Linköping Studies in Science and Technology

Dissertations. No. 1746

Analysis, Design, and Optimization of Embedded Control Systems

by

Amir Aminifar

Department of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden

Linköping 2016

Copyright © 2016 Amir Aminifar

ISBN 978-91-7685-826-4

ISSN 0345-7524

Printed by LiU Tryck 2016

URL: <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-124319>

To my family

ABSTRACT

TODAY, many embedded or cyber-physical systems, e.g., in the automotive domain, comprise several control applications, sharing the same platform. It is well known that such resource sharing leads to complex temporal behaviors that degrades the quality of control, and more importantly, may even jeopardize stability in the worst case, if not properly taken into account.

In this thesis, we consider embedded control or cyber-physical systems, where several control applications share the same processing unit. The focus is on the control-scheduling co-design problem, where the controller and scheduling parameters are jointly optimized. The fundamental difference between control applications and traditional embedded applications motivates the need for novel methodologies for the design and optimization of embedded control systems. This thesis is one more step towards correct design and optimization of embedded control systems.

Offline and online methodologies for embedded control systems are covered in this thesis. The importance of considering both the expected control performance and stability is discussed and a control-scheduling co-design methodology is proposed to optimize control performance while guaranteeing stability. Orthogonal to this, bandwidth-efficient stabilizing control servers are proposed, which support compositionality, isolation, and resource-efficiency in design and co-design. Finally, we extend the scope of the proposed approach to non-periodic control schemes and address the challenges in sharing the platform with self-triggered controllers. In addition to offline methodologies,

a novel online scheduling policy to stabilize control applications is proposed.

The research presented in this thesis has been partially funded by CUGS (the National Graduate School in Computer Science in Sweden), Ericsson's Research Foundation, eLLIIT (Excellence Center at Linköping–Lund on Information Technology), and the Swedish Research Council.

POPULÄRVETENSKAPLIG SAMMANFATTNING

I DAG omfattar många inbyggda system flera styrapplikationer som delar samma plattform. Dessa system används inom exempelvis bilindustrin, processindustrin och flygelektronik, för att nämna några. Det är välkänt att en sådan resursdelning leder till komplexa tidsbeteenden som försämrar kvaliteten på regleringen och i värsta fall kan även stabiliteten förloras.

I denna avhandling betraktar vi inbyggda styrsystem, där flera styrapplikationer delar samma behandlingsenhet. Här har vi valt att fokusera på reglering-schemaläggning co-designproblemet. Till skillnad från traditionella inbyggda system där regleringsparametrarna och schemalägningsparametrarna optimeras separat, behöver vi nu för inbyggda styrsystem optimera dessa samtidigt, vilket ger upphov till nya metoder. Denna avhandling är ytterligare ett steg mot effektiv design och optimering av inbyggda styrsystem.

ACKNOWLEDGEMENTS

Needless to say, there is more to an academic life than just a thesis. Nonetheless, this thesis would not be possible, if it were not for certain people and I would like to take the opportunity to acknowledge them here.

First and foremost, I would like to acknowledge my advisors, Petru Eles and Zebo Peng, for being open to new ideas and their support and encouragement. Zebo was always very patient and efficient with all kinds of support. Petru is certainly one of the most influential people on my world view. I would also like to thank Alireza Ejlali at Sharif University of Technology for introducing me to the embedded systems area and Petru.

I would like to acknowledge Anton Cervin from the Automatic Control Department of Lund University for his support in the control-related part of my thesis. It was indeed a pleasure to discuss and collaborate with Anton and other members of the Automatic Control Department of Lund University.

I spent the last summer in Italy and my gratitude goes to Enrico Bini for hosting my visit at Sant'Anna School of Advanced Studies and our collaboration. Enrico and all members of the Real-Time Systems Laboratory were exceptionally hospitable.

The ideas of the schedulability analysis of the self-triggered controllers were developed when I was visiting Paulo Tabuada, and I would like to thank Paulo for hosting my visit at the Electrical Engineering Department of the University of California, Los Angeles, USA.

I would like to thank Sudipta Chattopadhyay, Ahmed Rezine, and Ke Jiang for always having time to discuss all kinds of scientific and non-scientific topics. Soheil Samii and Unmesh Bordoloi helped me start in this area when I joined the Embedded Systems Group. I would like to thank them for all the help and our collaboration. I would also like to thank Rohollah Mahfuzi for our discussions and collaboration.

I met a lot of fantastic people during all these years, in Linköping University, Sharif University of Technology, University of California Los Angeles, and Sant'Anna School of Advanced Studies. In particular, I would like to use this opportunity to thank Amin Ojani, Meysam Aghighi and, his wife, Zeinab Ganjei, Adrian Lifa, and Farrokh Ghani Zadegan, for all the support and all the great memories we share.

The working environment in IDA was nice and I would like to thank all the staff who made this possible. Adrian Lifa and Sarah Alsaadi helped me with writing the pop-smf and I would like to thank them.

Finally, my gratitude goes to my family for their unconditional love and support. I will forever be in your debt.

