sigma_k \in M$. Higher powers of $\Sigma^{(1)}(,)$ may be recursively defined by

$$\Sigma^{(n+1)}(\Sigma_k, T_{k+n}, .., T_k) = \Sigma^{(1)}(\Sigma^{(n)}(\Sigma_k, T_{k+n-1}, .., T_k), T_{k+n}) \tag{44}$$

A recursive expression for release jitter by the sampling intervals T_k is given by a nondeterministic automaton in the shape of a double *token bucket* filter. We define the bucket heigth $\bar{c}_k$ recursively by

$$\bar{c}_{k+1} = min\{\delta, max\{0, \bar{c}_k + T_k - T\}\} \tag{45}$$

and T_k nondeterministically by $\bar{c}_k + T_k - T \leq \delta$, then for $\bar{c}_0 \in [0, \delta]$ the automaton generates exactly all sequences $\{T_k\}$, where $\sum_{i=k}^{k+m-1} T_i \leq m\ T + \delta$. Conversely a bucket depth $\underline{c}_k$ is defined recursively by

$$\underline{c}_{k+1} = max\{-\delta, min\{0, \underline{c}_k + T_k - T\}\} \tag{46}$$

and $\underline{c}_k + T_k - T \geq -\delta$. In this case for $\underline{c}_0 \in [-\delta, 0]$ the automaton generates all sequences $\{T_k\}$, where $\sum_{i=k}^{k+m-1} T_i \geq m\ T - \delta$. Combining (45) and (46) and requiring

$$-\delta + T - \underline{c}_k \leq T_k \leq \delta + T - \bar{c}_k \tag{47}$$

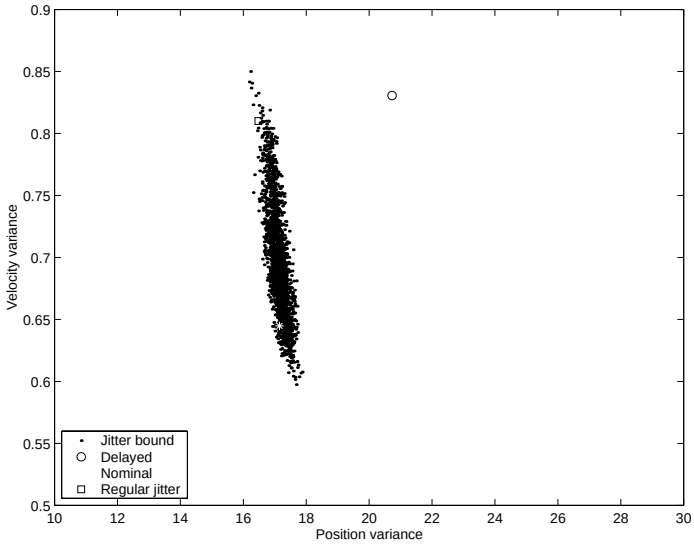

Figure 6: Stationary variances of nominal, jittered, delayed and regularly jittered systems.

our automaton generates exactly sequences where $m\,T-\delta \leq \sum_{i=k}^{k+m-1} T_i \leq m\,T+\delta$, i.e. the sampling sequences fulfilling (8). Including the covariance matrix defined in (42) into the state of our automaton, we have a state vector $[\underline{c}_k, \bar{c}_k, \Sigma_k]$ of a nondeterministic automaton defined by (42),(45),(46) and (47).

We define the state $S_0 = [0, \delta, \Sigma]$ where Σ uniquely solves $\Sigma = \Sigma^{(1)}(\Sigma, T)$. When for some $K > 0$ $\Sigma_K(\cdot, T_{k+K-1}, .., T_k)$ is a contraction for all sequences fulfilling (8), then the reachable set from S_0 is bounded. For stable nominal designs it is readily verified that S_0 is globally assymptotically reachable.

Now $\Sigma^{(1)}(\cdot, T)$ is continous for all $T \in [0, \delta]$. Consider some state S reachable from S_0 and an ϵ-neighbourhood of S. Then there exists a μ-neighbourhood B_μ of S_0, so from all states in B_μ some state within B_ϵ is reachable. Alltogether define I as the set of all states assymptotically reachable from S_0, then all states in I are mutually assymptotically reachable. Thus I is the closure of a unique smallest invariant set of the defined non deterministic automata.

Obtaining feasible hard bound estimates of I consitutes a challenge left for further research. However random explorations where T_k is drawn uniformly within the limits of (47) yields a finite dimensional irreducible, ergodic Markov chain Γ with a stationary distribution concentrated on I. Additonally for every interior point $r \in I$, there is a neighbourhood with positive measure. Thus empirical distributions of Γ may constitute feasible I estimates. Results of such an approach for the mechanical system defined above for $\delta = T$ are shown in Figure 6

Along with results from nominal, jittered and delayed systems, results from a regular jitter pattern is also shown in Figure 6. Jitter and delaying produces approximately the same increase in velocity variance, whereas delaying yields a significantly higher positional variance.

5. DISCUSSION AND FUTURE WORK

In the above we considered different approaches to the implementation of discrete time controllers on multitasking platforms. We argue that the time triggered approach and pre emptive scheduling may introduce undesirable performance drawbacks such as decreased robustness w.r.t. stability, overshoot and additional control error variance. An alternative approach based on dynamic *non preemptive* scheduling and thus allowing significant release jitter but bounding delay in the feedback loop is proposed. A typical proportional control system is analyzed under both schemes, and results strongly support the initial thesis that the non preemptive priority based scheduling performs better than a time triggered approach for standard control algorithms. The nonpreemptive approach alters the deterministic system response only insignificantly and overshoot is hardly possible as opposed to the time triggered approach. The effect of white noise disturbances is investigated and the proposed approach again performs significantly better than its time triggered counterpart. Deterministic and stochastic analysis of the effect of jitter in higher order systems is presented and an example mechanical system is presented. Both deterministic and stochastic results point in favour of the non preemptive approach.

Altogether we find evidence that nonpreemptive priority based scheduling is well suited for control algorithms. In contrast to a static schedule, well known priority assignment schemes can be used. A slight difficulty is the default preemptive implementation of control tasks found e.g. in POSIX compliant operating systems like RT-Linux. Non preemptive kernels are typically lighter than their preemptive counterparts and thus more suitable for embedded applications. From a performance, view point, context switching is typically lighter in non preemptive systems. Preemptivenes typically generates platform dependence, so migrating preemptive kernels to alternative embedded environments may by unessesarily tedious.
The example systems presented are simple though representative relevant control system existing. The results presented in this work call for significant generalization higher order controllers, e.g. with integral action or oberver schemes, coloured noise disturbances, measurement noise, open loop unstable systems and even non linear systems. The immediate direction of our future work point towards the analysis of higher order state controlled systems with observers and controllers with integral action.

6. REFERENCES

[1] L. Palopoli et. al., Synthesis of Robust Control Systems under Resource Constraints, *Proceedings of the HSCC 2002* (Eds. C. Tomlin and M. Greenstreet), Lecture Notes in Comp. Sc. 2289, pp.337 - 350, Springer-Verlag 2002.

[2] C. L. Liu and J. W. Layland, Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment., *JACM*, pp.46-61,1973.

[3] C. M. Kirsch, T. Henzinger and B. Horowitz, Embedded Control System Development with Giotto, *Proceedings of the ACM SIGPLAN 2001 Workshop on Languages, Compilers and Tools for Embedded Systems*, 2001.

[4] J. P. Lehoczky, L. Sha and Y. Ding, The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour, *Proceedings of the 10th IEEE Real-Time Systems Symposium*, pp. 166-171, 1989.

[5] J. P. Lehoczky, Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines., *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pp. 201-209, 1990.

[6] G. F. Franklin and J. D. Powell, *Digital Control of Dynamic Systems*, Addison-Wesley, 1980.

[7] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages (3rd edition)* Pearson Education Ltd., 2001.

[8] M. Törngren, Fundamentals of implementing real-time control applications in distributed computer systems. *Journal of Real-Time Systems*, 14, 219-250, 1998.

[9] K-E. Årzen, B. Bernhardsson, J. Eker, A. Cervin, K. Nilsson, P. Persson and L. Sha, Integrated Control and Scheduling., *Research report* ISSN 0820-5316, Dept. Automatic Control, Lund Institute of Technology, 1999.

[10] B. Wittenmark, J. Nilsson and M. Törngren, Timing problems in real-time control systems, *American Control Conference*, Seattle, Washington, 1995.

[11] P. Marti, J.M. Fuertes, G. Fohler and K. Ramamritham, Jitter compensation for real-time control systems, *22nd IEEE Real-Time Systems Symposium*, pp. 39-48, 2001.

[12] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, On Task Shedulability in Real-Time Control Systems, *IEEE Real-Time Systems Symposium.* Washington, DC, Dec 1996.

[13] P. Albertos, A. Crespo, L. Ripoll, M. Valles and P. Balbastre, RT control scheduling to reduce control performance degrading, *39 th. IEEE Conference on Decision and Control*, Sydney, Australia , 2001.

[14] M. Morari and E. Safrirou, *Robust Process Control*, Prentice Hall, 1989.

[15] K. Zhou, J.C. Doyle, K. Glover, *Robust and Optimal Control*, Prentice Hall, 1995.

[16] J. Eker and S. A., Cervin, Matlab Toolbox for Real-Time and Control Systems Co-Design., *In Proceedings of the 6th. International Conference on Real-Time Computing Systems and Applications*, Hong Kong, China, 1999.

[17] K.-E. Årzen, A. Cervin, J. Eker and L. Sha, An Introduction to Control and Scheduling Co-Design., *39th IEEE Conference on Decision and Control*, Sydney, Australia, 2000

[18] L. David, F. Cottet and N. Nissanke, Jitter Control in On-line Scheduling of Dependent Real-Time Tasks., *22nd IEEE Real-Time Systems Symposium*, pp. 49-58, 2001 (RTSS 2001)

[19] H. Kopetz, *Real-Time Systems, Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, Boston, ISBN 0-7923-9894-7, 1997.

MODELLING SELF-TRIGGERED TASKS FOR REAL-TIME CONTROL SYSTEMS

Manel Velasco, Pau Martí, Josep Mº Fuertes.

Automatic Control and Computer Engineering Department. Technical University of Catalonia, Barcelona, Spain.
Tel:+34 93 401 79 74 /Fax:+34 93 401 70 45
{manel.velasco, pau.marti, josep.m.fuertes}@.upc.es

Abstract: In real-time control systems, the objective of control activities, i.e., to control processes, and the objective of scheduling techniques, i.e., to meet deadlines, are accomplished separately. This may derive in sub-optimal designs in terms of both control performance and resource utilization. Control activities optimise control performance regardless the computational demands of other tasks and scheduling techniques optimise the use of resources regardless the dynamics of the control application. To overcome this problem, we present a control-based model for control tasks in which computing resources and control performance are jointly considered. Concretely, the model allows each control task to trigger itself: at each control task instance execution, the executing instance informs the scheduler when the next instance should be executed. The next instance execution point in time is dynamically obtained as a function of the utilization factor and control performance. Preliminary results show that control activities, at run time, are able to define self-execution patterns that dynamically balance optimal levels of control performance and resource utilization.

Keywords: Real-Time Systems, Task Models, Control Systems, Nonlinear Discrete-Time Control Systems.

1. INTRODUCTION

In real-time control systems, the objective of control activities, i.e., to control processes, and the objective of scheduling techniques, i.e., to meet deadlines, are accomplished separately. This may derive in sub-optimal designs in terms of both control performance and resource utilization.

On one hand, control activities optimise control performance regardless the computational demands of other tasks. This fact comes from the control design process: a discrete time controller is designed assuming a constant sampling period. In terms of task execution, that means that at run time the controller will execute demanding a constant processing capacity. Therefore, in the design process of the controller, it is not usually taken into account the possibility of taking advantage of processing capacity that may be released by other tasks. That is, the controller design does not allow increasing the execution rate (decrease the task period) to exploit available resources.

On the other hand, scheduling techniques optimise the use of resources regardless the dynamics of the control application. For instance, a periodic control tasks may not require the designed execution rate (the assigned processing capacity) if the controlled plant is in equilibrium. When a plant is in equilibrium, after an execution of the controller, no major change in the state of the plant can be appreciated. That is, the contribution of the controller execution can be considered as useless. Therefore, the processing capacity has been used when not necessary. In such situations, this processing capacity could have been used by other tasks with higher processing capacity demands.

To overcome these problems, we present a control-based model for control tasks in which computing resources and control performance are jointly considered. Concretely, the model allows each control task to trigger itself: at each control task instance execution, the executing instance informs the scheduler when the next instance should be executed. The next instance execution point in time is dynamically obtained as a function of the utilization factor (global parameter) and control performance (local parameter)[1]. Therefore, task-timing constraints (e.g., task period) are dynamically adjusted.

Consequently, we could say that each control task acts as a co-scheduler, helping the scheduler at the scheduling decisions. Note that scheduling decisions will depend not only on task deadlines (as standard scheduling policies do) but also on the utilization factor and the dynamics of the control applications. Figure 1 illustrates the operation of the whole system.

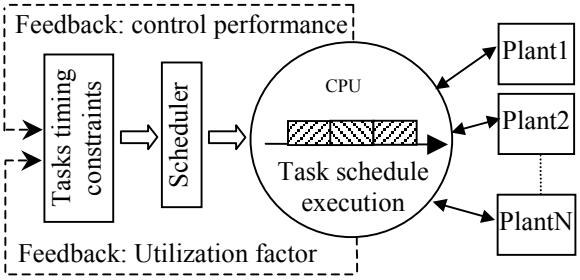

Figure 1. System operation model

Preliminary results show that control activities, at run time, are able to define self-execution patterns that dynamically balance optimal levels of control performance and resource utilization.

The rest of the paper is organized as follows. In section 2 we discuss the state of the art. In section 3 we provide basic background to define the problem formulated in section 4. In section 5 the self-triggered tasks model is developed. Preliminary simulations results are presented in section 6. Finally, in section 7 we conclude and point to future research work.

2. STATE OF THE ART

In this section we present a brief but relevant discussion on the state of the art. The model we present resembles the model presented in [CER02]. They propose to use feedback information from the controlled plants to take scheduling decisions. Specifically, all control tasks periods are proportionally enlarged or shorted at a given time instant as a function of the utilization factor. Therefore, they do not allow the exchange of processing capacity if the control application requires higher execution rates of specific tasks.

The later can be achieved using the elastic model of [BUT02]. In such model, the elastic coefficient of each task allows the scheduler change the task execution rate within specified ranges. The elastic coefficients are regarded as fixed parameters to be specified before run-time. The model we propose matches the elastic model if each elastic coefficient of control task would be treated as a dynamic parameter, being a function of the resource utilization and control performance.

It should be stressed that the work we present derives from [MAR02]. The authors point out that novel methods for control task scheduling in which scheduling decisions should depend on control performance and resource utilization are needed.

Some similarities may be identified between our model and event-based systems. In event-based systems the sampling period takes random values. The sampling period for our model is also variable, but there exist a slightly difference: in event-based

[1] The utilization factor of the system is obtained taken into account all tasks in the system. Therefore, it is a global parameter and affects all tasks. The control performance is obtained by each task from the corresponding controlled plant. Therefore, it is a local parameter and affects itself.

systems the next execution point in time is unknown; the suggested model in this paper deals with known future periods because they are a result of the model execution.

3. BACKGROUND

Traditionally, real-time tasks are characterised by the period (T_i) and the worst-case execution time (C_i) [BUT97]. For control tasks, the task period is given by the sampling period (h_i) that, among other parameters, determines the performance of the controlled system.

The sampling period h_i can be selected form a range of values. It has a lower limit (the shortest sampling period), which is given by the underlying technology (processing power). The shortest sampling period should include the worst-case execution time of the controller. This is represented in Figure 2 a) where all instances of a control task have as a task period equal to the worst-case execution time. However, other patterns of execution could be represented in Figure 2 by sequences like b) and c), where the task period is greater than the worst-case execution time (in Figure 2, boxes represent consecutive execution of instances of a task assuming worst-case execution times, C_i, and consecutive vertical doted lines represent the task period).