AMIR AMINIFAR
Sweden, 2016

CONTENTS

1	Introduction	1
1.1	Introduction and Background	1
1.2	State of the Art	5
1.3	Summary of Contributions	11
1.3.1	Control-Quality Driven Design with Stability Guarantees	12
1.3.2	Optimal Stabilizing Control Servers	13
1.3.3	An Online Stabilizing Scheduling Policy	13
1.3.4	Self-Triggered Controllers and Hard Real-Time Guarantees	14
1.4	List of Publications	14
1.5	Thesis Overview	16
2	Background and Preliminaries	19
2.1	System Model	19
2.1.1	Plant Model	19
2.1.2	Platform Model	20
2.1.3	Application Model	20
2.2	Control Performance and Controller Synthesis	22
2.2.1	Controller Synthesis	22
2.2.2	Expected Control Performance	24
2.2.3	Worst-Case Control Performance	29
2.3	Latency and Jitter Analyses	32
2.4	Response-Time Analysis	35

2.4.1	Worst-Case Response-Time Analysis	36
2.4.2	Best-Case Response-Time Analysis	36
2.5	Properties of Latency and Jitter	37
2.5.1	Analysis with Respect to Priorities	37
2.5.2	Analysis with Respect to Periods	40
2.6	General Problem Formulation	42
2.6.1	The Effect of Delay and Sampling Period	42
2.6.2	Task Scheduling and Controller Synthesis	45
2.6.3	Delay Distribution, Latency, and Jitter	46
3	Control-Quality Driven Design with Stability Guarantees	47
3.1	Introduction and Related Work	47
3.2	System Model	50
3.3	Control Performance and Synthesis	51
3.3.1	Expected Control Performance	51
3.3.2	Worst-Case Control Performance	51
3.3.3	Control Synthesis	52
3.4	Delay and Jitter Analyses	52
3.5	Motivational Example	54
3.5.1	Example 1: System Design Driven by Expected Control Quality	54
3.5.2	Example 2: Stability-Aware System Design with Expected Control Quality Optimization	56
3.6	Problem Formulation	57
3.7	Co-design Approach	58
3.7.1	Period Optimization	60
3.7.2	Priority Optimization and Control Synthesis	62
3.7.2.1	Worst-Case Sensitivity and Sensitivity Groups	63
3.7.2.2	Sensitivity-Based Application Clustering	63
3.7.2.3	Inside Group Optimization	64
3.8	Experimental Results	66
3.8.1	Efficiency of Our Proposed Approach	66
3.8.2	Runtime of Our Proposed Approach	68
3.9	Conclusions	69

4	Optimal Design of Stabilizing Control Servers	70
4.1	Introduction and Related Work	71
4.2	System Model	74
4.3	Server Model	74
4.4	Server-Based Analysis of Control Tasks	76
4.4.1	Exact Characterization	77
4.4.2	Characterization with Linear Bounds	78
4.5	Stability Constraint	78
4.6	Optimal Design of Stabilizing Servers	80
4.7	Theoretical Guarantees	85
4.7.1	Stability of Controllers	86
4.7.2	Schedulability of Servers	91
4.8	Asymptotic Analysis	94
4.9	Evaluation	95
4.9.1	Illustrative Example	95
4.9.2	Experimental Results	97
4.10	Conclusions	98
5	A Scheduling Policy to Stabilize Control Applications	100
5.1	Introduction and Related Work	100
5.2	System Model and Background	103
5.3	Motivational Example	105
5.4	Jfair Scheduling Policy	108
5.4.1	Scheduling Policy	108
5.4.2	Theoretical Guarantees	109
5.4.2.1	Non-Violation of Upper Lag Limit	109
5.4.2.2	Non-Violation of Lower Lag Limit	111
5.4.2.3	Scheduling Properties	111
5.5	Stability, Analysis, and Design	114
5.5.1	Control Stability and Jfair	114
5.5.2	Design for Stability and Jfair	115
5.6	Experimental Results	118
5.7	Conclusions	119
6	Self-Triggered Controllers and Hard Real-Time Guar-	
	antees	120
6.1	Introduction and Related Work	120
6.2	System Model	122
6.2.1	Task Model	122

6.2.2	Plant Model	123
6.2.3	Self-Triggered Controller	123
6.3	Problem Formulation	123
6.4	The Self-Triggered Controller	124
6.4.1	Offline Step	124
6.4.2	Online Step	124
6.5	The Big Picture	125
6.6	Finding Request-Bound Function	126
6.6.1	Extraction of the Transition Graph	126
6.6.1.1	Partitioning the State Space	127
6.6.1.2	Calculation of the Maximum Time h for Each Polytope	127
6.6.1.3	Construction of the Transition Graph	129
6.6.2	Extraction of the Worst-Case Request Pattern	129
6.7	Schedulability Analysis	130
6.8	Illustrative Example	132
6.9	Conclusions	135
7	Conclusions and Future Work	136
7.1	Conclusions	136
7.2	Future Research Directions	137

LIST OF FIGURES

1.1	Control applications sharing a processing unit.	2
1.2	Overview.	18
2.1	Modeling a controller as a task graph	22
2.2	Control cost, delay, and sampling period	26
2.3	Control cost, sampling period, and constant delay . . .	27
2.4	Control cost, delay, and fixed sampling period	28
2.5	General control problem formulation	29
2.6	The stability curves generated by Jitter Margin	30
2.7	Loop transformation of the delayed system	31
2.8	Graphical interpretation of the nominal sensor–actuator delay, worst-case sensor jitter, and worst-case actuator jitter	33
2.9	Graphical interpretation of the latency and worst-case response-time jitter	34
2.10	Non-monotonicity of response-time jitter	38
2.11	Non-monotonicity of response-time jitter with respect to priorities	41
2.12	Periodic sampling and constant delay	43
2.13	Periodic sampling and varying delay	44
3.1	Graphical interpretation of the nominal sensor–actuator delay, worst-case sensor jitter, and worst-case actuator jitter	53
3.2	Motivational examples	55

3.3	Overall flow of our approach	59
3.4	The coordinate and direct search methods	62
3.5	Experimental results	68
4.1	Overview of the proposed approach	72
4.2	Graphical interpretation of the latency and jitter . . .	74
4.3	Worst-case and best-case resource allocation scenarios	75
4.4	Linear, optimistic, and exact supply functions	86
4.5	Worst-case and best-case resource allocation scenarios for implicit deadline server	88
4.6	Experimental results	99
5.1	Comparing Pfair, server, and Jfair for a simple example	106
5.2	Deadline for Jfair	110
5.3	Comparison between Jfair and server approaches . . .	118
6.1	State space partitioning and the corresponding transi- tion graph	132
6.2	The worst-case triggering pattern of the self-triggered controller	133
6.3	The worst-case response time scenario	135

LIST OF TABLES

3.1	Experimental Results	66
4.1	Example: taskset data	96
4.2	Example: Solution to the server design problem. . . .	97
5.1	Summary of the characteristics of the schedule.	108
6.1	Example: taskset data	131

THIS chapter aims at providing a broad introduction to the topics covered in this thesis and highlighting the contributions of the thesis.

1.1 Introduction and Background

Today, many embedded or cyber-physical systems comprise several control applications. The majority of control applications are implemented as software tasks on microprocessors. These applications are in charge of controlling the physical plants associated with them.

Often, however, the processing unit (or alternatively the communication infrastructure) is shared among several applications. This scenario is shown in Figure 1.1. In this figure, there are two physical plants, inverted pendulums, with their controllers running on a shared processing unit. There might also exist other applications running on the same platform.

The interconnection of the physical plants to the cyber (processing) elements introduces the notion of physical time in today's embedded systems. Special care is needed for the implementation of such applications to ensure high performance and guarantee safety.

The design of embedded control systems involves two main steps: synthesis of the controllers and implementation of the control applications on a given execution platform. The controller synthesis step comprises period assignment, delay compensation, and control-law synthesis. The implementation step, on the other hand, is mostly

Figure 1.1: Control applications sharing a processing unit.

concerned with allocating computational resources to control applications (e.g., mapping and scheduling). This thesis focuses on the *control-scheduling* problem, and its extension to *control-scheduling co-design*.

The goal of the control-scheduling problem is to assign processing resources to the already synthesized control tasks such that the stability of the plants is guaranteed and the overall control performance of the system is optimized. The process of assigning processing resources to tasks is traditionally known as task scheduling.

One further step is the control-scheduling co-design problem, where the objective is to synthesize the controllers and schedule them in such a way that the control applications are guaranteed to remain stable, while providing as high performance as possible. In other words, the control-scheduling co-design problem addresses the joint optimization of the control parameters and scheduling parameters. Note that the interdependency between the controller synthesis and scheduling makes the problem all the more challenging. That is, changing the controller parameters will affect the scheduling parameters that may be used to guarantee stability and high performance and, respectively, the scheduling parameters affect the controller synthesis process.

Let us focus on an inverted pendulum, where we would like to change the cart position, while keeping the pendulum in the upright position. Ideally, at each moment, a dedicated continuous-time controller reads the angular position and velocity of the inverted pendu-

lum and the position and velocity of the cart and according to the sensed data, applies the optimal control signal (also known as control input), in terms of force. However, it is well known that to achieve high performance and acceptable stability margins as well as efficient resource-usage and flexibility, often, it is only required to control the plant frequently enough. The alternative is a discrete-time control task, e.g., executing periodically, which can also be implemented in software on a microcontroller.

Considering a continuous-time plant and a discrete-time control task, the following process is done periodically: (1) The position and velocity of the pendulum and cart are sampled. (2) Based on this sampling information the appropriate control input is calculated by a software task on the processing unit. (3) Finally, the control signal is applied in terms of force to the cart in the actuation phase.

Often, the controller is synthesized for a given sampling period and the time-delays experienced by the corresponding control task, e.g., the delay from the sampling instant to the actuation instant. Any runtime deviation from the assumptions on the sampling period or delays during the controller synthesis deteriorates the control performance.

Having the platform shared among several tasks, as in the example, the delay between sampling and actuation not only will be longer than on a dedicated platform, but also will be varying for different jobs of a control task. This is due to the fact that tasks compete for execution on the shared platform. (The situation only gets more complex if we take into consideration the variations in the computation times of the tasks due to the different input data and different states of the platform, e.g., cache and pipeline.) Therefore, as a result of sharing the platform, the control task may experience considerable amount of *latency* (the constant part of the delay) and *jitter* (the varying part of the delay), which affect the control performance and stability of the control application. Today, the literature does provide some results that account for the effect of the controller task schedule on the system performance and stability. For example, the effect of the latency from the sensing to the actuation [ÅW97], the effect of the jitter in the task completion [Cer12], and even that of the sensor-actuator delay distribution [CHL⁺03] on the control performance and stability are well understood.

To pin down the intricate relation between scheduling and control synthesis problems, let us take a closer look at this interdependencies. The controllers are often designed for a certain sampling period and delay characteristics. Then, the sampling period of the control application implies the scheduling of the control tasks. The scheduling has a direct impact on the delay experienced by the control task. The delay characteristic, as discussed before, is an important factor in controller synthesis and also affects the sampling periods which may be used for the control tasks to guarantee high performance and stability. In principle, this intricate mutual relation between controller synthesis and task scheduling advocates the need for a control-scheduling co-design procedure.

More concretely, on one hand, the controller synthesis determines the scheduling parameters that may be assigned to the control tasks in order to ensure the imposed performance requirements. On the other hand, the actual schedule leads to timing properties that need to be taken into consideration for controller synthesis to ensure performance requirements.

To sum up, sharing of the execution platform by several applications leads to complex timing behaviors. Ignoring such complex timing behaviors during the design process leads to poor control performance, and may even jeopardize the stability of control applications.

This thesis addresses several challenges in the design and optimization of embedded control systems comprising several control applications running on shared platforms. Observe that the problem has its roots in resource sharing and, therefore, most of the results in this thesis are also valid even in the case where a shared communication infrastructure or distributed platform is considered.

This class of embedded systems is traditionally known as embedded control systems, which lies at the heart of the cyber-physical systems concept. This thesis is one more step towards developing the theoretical foundation for the correct implementation of embedded control systems.