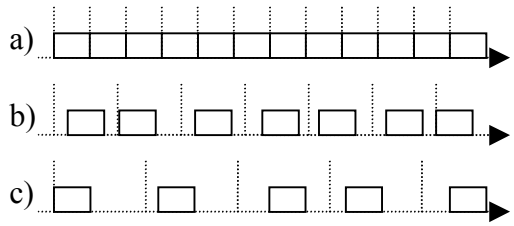

Fig.2. Possible periods for a control task. a) lower limit (shortest period), b) intermediate period and c) long period

The range of values for the sampling period has also an upper limit (the longest sampling period): beyond this limit, the control task looses the control of the controlled system. That is, the control tasks execution rate is too slow compared to the system dynamic, thus losing relevant information as illustrated in figure 3, where the effects of a low sampling rate are reflected. The dotted line shows the estimated sampled signal while the continuous line is the real signal. If the control task works with the dotted line, the information that the control task has is not accurate enough.

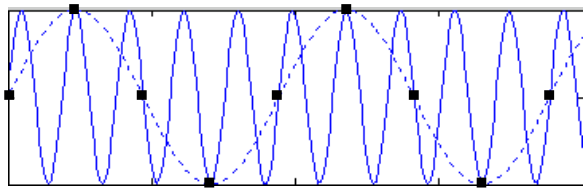

Fig.3. Real signal (continuous) and the estimation of the sampled signal (dotted). The little squares indicate the sample points.

More specifically, this upper limit depends on the natural frequency of the system to be controlled. This frequency can be easily found in different ways. For instance, the natural frequency of oscillation for a spring is determined by the constant of the spring (K) and the existent mass at his end (M). It also may be determined in an empiric way, stretching the spring lightly and observing the oscillation frequency of the system. The Shannon's theorem (see for example [PHI84]) tells us that the longest sampling period (the upper limit) should be at least given by (1), where f_0 denotes natural frequency.

$$h_i = \frac{1}{2 f_0} \qquad (1)$$

To avoid problems derived from working on limit situations, the recommended sampling period is usually taken following well-known rules-of-thumb (see for example [ÅST88][PHI84]). For example, a rule-of-thumb suggests taking a sampling period from 4 up to 20 times of theoretician one, as shown in expression (2)

$$h_i = \frac{1}{8 f_0} to \frac{1}{40 f_0} \qquad (2)$$

Each possible choice for the sampling period has advantages and disadvantages. In short, a short period allows a quick reaction in front of perturbations (which is positive from a control point of view), but increases the processor's load (which in negative from a resource utilization point of view). Using long periods decreases the processor's load but may give poor response in front of perturbations.
COMFORMATO
Note that the choice of the sampling period has to balance the desired control performance and the feasible computational demand. INCRUSTAR

4. PROBLEM FORMULATION

The selection of the execution period for a control task has been discussed in the previous section. The control engineer prefers small task periods in order to obtain good responses of the controlled system. The real-time engineer prefers long task periods to relax specifications and to facilitate task set schedulability. Both preferences are in conflict, and traditionally, a single value for the task period has to be choice before run-time [MAR01].

If we look at control tasks operation, we can distinguish two clearly differentiated situations that affect the execution rate of a control task. Firstly, we have the situation in which the controlled system is clearly outside of the desired working point. That is, the controlled system behaves differently as it should do. In this situation, a small period for the control task would imply a fast correction of the non-desired behaviour of the controlled system, bringing the system to the desired working point. Once the desired point is reached, no more correction is needed. Looking at control signals, this means that control actions tend quickly to zero. If control actions are

almost zero, a small period is no longer needed because the contribution of each control action has no effect on the controlled system (they are zero).

The opposed situation appears when the system is in the desired working point (in equilibrium). In such situation, as be stated previously, the period of the control tasks can be long because the contribution of control actions have no effect on the controlled system (they are zero). However, a perturbation may suddenly affects the controlled system, bringing the system away from the desired working point. Then, if the task period is large, the controlled system will respond slowly, taking long time before reaching again the equilibrium point, which may not be good for the given performance specifications.

In fact these two situations, which dynamically appear due to perturbations that affect the controlled system, can be considered as "transitory". Generally, the transition from one to the other can be considered as a soft evolution.

Note that from a control point of view, depending on the status of the system (depending on the situation that the controlled system presents), if the control tasks has a constant value for the task period, the real profit of the control task CPU cycles (given by the task period) can be very high or very low. When the controlled system is in equilibrium, from a control point of view, the CPU cycles of the control task are not fully exploited, thus wasting resources. In the opposite situation, the CPU cycles are appropriately exploited.

Concluding, long periods for control tasks are preferable when the controlled system is stable in the desired working point and small ones when the controlled system is far away from the desired working point. This demands models that can dynamically accommodate different values for the task (controller) period. However, traditional control and scheduling techniques do not provide this feature.

The model we present allow control tasks to have varying values for the period. The exact value for the period (at each control task instance execution) is dynamically adjusted depending on the controlled system status and the CPU load. Figure 4 shows the relation between the task period and the controlled system status. The curve at the bottom of the figure represents the status of the controlled system and the curve on the top of the figure represents the value for the task period for each instance execution. When a perturbation appears (when the bottom line presents a "mountain" shape) and the controlled system is brought away from the desired working point, the task periods are decreased. When the controlled system response reaches the desired working point (it has a horizontal shape) the task period is adjusted to the nominal value (longest period).

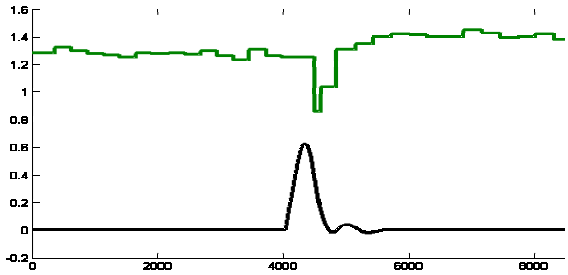

Fig.4. System evolution while period adjusting. Upper line corresponds to periods, lower line corresponds to system.

5. SELF-TRIGGERED TASK MODEL

The main idea of the model we present is to use the common models that are used in the analysis and design of control systems. Concretely, we propose to use an extended state-space representation. State-space models allow us to describe the future response of a system, given the present state (characterized by the state variables), the excitation inputs and the equations describing its dynamics (see [ÅST88] o [PHI84] for further reading on discrete-time state space models). The extension we suggest is to incorporate as new state variables the task period and the utilization factor. Therefore, we will be mixing the control behaviour (already represented in the state space model) with the execution rate of the task and the processing demand. In the following subsections, step-by-step, we develop the model.

5.1. Basic model

Let us think on a closed loop system formed by a ball and beam (see figure 5), which is the plant to be controlled, and a control task that has to be executed on a processor and has to control the ball and beam. The ball and beam system has a motor that balances (by rotating movements) a beam in order to keep the ball (that can rotate freely along the beam) in the desired beam position [AST88], as illustrated in figure 5..

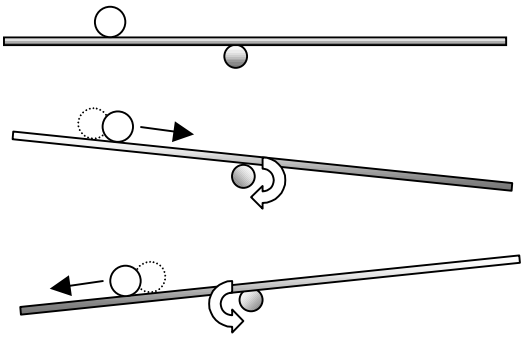
Fig.5 Ball and beam system.

The objective of the controller is to actuate on the motor to locate the ball in the desired position. To do so, at each sampling time, the controller takes the value of the position of the ball and the angle of the beam and generates the new angle for the beam that derives in the corresponding actuation on the motor.

The ball and beam system is habitually represented by the linear discrete-time invariant state-space model [AST88] given in (3)

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_k \\ y_k \end{bmatrix} + \begin{bmatrix} \frac{h^2}{2} \\ 2 \cdot h \end{bmatrix} U \qquad (3)$$

In equation (3), x_k and y_k (which are the state variables) represent the position of the ball and the beam angle respectively at the k sampling instant. The first matrix (2x2 dimension), called system matrix or state transition matrix, describes the dynamics of the ball and beam. The second matrix (2x1 dimension), called input coefficient matrix, links the input with the system dynamics. In both matrices, h is the sampling period. U is the available vector of inputs; in our case it is the tension (1x1 dimension) that we provide to the motor. The input can adopt positive and negative values, allowing the motor of the beam to rotate to both sides.