1.2 State of the Art

In this chapter, we present a generic outline of the state of the art, based on which we can discuss the contributions of the thesis. More focused discussion of the previous work will be given in the individual chapters.

The implementation of control applications on shared platforms consists of two main steps: controller synthesis and control task scheduling. The traditional design flow of such applications is based on the principle of *separation of concerns*. The main drawbacks of this principle are poor resource utilization and control performance [ÅC05]. This is due to the fact that the effects of the decisions made during one of the design steps are not considered on the decisions that can be made during the second step. This often leads to a suboptimal solution.

The timing problems in real-time control systems were first brought up by Wittenmark et al. [WNT95]. They discuss the issue of time-varying delays introduced during the implementation phase of such systems.

The interaction between control task performance and task scheduling is investigated by Seto et al. [SLSS96]. They find the optimal sampling periods for a set of controllers implemented on a uniprocessor platform with respect to a given cost function. They, however, do not consider the effect of delay and jitter experienced by each control task as a result of task scheduling.

In [ÅCES00], the authors discuss the need for control-scheduling co-design methodologies, allowing trade-offs between control performance and resource utilization.

Rehbinder and Sanfridson [RS00] propose an approach for integration of offline scheduling and optimal control. They consider the static-cyclic scheduling policy and account for deterministic jitter in the optimal control design by formulating it as a periodic control problem. However, they point out that the proposed approach is in general intractable and in practice it only works for a small number control applications and limited number of permutations, due to its combinatorial nature.

Ben Gaid et al. [BCH06] formulate the control-scheduling problem over a communication channel as a single problem, assuming static scheduling. The problem is then transformed into a mixed

integer quadratic programming formulation, which essentially has exponential time complexity.

The Jitterbug toolbox [LC02] computes the expected control performance for a linear control system under various timing conditions, e.g., in the presence of delay, jitter, or even lost samples. Therefore, using this toolbox it is possible to study the effects of delay, jitter, lost samples, etc. on control performance [CHL⁺03]. However, in many cases, the underlying assumptions on independence of time delays renders it impossible to provide hard guarantees regarding, for example, stability.

In [CLE⁺04], Cervin et al. introduce the notion of *jitter margin* and propose an iterative control–scheduling co-design procedure, based on the worst-case control performance and stability. The jitter margin is the maximum amount of jitter a control application can experience and still remain stable. In [BI07], similar to [CLE⁺04], the authors propose an integrated approach for control design and real-time scheduling, based on the jitter margin performance metric.

In [BC08], Bini and Cervin extend the previous results from [SLSS96], by incorporating delays into the objective function. They consider an objective function which is linear in the sampling periods and expected delays experienced by each control application and find the optimal sampling periods (with respect to the linear cost function). However, in general, the control cost may be a nonlinear function. Therefore, an iterative optimization approach is used to further improve the solution obtained.

Naghshtabrizi and Hespanha [NH08] discuss the importance of considering the overall system performance, while providing certain correctness guarantees for distributed control systems with shared communication and computation resources.

Zhang et al. [ZSWM08] consider the control–scheduling co-design problem, and find the optimal sampling periods of a set of controllers, under certain assumptions, which minimize a worst-case control cost.

Samii et al. [SCEP09] propose an integrated task scheduling and control synthesis approach to optimize the overall control performance. They consider both static-cyclic and priority-based scheduling policies on distributed platforms. This work is then extended to consider embedded control systems which switch between different functional modes [SEPC09] and to the case of fault-tolerant control

systems [SBE⁺12].

Goswami et al. [GLSC12] propose an Integer Linear Programming (ILP) formulation for time-triggered implementation of control and time-critical applications in the automotive domain. Since they do not consider the effect of jitter on the stability of control applications, either the periods should be in harmonic relations, or the worst-case delay should be considered. Either way, this leads to poor control performance. In the former, the delays will be short, but the periods can only increase exponentially, which leads to performance degradation. In the latter, the periods are not restricted, but the output of the control task should be buffered for the worst-case delay to avoid jitter, which, again, leads to performance degradation.

In [GSC11a], the authors consider a more flexible delay constraint model compared to [GLSC12], where not all the control task executions meet their deadlines. In this way, the authors avoid designing the controller for the worst-case sensor-actuator delay. While in such an approach some of the control computations are ignored, the design is based on a range of delay values that are most likely to occur.

In [GSC11b], the authors discuss a hybrid protocol where both the communication schedule and the control input change based on the state of the system. The goal is to achieve a trade-off between control performance and communication bandwidth utilization, considering the FlexRay protocol [Con05]. The idea is to use the time-triggered communication slots during the transient phase, which provides higher predictability, whereas the event-triggered scheme is used once the plant is in the steady state, which is advantageous from the bandwidth utilization point of view. This is based on the observation that delays during the transient phase cause more performance deterioration, compared to the steady state. The authors also point out that the approach is only effective if the switching between the two modes is not very frequent. Masrur et al. [MGC⁺12] discuss whether it is possible to switch from the event-triggered mode to the time-triggered mode in case of an external disturbance, within a specified deadline. To obtain a realistic disturbance model, however, is not a trivial task.

In [KGC⁺12], Kumar et al. introduce the *delay-frequency* interface, by which they bound the frequency of the worst-case delay. This will allow verifying tighter control performance properties, since, of-

ten, the worst-case delay occurs very rarely.

Similar to [KGC⁺12], in [BS13], the authors discuss the *delay density* model for network control systems. The model is richer than the previously discussed interfaces, as it captures the correlation among the consecutive samples. This is simply done by considering the cumulative delay in an interval of certain length. While considering this model probably leads to a better resource usage, it is not suitable in providing hard guarantees, due to its stochastic nature. Even though intuitively this model is more accurate, the systematic incorporation of the model in performance and stability analysis is yet to be done. A similar idea was discussed in [BS15], but this time with respect to the density of dropped samples. This model captures the number of dropped samples among an arbitrary number of consecutive samples.

Naghshtabrizi and Hespanha [NH09] consider the analysis problem of distributed control systems with shared communication and computation resources. While the control stability results are developed for variable, but bounded, delay and sampling separation interval, the schedulability analysis considered is restricted to a set of periodic tasks.

In [BPZ02, MSZ11, SBM15], the authors discuss scheduling of control applications in the presence of packet loss. In fact, the authors exploit packet loss to guarantee asymptotic stability of a set of control applications on a shared platform, which was originally overloaded.

In [AEPC13b], it is pointed out that control applications often do not enforce hard deadlines. The authors also discuss several cases of anomalies for control applications. It is shown that increasing the priority level of a control task or higher sampling rates might lead to higher values of jitter and, in turn, instability.

Xu et al., in [XÅBC14], discuss the importance of considering the time-varying delays experienced by each control application implemented on a shared platform, during the design phase. In [XÅC⁺15], the authors focus on the static-cyclic scheduling policy and propose a control-scheduling co-design approach in this setting. Exploiting the periodic delay patterns inherent in the periodic task execution paradigm, they present an analytical procedure to design periodic Linear-Quadratic-Gaussian (LQG) controllers.

In [MBP11, MBP14], the authors propose frameworks for control-quality driven priority and period assignment for control tasks, assum-

ing a fluid execution model and constant delay. Similarly, a stability-aware priority assignment algorithm is discussed in [AEPC13b].

Buttazzo et al. [BVMF04] discuss a method for managing control performance under overload conditions. The proposed approach is based on the elastic scheduling theory [BLCA02] and the idea of switching the controller if it cannot provide the required performance at the current rate.

Palopoli et al. [PPBSV05] propose an approach to find the stability radius for control applications, considering a time-triggered model of computation and by translating the stability into deadline. The sensitivity analysis of embedded control applications with respect to variations in the sampling period is discussed in [AEPC13b].

In [NPAG06], the authors show that the performance gap between the Proportional-Integral-Derivative (PID) controller designed based on the mathematical model and the implementation of the controller on a predictable time-triggered platform can be quantified.

Weiss and Alur [WA07] consider automata theory to find the language that does not violate the performance requirements of a system. Then, a non-empty intersection of the scheduling automaton and performance requirements automata indicates the schedulability of the system. Similarly, in [DKGT14], the authors check whether the language (in terms of deadline miss/hit) of the provided resource by the real-time platform is included in the language (in terms of deadline miss/hit) which meets the performance constraints of a control application.

In [KKH⁺08], Koutsoukos et al. propose a passivity-based control design for cyber-physical systems. The main idea is that, by imposing passivity constraints on the component dynamics, the network effects can be ignored, hence the separation of concerns between the control design and implementation. Thus, the main advantage of the proposed approach is in facilitating compositional component-based design of cyber-physical systems. However, the price to pay for this separation is worse performance compared to the case where the control design and implementation are done in an integrated manner.

The analysis and design of servers for real-time applications are discussed by [SRLK02, LB03, SL03, AP04, EAL07, FD12]. In [CE05], Cervin and Eker introduce the concept of *control server*, as a simple interface between the controller synthesis and task scheduling.

In [FPG13], the authors consider the problem of bandwidth allocation for a set of control tasks, assuming a fluid approximation for execution model. While a time-triggered approach is considered, to avoid poor performance, they ignore the sensor data if the delay is longer than one sampling period. In [FGP13], they formulate a similar problem into a mixed integer optimization, hence combinatorial time complexity.

In [FPA13], the authors propose another execution model for real-time control applications to investigate stochastic stability, but ignoring the dependencies among stochastic variables, hence no hard stability guarantees.

The theory underlying the non-periodic (event-triggered and self-triggered) control paradigm is now well developed, assuming dedicated execution platforms [ÅB99, Årz99, VFM03, HJC08, HSB08, MAT09, WL09, VMB15]. However, the efficient and safe implementation of such controllers on shared platforms is still an open problem, despite several attempts [LCHZ07, VMB08, CH08, AT09, SEP⁺10, BMS13, AH14]. Current practice either suffers from extreme pessimism or provides no hard guarantees.

Cervin and Henningsson [CH08] discuss the suitability of different medium access schemes for implementation of time-triggered and event-triggered control paradigms.

Lemmon et al. [LCHZ07] consider online scheduling of self-triggered controllers using elastic scheduling, but no stability guarantees are provided. Samii et al. [SEP⁺10] discuss dynamic scheduling of self-triggered controllers, with an objective function which considers both control quality and resource usage. However, the existence of such a dynamic schedule is only guaranteed under a very restrictive condition.

Velasco et al. [VMB08] discuss the challenging task of offline schedulability analysis when the platform is shared with self-triggered controllers. However, the main problem of finding the worst-case triggering pattern is left open. Anta and Tabuada [AT09] discuss the benefits of relaxing periodicity constraints over communication networks. However, to provide schedulability guarantees, the authors consider the minimum inter-arrival time for all possible initial states, which is extremely pessimistic and defeats the purpose of the self-triggered control paradigm. In [BMS13], the authors discuss an approximate

maximum inter-event time for event-triggered controllers, justified by the fact that a more regular execution pattern of event-triggered controllers leads to better schedulability results for other tasks.

In [EHÅ00, CEBÅ02], the authors discuss feedback–feedforward scheduling of control tasks. The scheduler uses feedback from execution-time measurements and feedforward from workload changes to assign the sampling periods of the control tasks at runtime. In [HC05, CVMC11], the authors propose a feedback scheduling strategy to assign the sampling period based on the plant state information. Similarly, in [MLB⁺04], the authors discuss that the open-loop resource management policy may not be optimal for resource constrained systems. For instance, a control task may not require the allocated bandwidth if the corresponding controlled system is already in equilibrium.