To control the system it is necessary to use a rule (a control law) that allows us to find values for U at each sampling instant that guaranties the desired behaviour for the system. By means of classical control techniques it is possible to design a control law that generates appropriate inputs U to the system in order to locate the ball at the desired position.

Note that in the state space representation of the ball and beam, the variables that describe the system are x_k, the angular position of the beam (angle), and y_k, the position of the ball on the beam. At each sampling instant, x_k and y_k vary according to the system dynamics (state transition matrix) and the input (applied through the input matrix). However, h, the sampling period, which appears on the matrices as a result of the discretization process, has a constant value that has been chosen at the controller design stage. Recall that (3) is a discrete-time model obtained via discretization of the continuous-time model. Therefore, h has nothing to do with the state of the system, although it influences its dynamics.

5.2. First model modification.

Up to now, we have described the classical ball and beam state-space representation. As we stressed in section 3, we want a model able to accommodate different values for the sampling period (i.e., the task period) according to the desired control performance and taking also into account the processor load.

The first extension for the previous model allows us to have varying sampling periods according the controlled system dynamics. To achieve this objective we extend the state representation of the ball and beam system model with a new system variable, the task period, h_k. Therefore, at each task instance execution, the task period will be changed according to the state of the system given by x_k, y_k and the new state variable h_k. This is intuitively expressed in (4)

$$h_{k+1} = h(x_k, y_k, h_k) \qquad (4)$$

Recall that the state of the system can be directly related to control performance. For instance, a simple rule could be *the smaller the norm of the state vector, the better the controlled system performance.* In terms of the ball and beam: the smaller the deviation of the beam with respect to the horizontal position and the smaller the distance of the ball with respect to the desired location, the better the performance. Therefore, at each task instance execution, the added state variable determines the next task instance execution point in time as a function of control performance.

It has to be pointed out that if the system state is increased by the new variable, a new control law giving the appropriate sequence of values for the input U is needed. The extended model is represented in (5)[2].

$$\left[\begin{array}{c} x_{k+1} \\ y_{k+1} \\ \hdashline h_{k+1} \end{array}\right] = \left[\begin{array}{cc:c} 1 & h_{k+1} & 0 \\ 0 & 1 & 0 \\ \hdashline \alpha & \beta & \omega \end{array}\right] \cdot \left[\begin{array}{c} x_k \\ y_k \\ \hdashline h_k \end{array}\right] + \left[\begin{array}{c} \frac{h_{k+1}^2}{2} \\ 2h_{k+1} \\ \hdashline 0 \end{array}\right] \cdot U \qquad (5)$$

Note that in the system given by (5), h_k is the new state variable. The dependency of this variable with the others system variables is given by parameters α, β and ω. Let's discuss some properties of the extended model, depending on values of α, β and ω:

- If α and β are set to zero and ω is set equal to 1, then, for each k, $h_{k+1}=h_k$, and the system may be considered as the original one (given in (3)). The control law that will give the sequence of inputs U can be obtained by classical controller design methods. This model will result on the classical real-time implementation with the task period equal to the sampling period (T=h).

- If α and β are set to zero and $0<\omega<1$, the sampling period will be decreased at each instance execution, tending to 0, thus leading to a non real system. From a schedulability point of view, at some point in time, the task period will be shorter than the task worst-case execution time, leading to an unfeasible system.

[2] Note that h_{k+1} (and not h_k) appears inside of the system and input matrices. This is due to the solution of the system equations. In the non-extended model, the sampling period of the system and input matrices has no index (k+1 or k) because it is constant (h at the k instant and h at the k+1 instant have the same value). In the present model, since h_k is a system variable that varies form instance execution to instance execution, it is necessary to distinguish in the matrices which is the appropriate k index.

- If α and β are set to zero and $\omega \geq 1$, the sampling period will be increased at each instance execution, tending to ∞, thus violating the Shannon's sampling theorem. Note that from a real-time point of view, this tendency could be desirable because implies less pressure on the use of resources.

- If α and β are set different from zero and $0<\omega<1$, we have a system with a variable period, each one depending on the previous system state. The computed value for each h is given by (6).

$$h_{k+1} = \alpha \cdot x_k + \beta \cdot y_k + \omega \cdot h_k \quad (6)$$

  The formula for h_{k+1} shows that the new value for the next task period depends on the previous one (ωh_k). This implies smooth transitions in period variations. This model will require real-time implementations able to accommodate varying task periods (which depend on the state variables including the previous period), similar, for example, to the models used in [BUT02]. The main problem of this combination of values for α, β and ω is that the state space model becomes nonlinear (that is, a small change in the inputs may result in chaotic outputs), as we outline later in this section. Therefore, finding the adequate control law giving the appropriate sequence of inputs U will be a more difficult task (see for example [ISI89]), if feasible.

- If α and β are set different from zero and ω is set to zero, we get a variable period system depending only on the original state variables. Consequently, the more quickly these variables move (angle and position), the faster the period changes, thus loosing the smooth transitions found in the previous case. This may result in values for the sampling periods out of the permissible ranges (that we explained in section 2), which from a control point of view may violate the limit given by Shannon and from a real-time point of view, it may result in an unfeasible schedule (if the period is shorter than the worst-case execution time).

- If α and β are set different from zero and $\omega \geq 1$, the evolution of the system will depend on the chosen values, which require a deeper analysis, out of the scope of this paper.

From the extended model given by (5), two elements should be highlighted:

- The system is nonlinear, since h is a state variable and it also appears multiplying to other state variables.

- It would be possible to obtain negative values for the task period from the actual extended state space model. Considering only a theoretical view, this possibility means that the system should return to the past in order to modify already taken decisions. But this is clearly non-programmable in a real system. We could solve this problem by using the absolute value for the *h* at each task instance execution. This will guarantee that h will be always positive.

Taking into account the previous points, the model should be modified to the expression given by (7).

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ |h_{k+1}| \end{bmatrix} = \begin{bmatrix} 1 & |h_{k+1}| & 0 \\ 0 & 1 & 0 \\ \alpha & \beta & \omega \end{bmatrix} \cdot \begin{bmatrix} x_k \\ y_k \\ h_k \end{bmatrix} + \begin{bmatrix} \frac{h_{k+1}^2}{2} \\ 2|h_{k+1}| \\ 0 \end{bmatrix} \cdot U \quad (7)$$

In (7), $|h_{k+1}|$ means absolute value of h_{k+1}. Note that in this expression, h always is positive, due to the absolute value operator. Note also that h in the k instant does not have the absolute value operator because it comes from the previous task instance execution.

Finally, is also have to be stressed that the system matrix has h_{k+1} at the k instant, which is an inconsistency. However, this can be easily solved by substituting the h_{k+1} value for the expression given in (6), which is already known at the k instant.

5.3. Second model modification

Looking at the final model given by (7), two difficulties, beyond having a nonlinear model, can be identified:

- the absolute value operator makes the mathematical tractability implies using two symmetric models, one for positive values of h and the other for negative values. From a programmable point of view, this involves no major problems (a if-then-else structure is required). However, from a control tractability viewpoint, this model duplicity is not desirable because it requires using specific control methods like switching mode controllers (see [LEI03] for a benchmark study of switching techniques).

- The possible values that h may take are not bounded, due to the linear relation between h and the original state variables. Note that if the state variables take huge values, h will rapidly increase (and viceversa). As we outlined in the previous section, this introduces control and real-time problems.

To solve the previous problems, we suggest to bound the possible h values by introducing an appropriate function of the state variables instead of having a simple linear relation. We call this function *h-function*.

Taking advantage of the use of the h-function, we incorporate the utilization factor in the model, as a measure of the processing capacity. Recall that up to now, the model only related the varying period of the task with the original state variables (as a measure of control performance). In this new extension of the model (adding the h-function), we will relate the period variation to the control performance as well as to the processing capacity.

The second extension we present is given by (8)

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ h_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + y_k \cdot f(x_k, y_k, h_k, \zeta) \\ y_k \\ f(x_k, y_k, h_k, \zeta) \end{bmatrix} + \begin{bmatrix} \frac{f(x_k, y_k, h_k, \zeta)^2}{2} \\ 2f(x_k, y_k, h_k, \zeta) \\ 0 \end{bmatrix} \cdot U \quad (8)$$

In (8), ζ represents the utilization factor of the processor at the k instant and the h-function is given by $f(.)$. This new model is obtained as a natural extension of the previous one. In the previous section h_{k+1} was obtained as lineal combination of the other state variables. In the new extension h_{k+1} is obtained by an appropriate function of the state of the controlled system and the CPU load. Note that the goal of the h-function is to allow the task period to take values from a bounded range.