Related to non-periodic task models, the rate-adaptive task model is introduced by Buttazzo et al. in [BBB14]. The task model is considered in the context of schedulability analysis for engine control task. The idea is that the higher the angular velocity of the engine is, the higher the rate at which the related tasks need to execute (but with shorter execution times). However, a control-oriented metric which relates the task execution rate to the performance of the engine, instead of hard deadlines, is still needed.

1.3 Summary of Contributions

In this thesis, we propose systematic methodologies for the design and optimization of embedded control systems. First, the importance of considering both the expected control performance and stability is discussed and a design methodology is proposed to optimize control performance while guaranteeing stability [ASE⁺12]. Orthogonal to this design methodology, the design of bandwidth-efficient stabilizing control servers is addressed [ABEP15], which facilitates the compositional, isolated, and resource-efficient design of embedded control systems. Finally, the methodologies discussed in this thesis go beyond the traditional periodic controllers, since the thesis also addresses the analysis and design of such systems in the presence of novel self-triggered control schemes [ATEP16]. In addition to offline design methodologies, a novel scheduling policy, *Jfair*, to stabilize control

applications is proposed [AEP15]. The contributions of this thesis are summarized in the following subsections.

1.3.1 Control-Quality Driven Design with Stability Guarantees

In Chapter 3, two kinds of metrics to capture control performance are discussed: (1) stochastic control performance metrics and (2) robustness (stability-related) metrics. The former identifies the expected (*mathematical expectation*) control performance of a control application, whereas the latter is considered to be a measure of the worst-case control performance. Although considering both the expected control performance and worst-case control performance during the design process is crucial, previous work only focuses on one of the two aspects. The main drawback of such approaches, e.g., based solely on the expected control performance, is that the resulting high (expected) performance design solution does not necessarily satisfy the stability requirements in the worst-case scenario. On the other hand, considering merely the worst case, often results in a system with poor expected control performance. This is due to the fact that the design is solely tuned to a scenario that occurs very rarely. Thus, even though the overall design optimization goal should be the expected control performance, taking the worst-case control stability into consideration during the design space exploration is indispensable.

In Chapter 3, we propose an integrated control-scheduling co-design approach to optimize the expected control performance, while guaranteeing stability and robustness, even in the worst-case scenario.

Previously, the literature considers either the stability and worst-case control performance or the expected control performance, except for the simple case of time-triggered and static-cyclic scheduling policies [RS00, GLSC12]. In these approaches, eliminating the element of jitter or focusing on fixed delay patterns makes the analysis straightforward. Our proposed approach, however, goes beyond such specific and restrictive policies.

It has been shown that the results in this chapter are also valid in the case of distributed embedded control systems [AEPC13a].

1.3.2 Optimal Stabilizing Control Servers

In Chapter 4, the design of embedded control systems considering a server-based resource reservation mechanism is addressed. Note that the design issues discussed in this chapter are orthogonal to the design issues discussed in Chapter 3. The benefits of employing a server-based approach are manifold: providing a compositional and scalable framework, protection against other tasks' misbehaviors, and systematic control server design and controller-server co-design. We propose a methodology for designing bandwidth-optimal servers to stabilize control tasks. The pessimism involved in the proposed methodology is both discussed theoretically and evaluated experimentally. It has been proved that if it is possible to implement the system over a unit-speed processor, it is also possible to implement the system using our design methodology, but considering a processing unit which runs at most two times faster.

The control server concept was first introduced in [CE05]. However, since then, the problem of designing such servers in a systematic manner was the main obstacle in using these servers. Current practice either considers simplified fluid execution models [FPG13] or ignores the dependencies among stochastic variables, which leads to lack of hard stability guarantees [FPA13]. In Chapter 4, we not only address the problem of optimal control server design to guarantee stability with the minimum bandwidth usage, but also provide analytical bounds on the amount of pessimism involved in our proposed approach.

1.3.3 An Online Stabilizing Scheduling Policy

In Chapter 5, we propose an online scheduling policy to stabilize control applications. We prove that it is possible to guarantee any jitter limits for tasks as long as the total processor utilization is not exceeding 100%. It is also proved that the number of preemptions needed by the proposed scheduling policy will be at most three times that of any other valid scheduling policy, including the optimal one. We discuss how to select the scheduling parameters, given the constraints on the latency and jitter, to guarantee stability. Further, a design optimization problem is formulated to minimize the total amount of resources utilized to guarantee the stability of the control applications.

1.3.4 Self-Triggered Controllers and Hard Real-Time Guarantees

In Chapter 6, as opposed to the previous chapters, we focus on the novel self-triggered controller paradigm. Self-triggered controllers, as opposed to the traditional periodic controllers, execute only when it is required to guarantee their performance metric. It is well known that event-triggered and self-triggered controllers implemented on dedicated platforms can provide the same performance as the traditional periodic controllers, while consuming considerably less bandwidth. However, since the majority of controllers are implemented by software tasks on shared platforms, it might no longer be possible to grant access to the event-triggered controllers upon request. On the other hand, due to the seemingly irregular requests from self-triggered controllers, other applications sharing the same processing element, while in reality schedulable, may be declared unschedulable, if not carefully analyzed. The schedulability and response-time analysis in the presence of self-triggered controllers is still an open problem and the topic of Chapter 6.

Existing approaches to the schedulability analysis in the presence of self-triggered controllers are either extremely pessimistic [AT09, SEP⁺10, BMS13] or do not provide any hard guarantees [LCHZ07, VMB08, CH08, AH14]. In Chapter 6, we address this problem and propose an offline schedulability analysis technique in the presence of self-triggered control tasks that allows to leverage the potential advantages of self-triggered control compared to periodic control.

1.4 List of Publications

The following published articles are directly related to the contributions presented in this thesis:

- Amir Aminifar, Paulo Tabuada, Petru Eles, Zebo Peng. "Self-Triggered Controllers and Hard Real-Time Guarantees", Design, Automation and Test in Europe (DATE), Germany, 2016 [ATEP16].
- Amir Aminifar, Enrico Bini, Petru Eles, Zebo Peng. "Analysis and Design of Real-Time Servers for Control Applications", IEEE Transactions on Computers (TC), 2015 [ABEP15].

- Amir Aminifar, Petru Eles, Zebo Peng. "Jfair: A Scheduling Algorithm to Stabilize Control Applications", Real-Time and Embedded Technology and Applications Symposium (RTAS), USA, 2015 [AEP15].
- Amir Aminifar, Enrico Bini, Petru Eles, Zebo Peng. "Designing Bandwidth-Efficient Stabilizing Control Servers", Real-Time Systems Symposium (RTSS), Canada, 2013 [ABEP13].
- Amir Aminifar, Petru Eles, Zebo Peng, Anton Cervin. "Stability-Aware Analysis and Design of Embedded Control Systems", International Conference on Embedded Software (EMSOFT), Canada, 2013 [AEPC13b].
- Amir Aminifar, Soheil Samii, Petru Eles, Zebo Peng, Anton Cervin. "Designing High-Quality Embedded Control Systems with Guaranteed Stability", Real-Time Systems Symposium (RTSS), Puerto Rico, 2012 [ASE⁺12].

The following publications are related to the topic of this research, but are not directly covered in this thesis:

- Amir Aminifar, Enrico Bini, Petru Eles, Zebo Peng. "Bandwidth-Efficient Controller-Server Co-Design with Stability Guarantees", Design, Automation and Test in Europe (DATE), Germany, 2014 [ABEP14].
- Unmesh D. Bordoloi, Amir Aminifar, Petru Eles and Zebo Peng. "Schedulability Analysis of Ethernet AVB Switches", International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), China, 2014 [BAEP14].
- Amir Aminifar, Petru Eles, Zebo Peng, Anton Cervin. "Control-Quality Driven Design of Cyber-Physical Systems with Robustness Guarantees", Design, Automation and Test in Europe (DATE), France, 2013 [AEPC13a].
- Amir Aminifar, Soheil Samii, Petru Eles, Zebo Peng. "Control-Quality Driven Task Mapping for Distributed Embedded Control Systems", International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Japan, 2011 [ASEP11].

1.5 Thesis Overview

This thesis is organized in seven chapters. In Chapter 2, we will discuss the background of the work and the system model we consider. The control synthesis process and different control performance metrics are introduced. It will be shown that the stability of a control application depends on the real-time metrics, specifically latency and jitter, introduced through task scheduling. To compute the latency and jitter, the basics of real-time scheduling and response-time analysis will be discussed. Finally, the general control-scheduling problem is introduced. Note that, throughout this chapter, we try to provide intuition for the reader by discussing simple examples and simplified proofs.

In Chapter 3, we propose a design methodology to optimize the expected control performance, while guaranteeing robustness and stability in the worst-case scenario. The importance of such a design methodology is emphasized. On one hand, it has been shown that considering only the expected control performance leads to situations where we do not have any guarantees on the stability of the plants, in the worst-case scenario. On the other hand, the experiments demonstrate that considering only the worst-case control performance and optimization, based on the worst-case control performance, results in a design which is optimized to a scenario which occurs very rarely, and might also involve significant amount of pessimism. This, in turn, leads to poor control performance and resource utilization. Therefore, to provide high quality of control while guaranteeing stability, it is important to consider both the expected and worst-case control metrics.

Orthogonal to Chapter 3, in Chapter 4, we discuss a virtualization framework to facilitate the design problem. The resource-reservation mechanism provides compositionality, isolation, and a simple interface for a systematic design methodology. This simple interface enables us to find the minimum bandwidth required to guarantee the stability of control applications. The pessimism introduced is discussed theoretically and experimentally. In particular, it is proved that our proposed approach will, in the worst-case, require a processor which runs at double-speed, when compared to the optimal.

While in Chapters 3 and 4 we consider offline design methodologies, in Chapter 5, we propose an online scheduling algorithm, called

Jfair, to stabilize control applications. Roughly speaking, the algorithm is an extension of the *Pfair* algorithm [BCPV96]. It has been shown that the algorithm can guarantee any jitter limits as long as the processor utilization is not above 100%. Further, we guarantee that the number of preemptions in our proposed algorithm to guarantee certain lag limits (which can be translated into jitter limits) is at most three times that of the optimal scheduling policy. We further discuss an optimization approach to determine the lag limits required to guarantee the stability of a control application scheduled by Jfair.

Thus far in the thesis we considered the periodic control paradigm. In Chapter 6, however, we extend the scope of our discussion to the self-triggered control schemes. As opposed to the periodic control paradigm, the seemingly irregular execution pattern makes it challenging to provide any guarantees when the platform is shared with self-triggered controllers. In this chapter, we investigate the execution patterns of self-triggered controllers and provide hard guarantees in the presence of self-triggered controllers on shared platforms.

Finally, the thesis will be concluded in Chapter 7. Future research directions will also be discussed in this chapter.

Figure 1.2 shows an overview of the thesis as a whole, and how the individual contributions of this thesis can be put together. The figure shows several servers and control tasks sharing the same platform, which are scheduled by the earliest-deadline-first (EDF) scheduling policy. For simplicity, we do not show the plants for all control tasks.