This allows a grater variety of possibilities in the selection of how h changes at each moment. For instance, the same system with two different h-functions will result in very different behaviours in terms of CPU load and control performance. Note that choosing a specific h-function could facilitate system schedulability (this could be done either offline or even online) as well as improve control performance.

It is important to point out that for the state space model given by (8), the analysis and design of a control law can be a complex task. However, it is possible to design control laws that guarantee the complete stability of the system around a desired working point. These techniques range from the system linealization [ISI89] to the complex techniques of feedback linealization for discrete systems [NIJ90].

5.4. The selection of the h-function

As we outlined before, the selection of the h-function determines controlled system performance and CPU load. Therefore, it is a crucial design choice. The most natural way for selecting the h-function is to mathematically translate in a function the following desired rule: as the system gets closer to the desired working point (equilibrium), the period should be as larger as possible, keeping Shannon limit (recall discussion of section 3). If a perturbation appears on the system, bringing it away from the equilibrium point, the period should be decreased (to improve control performance) taking into account the available processing capacity. Mathematically, this can be accomplished by the h-function given by (9)

$$h_{k+1} = f(x_k, y_k, h_k, \zeta_k) = e^{-(x_k^2 + y_k^2)} g(\zeta_k) \quad (9)$$

Note that (9) has a negative exponential shape, with two main parts. The fist part is the exponential function, which contributes to the h values taking only into account, measures of control performance. It allows each value to smoothly vary form an upper limit to a lower limit, if the exponent of the exponential function is kept positive. That's why we suggest putting the square exponents over x_k and y_k to convert the possible negative values (of the state variables) into positive ones. Note that as we explained in section 5.2., in this case, the state variables already are measure of control performance. Otherwise, the exponent should include the operation needed to measure the controlled system performance. The second part, the function $g(\zeta_k)$, allows to correct the next value of h taking into account the processor's utilization factor.

The fig.6 shows possible ranges of h values obtained using the h-function given by (9). The figure has two degrees of freedom: the control performance (horizontal axis) and the utilization factor (which would correspond to a perpendicular axis). The results are given according to the vertical axis, which are the possible values for the task period. When the control performance (understood as the system deviation with respect to the equilibrium point) decreases (the deviation increases), the values for the task period tend to short values, with the aim of quickly correct the deviation. In addition these values are increased as the load increases.

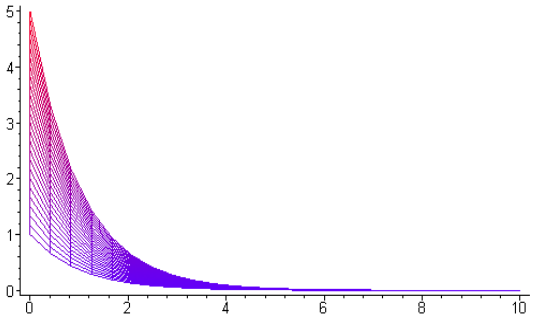

Fig.6. Possible values for the task period

5.5. Concluding remarks

The use of tasks that are driven by the model given by (8) have a remarkable advantage:

The task observes the state of the system which is composed by the processor and the controlled system.

In this way, at each control task instance execution, the period selection is in consonance with all the elements that are involved in the control of the system

and in the scheduling to the task set, thus facilitating the optimisation of the whole system, in terms of both control performance and resource utilization.

The main goal of the presented model is that if several tasks are driven according to this model, the processing capacity can be dynamically balanced among them according to the controlled performance measured (see simulation results section). Moreover, note that although the processing capacity is dynamically exchanged, the feasibility of the task set is kept. If more tasks are added to the system, the probability of keeping a feasible schedule is high due to the fact that decisions (on the period selection) are based on the utilization factor.

6. SIMULATION RESULTS

In figure 6 we show the results of two *ball and beam* control tasks executing a single processor. The control laws implemented in the two tasks have been calculated by means of linealization techniques (not detailed in this paper), according to model we have presented.

In Figure 6 we can observer 4 lines. The upper ones correspond to the sequences of values for each task period, and the lower ones correspond to the dynamics of each controlled system. The task period values take into account the utilization factor, which is injected as a simulation variable. At the beginning (left side of the figure), both systems are stable at the desired working point, so both have the same value for the execution period.

Later on a perturbation affects system 1, deviating the system away form the desired working point. This perturbation causes an immediate decrease of the task period controlling system 1 and an increase of the task period controlling 2. Therefore, the exchange of the processing capacity among the two control tasks has started. From

The delay observed between the perturbation arrival time and the first decrease in the task period is due to two factors:

- There is always an offset among the moment in which the perturbation takes place and the moment in which the task samples the system. In the worst case, this offset could be as big as the period. However, this situation doesn't necessarily lead to worse system dynamics
- After the perturbation arrival time the error is small and the decrease in the sampling period is not very significant. One period later, the error has increased and the decrease on the sampling period becomes more remarkable.

Note that the communication between both tasks only takes place through the processor's utilization factor, which is a global parameter (see section 1 for further details).

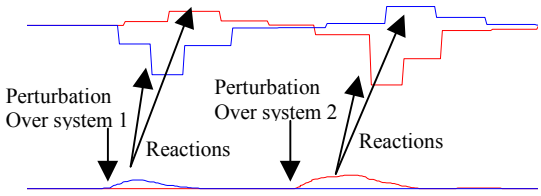

fig 7. Analisys of responeses and periods enlargements.

Later on another perturbation affects system 2. The system reacts in a similar way to the previous one. That is, the processing capacity exchange takes place as before, but in inverse direction.

7. CONCLUSIONS

In this paper we have presented the self-triggered task model that drives control task executions according to controlled system performance and available processing capacity. Specifically, the model allows control task to adjust their execution rate, acting as a co-scheduler.

As we outlined, the main research issues behind this work is the analysis and design of the controller, which must give the appropriate inputs to drive the whole system to the desired behaviour.

ACKNOWLEDGEMENTS

This research has received support from the Spanish Ministerio de Ciencia y Tecnologia Project ref. DPI2002-01621.

REFERENCES

[ÅST88] Krarl J.Åström andBjörn Wittenmark."Sistemas controlados por computador",ISBN:84-283-1593-0, Paraninfo, 1988

[BUT97] Buttatzzo C. "Hard Real-Time Computing Systems". Kluwer academic publishers, 1997

[BUT02] Buttatzzo, G. Lipari, G. Caccoamo, M. Abeni, L "Elastic Scheduling for Flexible Workload Management". IEEE Trans. on Computers, Vol. 51, NO. 3, March 2002

[CER02] A. Cervin, J. Eker, B. Bernhardsson, K.-E. Årzén "Feedback-Feedforward Scheduling of Control Tasks", *Real-Time Systems,* **23:1**, 2002

[ISI89] Isidori, A. Nonlinear Control Systems. Spring Verlag, New York, 1989

[LEI03] Leith, D.J., Shorten, R.N., Leithead, W.E., Mason, O., and Curran P. "Issues in the design of switched linear control system: A benchmark study". Int. Journal of Adaptive Control and Signal Processing. 2003; 17:103-108

[MAR01] Martí, P. Fohler, G. Ramamritham ,K. Fuertes, JM. Jitter Compensation for Real-time Control Systems. 22nd IEEE Real-Time Systems Symposium, London, UK, December 2001.

[MAR02] Martí, P. Fohler, G. Ramamritham ,K. Fuertes, JM. Jitter Compensation for Real-time Control Systems.

22nd IEEE Real-Time Systems Symposium, London, UK, December 2001.

[NIJ90] Nijmeier, H. "Nonlinear dynamical control systems", Springer-Verlag, 1990.

[PHI84] Philips, C.L and Troy, N. G.. "Digital control systems, analysis and design", Prentice-Hall, Englewood Cliffs, N.J., 1984.

Using Jitterbug to Derive Control Loop Timing Requirements

Anton Cervin

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

Abstract

Linking scheduling attributes to control performance specifications is a difficult problem. This paper discusses how the MATLAB toolbox Jitterbug can be used to derive timing requirements for control loops from various control performance specifications. The resulting timing requirements include specifications on sampling periods, latencies, and jitter. An overview of the Jitterbug approach is given, and limitations of the tool are pointed out. A control design example is given, and, finally, topics where more research is needed are outlined.

1. Introduction

The design of a real-time control system is essentially a codesign problem, where limited resources should be allocated to control tasks and other tasks such that optimum overall performance is achieved. In this paper, we will focus on the control and scheduling codesign problem. More specifically, we will deal with the problem of scheduling-induced jitter in periodic control loops.

A digital controller is normally designed assuming a fixed sampling period T, and, possibly, assuming a fixed computational delay τ. These simplistic design assumptions are seldom met in the target system. When executing as a task in a real-time system, the controller will suffer from time-varying latencies, induced by preemption from interrupts and higher-priority tasks. The result is degraded control performance. Some performance degradation is normally acceptable, as long as the controller meets its design specifications.

In the scheduling design, a controller is traditionally described as a periodic task with a period T, a deadline D, and a computation time C. It is normally assumed that $D = T$, although a shorter deadline can be used to limit the end-to-end latency in the controller. It can be argued that the traditional timing model is too simplistic, since it does not reflect the fact that a controller is composed of (at least) three distinct operations: the input operation (or *sampling*), the control computation, and the output operations (or *actuation*). To get better control of the latency and jitter in the controller, it is possible to schedule the different part of the controller as separate tasks. Subtask scheduling of control tasks has been treated in, e.g., [Crespo *et al.*, 1999] and [Cervin, 1999]. These papers references assume a particular scheduling policy and that the sampling periods of the controllers are fixed at the scheduling design stage. In reality, the sampling period of the controllers are also design parameters. The sampling periods are typically chosen according to rules of thumb. One such rule [Åström and Wittenmark, 1997] states that the sampling period T should be chosen such that

$$\omega_b T \approx 0.2\text{–}0.6, \tag{1}$$

where ω_b is the bandwidth of the closed-loop system. It should be noted that faster sampling may be required if there is latency and jitter in the control loop.

In order to make correct trade-offs in the scheduling design, the designer needs to what ranges of sampling periods, latencies, and jitter that are acceptable to each control loop. In [Bate *et al.*, 2003], time-domain analysis involving extensive simulations are used to derive timing requirements for digital controllers. In contrast to that work, this paper relies entirely on analytical computations of cost functions and frequency responses to derive the timing requirements. the Jitterbug toolbox [Lincoln and Cervin, 2002] linear model,

The rest of this paper is outlined as follows. In the next section, an overview of control loop timing and its relation to control performance is given. In Section 3, it is described how Jitterbug can be used to model the timing variations in a control loop. Also, an overview of the control design criteria that can be evaluated using Jitterbug are given. In Section 4, the approach is exemplified on a control application, deriving bounds on the sampling period, latency, and jitter given a performance specification. In Section 5, the problem of linking timing requirements to scheduling analysis is discussed. Finally, in Section 6, some concluding remarks are given, and areas where further research are needed are outlined.

2. Control Loop Timing

A control task generally consists of three distinct operations: input data collection, control algorithm computation, and output signal transmission, see Figure 1. The timing of the operations are crucial to the performance of the controller. Ideally, the control algorithm should be executed with perfect periodicity, and there should be zero delay between the reading of the inputs and the writing of the outputs. This will not be the case in a real implementation, where the execution and scheduling of tasks introduce la-

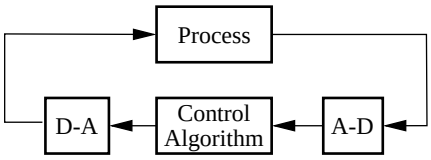

Figure 1 A computer-controlled system. The control task consists of three distinct parts: input data collection (A-D), control algorithm computation, and output signal transmission (D-A).

tencies.

The basic timing parameters of a control task are shown in Figure 2. It is assumed that the control task is released periodically at times given by $r_k = kT$, where T is the sampling interval of the controller. Due to preemption from other tasks in the system, the actual start of the task may be delayed for some time L_s. This is called the *sampling latency* of the controller. A dynamic scheduling policy will introduce variations in this interval. The *sampling jitter* is quantified by the difference between the maximum and minimum sampling latencies in all task instances,

$$J_s \stackrel{\text{def}}{=} L_s^{max} - L_s^{min}. \tag{2}$$

Normally, it can be assumed that the minimum sampling latency of a task is zero, in which case we have $J_s = L_s^{max}$.

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

$$J_{io} \stackrel{\text{def}}{=} L_{io}^{max} - L_{io}^{min}. \tag{3}$$

In general terms, the performance of a digital controller depends on the sampling period and the particular sequences of sampling and input-output latencies, $\{L_s^k\}$ and $\{L_{io}^k\}$. From the controller's point of view, the time-varying latencies can be viewed as random variables (that are independent between periods). Under the simplifying assumption that the distributions of the latencies can be sufficiently accurately described by their minimum and maximum values, the performance J of the controller can be expressed as a function of the sampling period T, the minimum input-output latency L_{io}^{min}, the sampling jitter J_s, and the input-output jitter J_{io}:

$$J = J(T, L_{io}^{min}, J_s, J_{io}). \tag{4}$$

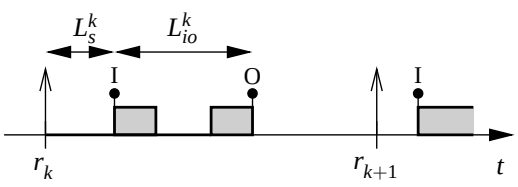

Figure 2 Digital controller timing. Each period, the controller experiences sampling latency, L_s, and input-output latency, L_{io}.

The goal of the analysis in the next section is to derive bounds on T, L_{io}^{min}, J_s, and J_{io} from various control performance specifications.

3. Analysis Using Jitterbug

Jitterbug [Lincoln and Cervin, 2002] is a MATLAB-based toolbox that is used to analyze linear control systems with time-varying delays. The control system is described by a number of connected continuous-time and discrete-time linear systems, representing the plant and the controller. In the simplest case, a periodic timing model with random delays is used to describe the execution of the discrete-time systems, i.e., the control task.

A Jitterbug model corresponding to the computed-controlled system in Figure 1 is shown in Figure 3. The signal model consists of three connected linear systems. The process is described the continuous-time system $G(s)$. The digital controller is described by two discrete-time blocks, *Samp* and $C(z)$. The first block models the sampling operation, while the second block represents the control algorithm and the actuator. (Implicit in each discrete-time block is a sampler at the input and a zero-order-hold circuit at the output.) The associated timing model consists of three nodes. The first node is periodic (with a given period T) and represents the release of the control task. There is a random delay L_s until the second node where H_1 is updated, and another random delay L_{io} until the third node where H_2 is updated.

In general, Jitterbug can accept arbitrary probability density distributions in the timing model. Here, to limit the design space, we let the controller timing be described by the variables T, L_{io}^{min}, J_s, and J_{io} only. Furthermore, we assume that the latencies are uniformly distributed between their minimum and maximum values. Hence, we let

$$L_s \in U(0, J_s), \tag{5}$$

and

$$L_{io} \in U(L_{io}^{min}, L_{io}^{min} + J_{io}), \tag{6}$$

where $U(a, b)$ denotes a uniform probability distribution between a and b. It should be noted that these uniform latency distributions are quite "nice" to the control loop. A more malign choice of distributions would be to let the

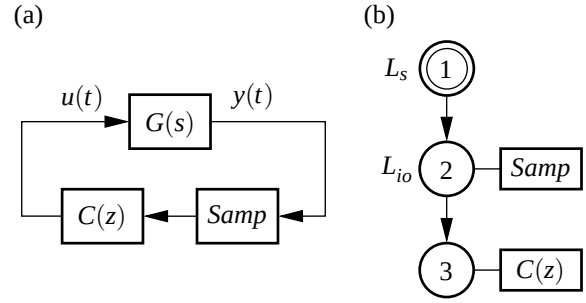

Figure 3 Jitterbug model of a digital control loop: (a) signal model, and (b) timing model.

latencies vary between the extreme points only. This could result in quite conservative timing requirements, however.

3.1 Performance Criteria and Jitterbug

Below, an overview of the control performance criteria that can be evaluated analytically using Jitterbug are given. Control design always involves trade-offs between various design specifications. A good overview of common performance specifications in computed-controlled systems is given in [Wittenmark *et al.*, 2002]. More material on trade-offs in linear control design can be found in [Boyd and Barratt, 1991].

Stability. A first requirement for any control loop is that it is stable. This property is always checked by Jitterbug. However, since the system is stochastic (due to the time-varying delays), Jitterbug only guarantees so called *mean square stability* of the closed-loop system. This means that there might exist particular sequences of delays and noises that make the system go stable, although the probability of this is zero. (For further discussion on different stability concepts, see [Ji *et al.*, 1991].)

Quadratic Cost Functions. The main purpose of Jitterbug is to facilitate control performance analysis via the computation of *quadratic cost functions*. Such functions are commonly used evaluate the performance of linear controllers. External inputs (reference signals and disturbances) are modeled as white noise processes that enter the control loop at various points. Given a model, Jitterbug can compute a stationary cost function on the form

$$J = \lim_{t\to\infty} \frac{1}{t} \int_0^T x^T(s) Q x(s)\, ds, \tag{7}$$

where x is the state vector for the total system (including the plant states and the controller states) and Q is a chosen semi-definite weighting matrix.

In the LQG design method, a linear controller is explicitly designed to minimize a quadratic cost function. It is then natural to use the same cost function when evaluating the performance of the controller. Given a nominal design, a typical performance specification could be to allow the value of the cost function to increase by, e.g., 10 percent due to scheduling-induced latencies. The same approach can be used also for other linear controllers.

Frequency-Domain Specifications. A classical approach to control design is to use frequency-domain specifications. With Jitterbug, it is possible to compute the *magnitude* of various closed-loop transfer functions, also in the presence of jitter.

Let the sampled-data representation of the process be $P(z)$, while the control algorithm is given by $C(z)$. The response of the closed-loop system is then completely characterized by the four transfer functions

$$\begin{aligned} H_1(z) &= \frac{1}{1+C(z)P(z)}, \quad H_2(z) = \frac{C(z)}{1+C(z)P(z)}, \\ H_3(z) &= \frac{P(z)}{1+C(z)P(z)}, \quad H_4(z) = \frac{C(z)P(z)}{1+C(z)P(z)}. \end{aligned} \tag{8}$$

Performance requirements are commonly expressed as requirements on the magnitudes of these functions. For instance, for reference signal tracking, it can be required that H_4 has a certain bandwidth (i.e., that the magnitude stays above -3 dB up to a certain frequency). The response to input load disturbances is given by H_3, and is typically required to be low at low frequencies, and so on.

Formally, transfer functions are only defined for linear, time-invariant systems. However, using the concept of *spectral densities*, Jitterbug can also compute the frequency response of systems with jitter. Given a time-invariant closed-loop system $H(z)$ which is excited by discrete-time white noise with unit intensity, the spectral density ϕ_y of the output is given by

$$\phi_y(\omega) = |H(e^{i\omega})|^2. \tag{9}$$

We hence can find the magnitude of the frequency response by

$$|H(e^{i\omega})| = \sqrt{\phi_y(\omega)}. \tag{10}$$

For systems with jitter, $\phi_y(\omega)$ is still defined, and, furthermore, it can be computed with Jitterbug. The quantity $\sqrt{\phi_y(\omega)}$ should then be interpreted as the average gain of the closed-loop system at a given frequency.

Robustness Measures. Two common robustness measures for control systems are the maximum sensitivity and the maximum complementary sensitivity. The sensitivity function is defined as

$$S(z) = \frac{1}{1+C(z)P(z)}, \tag{11}$$

and the maximum sensitivity is given by

$$M_s = \max_\omega |S(e^{i\omega})|. \tag{12}$$

For linear, time-invariant systems, $1/M_s$ can be interpreted as the distance from the loop gain $C(z)P(z)$ to the instability point -1 in the Nyquist diagram. Similar to above, using spectral density calculations, an interpretation for systems with jitter is also possible.

Likewise, the complementary sensitivity function is given by

$$T(z) = \frac{C(z)P(z)}{1+C(z)P(z)}, \tag{13}$$

and the maximum complementary sensitivity by

$$M_t = \max_\omega |T(e^{i\omega})|. \tag{14}$$

Common design specifications for M_s and M_t are in the range of 1.2 to 2.0.

3.2 Limitations of Jitterbug Approach

A number of limitations with the proposed approach exist:

- The timing model in Jitterbug is quite simplistic, in that the delays are assumed to be independent from period to period. Hence, the model can not fully describe the timing variations introduced by a dynamic

scheduling algorithm. Also, the tool cannot be used to analyze systems where the scheduling parameters change over time (as in feedback scheduling applications).

- The toolbox only computes a *mean* performance index, averaged over an infinite time horizon. Stability is only guaranteed in the *mean square* sense, i.e., the system might become unstable for a particular (but highly unlikely) sequence of delays.
- There is no time-domain analysis in Jitterbug. It is for instance not possible to give specifications on rise-time or maximum overshoot. However, time-domain control specifications can often be translated into frequency-domain specifications, see [Boyd and Barratt, 1991].
- In Jitterbug, it is necessary to specify the distribution of the sampling and input-output latencies. Since these are generally unknown, certain probability distributions must be assumed. Uniform distributions (which are used here) might be too benign, whereas end-point distributions might be to pessimistic.

4. Example

In this section, a design example is given, where we consider LQG (linear-quadratic-Gaussian) control of a servo process, described by the continuous-time transfer function

$$P(s) = \frac{1000}{s(s+1)}.$$

The process is assumed to be disturbed by continuous-time white input noise and with unit variance and discrete-time measurement noise with a variance of 0.1. An LQG controller, denoted $C(z)$, is designed using a sampling interval of T and an assumed a constant input-output latency of L. The controller is designed to minimize the continuous-time cost function

$$J = \lim_{t\to\infty} \frac{1}{t} \int_0^T \left(y^2(s) + u^2(s) \right) ds. \qquad (15)$$

using the Jitterbug command `lqgdesign`. The Jitterbug model of the control system was shown in Figure 3.

4.1 Cost Function Specification

First, we consider a cost function specification, where the value of the cost (15) is evaluated for different values of T, L_{io}^{min}, J_s, and J_{io}. From initial design attempts and time-domain simulation, it has been decided that a cost of at most $J = 100$ gives acceptable performance for the control loop. (Remember that a lower cost means better performance.)

By fixing two of the timing parameters, the cost as a function of the remaining parameters can be illustrated in a diagram. In Figure 4, the cost has been computed as a function of T and L_{io}^{min}, assuming zero sampling jitter and zero input-output jitter. It is seen that the control loop is quite sensitive to input-output latency, even though the controller has been designed to compensate optimally for

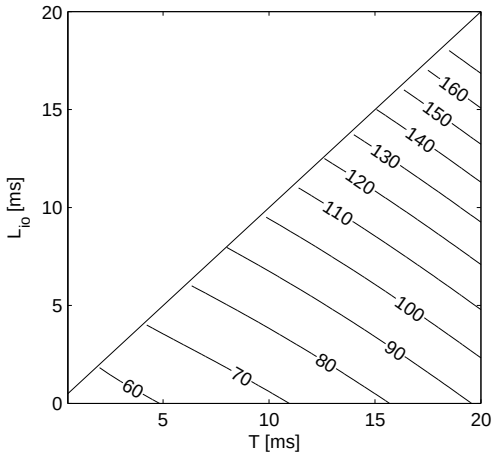

Figure 4 Cost as a function of T and L_{io}^{min}, assuming $J_s = 0$ and $J_{io} = 0$.

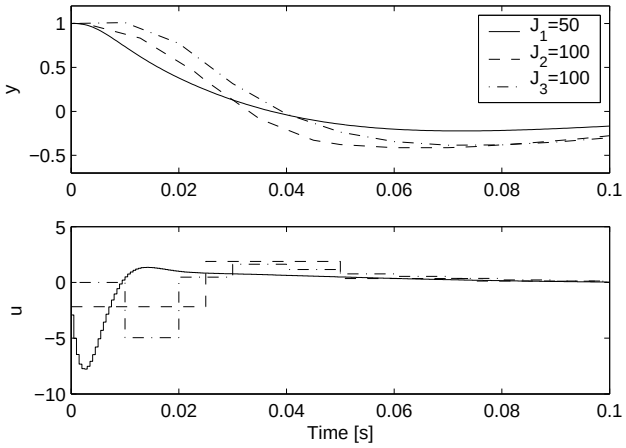

Figure 5 Examples of time-domain control performance corresponding to different values of the cost function.

the delay. If a delay is present, faster sampling is required to obtain a cost below 100.

To illustrate what the cost function means, control designs corresponding to three points in the design space in Figure 4 have been evaluated in time-domain simulations. In Figure 5, the responses to an impulse disturbance at time zero have been plotted:

- The full response has the cost $J_1 = 50$, corresponding to the parameters $T = 0.5$ ms and $L_{io} = 0$.
- The dashed response has the cost $J_2 = 100$, corresponding to the parameters $T = 25$ ms and $L_{io} = 0$.
- The dot-dashed response has the cost $J_3 = 100$, corresponding to the parameters $T = 10$ ms and $L_{io} = 10$ ms.

Next, the impact of sampling jitter and input-output jitter on control performance is studied. The sampling period has been fixed to $T = 20$ ms and the minimum latency is set to zero (corresponding to the lower-right corner of Figure 4). The controller is designed assuming a constant latency

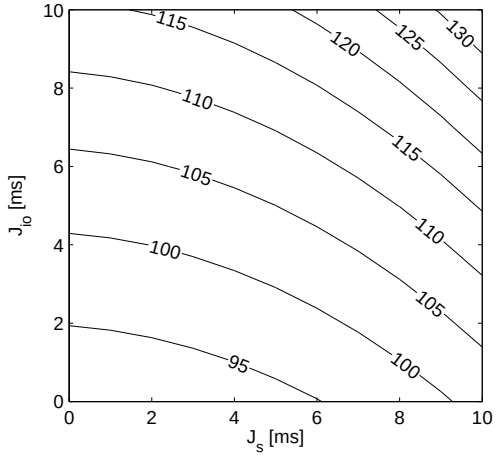

Figure 6 Cost as a function of sampling jitter and input-output jitter, assuming $T = 20$ ms and $L_{io}^{min} = 0$. The controller is designed assuming a constant delay of $J_{io}/2$.

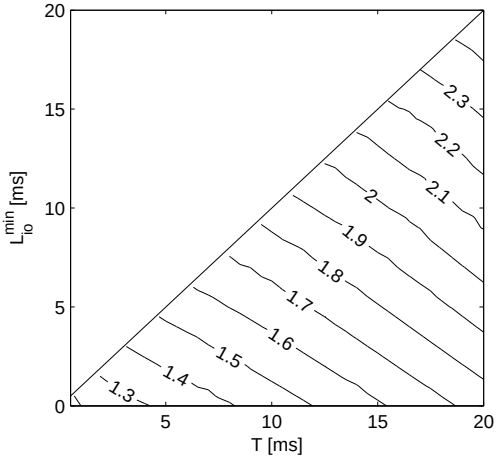

Figure 7 Maximum sensitivity as a function of T and L_{io}^{min}, assuming $J_s = 0$ and $J_{io} = 0$.

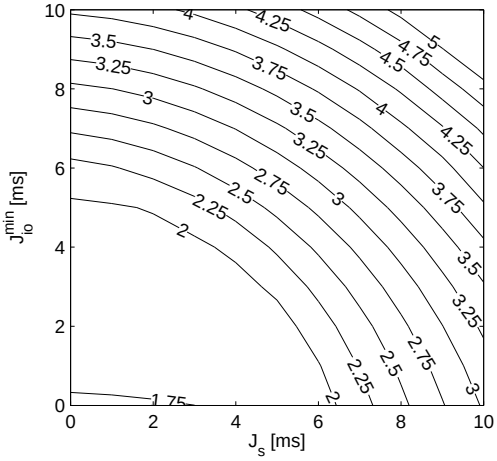

Figure 8 Maximum sensitivity as a function of sampling jitter and input-output jitter, assuming $T = 20$ ms and $L_{io}^{min} = 0$. The controller is designed assuming a constant delay of $J_{io}/2$.

equal to $J_{io}/2$. The resulting cost is shown in Figure 6. It is seen that, in this example, the control loop is more sensitive to input-output jitter that to sampling jitter. To keep the cost below 100, the both jitters must be less than a fraction of the sampling interval.

4.2 Robustness Specification

It can sometimes be difficult to select a reasonable upper bound on a cost function, especially in cases where the controller was not directly designed using a cost function (as it is in LQG-control). An interesting alternative is to instead place a bound on the sensitivity function (and possibly also the complimentary sensitivity function). This is often a more simple task, since a value of the maximum sensitivity, M_s, can be chosen independently of the size of the plant and controller parameters.

To continue the example, we assume that a reasonable value of M_s is 2.0. Similar to above, the value of M_s is evaluated as a function of T, L_{io}^{min}, J_s, and J_{io}. In Figure 7, M_s has been computed as a function of T and L_{io}^{min}, assuming zero sampling jitter and zero input-output jitter. Compared to Figure 4, we obtain similar bounds on the timing parameters.

Next, the maximum sensitivity is computed as a function of the amount of sampling jitter and input-output jitter. As before, the sampling period is set to $T = 20$ ms and the minimum latency is set to zero. The result is shown in Figure 8. According to this measure, the system is quite sensitive towards both sampling jitter and input-output jitter (compare with Figure 6).

5. Linking Scheduling Analysis to Controller Timing

The above analysis has assumed that values of T, L_{io}^{min}, J_s, and J_{io} are given. Assuming a controller task set and standard fixed-priority scheduling, the values of L_s^{max}, L_{io}^{min}, and L_{io}^{max} can be found using worst-case and best-case response-time analysis [Joseph and Pandya, 1986; Redell and Sanfridson, 2002]:

$$L_{s_i}^{max} = \sum_{j \in hp(i)} \left\lceil \frac{L_{s_i}^{max}}{T_j} \right\rceil C_j. \tag{16}$$

$$L_{io_i}^{max} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{L_{io_i}^{max}}{T_j} \right\rceil C_j. \tag{17}$$

$$L_{io_i}^{min} = C_i^b + \sum_{j \in hp(i)} \left\lceil \frac{L_{io_i}^{min} - T_j}{T_j} \right\rceil C_j^b, \tag{18}$$

Here, C_b^i denotes the *best-case* execution time of task i. As pointed out before, for more accurate analysis, it would be necessary to have the distributions of the latencies as well. This is an area where statistical approaches to scheduling analysis could be used. Also, results regarding minimum response times are lacking under EDF scheduling.

A difficult part of the codesign process is to modify the scheduling parameters such that all performance specifica-

tions are met. For this purpose some kind of search procedure must be used. One problem is that the timing attributes (T, L_{io}^{min}, J_s, and J_{io}) depend on the scheduling parameters (T, D, C) in a very nonlinear manner. Other scheduling policies than fixed-priority scheduling could give simpler design problems. One example is the Control Server model [Cervin and Eker, 2003], where T and L_{io} are determined directly by the task utilization factor U.

6. Conclusion

We have described how Jitterbug can be used to derive timing requirements from control performance specifications. The derived requirements are expressed in terms of the sampling interval, the minimum input-output latency, the sampling jitter, and the input-output jitter. The performance specifications can be given in terms of a quadratic cost function, or as constraints on the magnitude of certain closed-loop transfer functions (e.g., the sensitivity function). The analysis is only approximate, since it assumes that the delays introduced by the scheduling can be described by independent random variables with uniform distributions. Jitterbug allows for arbitrary distributions to be used, but the current state of the art in scheduling analysis does not allow the delay distributions to be derived.

References

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

Bate, I., P. Nightingale, and A. Cervin (2003): "Establishing timing requirements and control attributes for control loops in real-time systems." In *Proceedings of the 15th Euromicro Conference on Real-Time Systems.* Porto, Portugal.

Boyd, S. P. and C. H. Barratt (1991): *Linear Controller Design—Limits of Performance.* Prentice Hall.

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

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

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

Ji, Y., H. Chizeck, X. Feng, and K. Loparo (1991): "Stability and control of discrete-time jump linear systems." *Control-Theory and Advanced Applications*, **7:2**, pp. 247–270.

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

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.

Redell, O. and M. Sanfridson (2002): "Exact best-case response time analysis of fixed priority scheduled tasks." In *Proc. 14th Euromicro Conference on Real-Time Systems.* Vienna, Austria.

Wittenmark, B., K. J. Åström, and K.-E. Årzén (2002): "Computer control: An overview." Technical Report. IFAC professional brief.