The designer may decide to run several control tasks within a server. This scenario is shown in Figure 1.2, where there are three control tasks running within server 3. Let us assume that the three control tasks are scheduled based on the fixed-priority policy. In this scenario, the control-scheduling co-design approach presented in Chapter 3 can be used within the server to optimize control quality while guaranteeing stability, with respect to the controller and scheduling parameters.

The design methodology proposed in this thesis supports compositionality and isolation (see Chapter 4). As shown in the figure, there might be several servers running in parallel and hosting control (server 1) or other applications (server 4). The optimal stabilizing server design technique in Chapter 4 can be readily utilized to find the minimum bandwidth required to guarantee the stability of the

Figure 1.2: Overview.

control task within server 1.

Since the scheduling policy on the processing unit is the earliest-deadline-first (EDF) policy, the Jfair scheduling policy in Chapter 5 can be used to guarantee the stability of some control tasks, from which we only know very limited information.

Finally, for hard real-time tasks to be able to meet their deadlines when sharing the platform with a self-triggered controller (e.g., the real-time task within server 2), it is required to calculate the interference that the hard real-time task experiences from the self-triggered control task. This issue is discussed and addressed in Chapter 6.

BACKGROUND AND PRELIMINARIES

THE aim of this chapter is to introduce a common basis for the models considered in the following chapters. Further, this chapter provides helpful intuition regarding the mathematics used in this thesis, both in the control theory and real-time areas.

2.1 System Model

The system model is determined by the plant model, the platform model, and the application model. In this chapter we consider a general system model.

2.1.1 Plant Model

Let us consider a given set of plants $\mathbf{P}$. Each plant $\mathbf{P}_i$ is modeled by a continuous-time system of equations

$$\begin{aligned}\dot{\mathbf{x}}_i &= \mathbf{A}_i \mathbf{x}_i + \mathbf{B}_i \mathbf{u}_i + \mathbf{v}_i, \\ \mathbf{y}_i &= \mathbf{C}_i \mathbf{x}_i + \mathbf{e}_i,\end{aligned}\tag{2.1}$$

where $\mathbf{x}_i$ and $\mathbf{u}_i$ are the plant state and control signal, respectively. The additive plant disturbance $\mathbf{v}_i$ is a continuous-time white-noise process with zero mean and given covariance matrix $\mathbf{R}_{1i}$. The plant output is denoted by $\mathbf{y}_i$ and is sampled periodically with some delays at discrete time instants—the measurement noise $\mathbf{e}_i$ is a discrete-time Gaussian white-noise process with zero mean and covariance $\mathbf{R}_{2i}$. The control signal will be updated periodically with some delays

at discrete time instants and is held constant between two updates by a hold-circuit in the actuator [ÅW97].

For instance, an inverted pendulum can be modeled using Equation (2.1) with

$$\mathbf{A}_i = \begin{bmatrix} 0 & 1 \\ g/l_i & 0 \end{bmatrix}, \quad \mathbf{B}_i = \begin{bmatrix} 0 \\ g/l_i \end{bmatrix}, \quad \mathbf{C}_i = \begin{bmatrix} 1 \\ 0 \end{bmatrix}^\top, \quad (2.2)$$

where $g \approx 9.81 \text{ m/s}^2$ is the gravitational constant and l_i is the length of pendulum $\mathbf{P}_i$. The two states in $\mathbf{x}_i = \begin{bmatrix} \phi_i \\ \dot{\phi}_i \end{bmatrix}$ are the pendulum position ϕ_i and speed $\dot{\phi}_i$. For plant disturbance and measurement noise, we have $R_{1i} = \mathbf{B}_i \mathbf{B}_i^\top$ and $R_{2i} = 0.1$, respectively.

2.1.2 Platform Model

The platform considered in this thesis is a uniprocessor, even though the results are mostly valid also for communication infrastructures and distributed platforms. For instance, in [AEPC13a], we extend our co-design approach proposed in Chapter 3 to distributed cyber-physical systems and demonstrate that the conclusions are also valid even when a distributed platform is considered.

In Chapter 4, however, we will not consider any platform explicitly since the server concept is based on the virtualization idea.

2.1.3 Application Model

For each plant $\mathbf{P}_i \in \mathbf{P}$ there exists a corresponding control application denoted by $\Lambda_i \in \mathbf{\Lambda}$, where $\mathbf{\Lambda}$ indicates the set of applications in the system. Each application Λ_i is modeled as a task graph. A task graph consists of a set of tasks and a set of edges, identifying the dependencies among tasks. Thus, an application is modeled as an acyclic graph $\Lambda_i = (\mathbf{T}_i, \mathbf{\Gamma}_i)$, where $\mathbf{T}_i$ denotes the set of tasks and $\mathbf{\Gamma}_i \subset (\mathbf{T}_i \times \mathbf{T}_i)$ is the set of dependencies between tasks. We denote the j -th task of application Λ_i by τ_{ij} . The execution time, c_{ij} , of the task τ_{ij} is modeled as a stochastic variable with probability function ξ_{ij} , bounded by the best-case execution-time c_{ij}^b and the worst-case execution time c_{ij}^w . The probability function ξ_{ij} is used for the system simulation which is utilized in delay compensation for controller design and computing the expected control performance.

Worst-case stability guarantees only depend on the worst-case and best-case execution times. Further, the dependency between tasks τ_{ij} and τ_{ik} is captured by $(\tau_{ij}, \tau_{ik}) \in \Gamma_i$.

Control applications can typically provide satisfactory control performance over a range of sampling periods. One extensively used rule of thumb for identifying the interval from which the sampling period, denoted by h_i , can be chosen is as follows [ÅW97]:

$$0.2 \leq \omega_i h_i \leq 0.6, \quad (2.3)$$

where ω_i is the bandwidth of the closed-loop system for application Λ_i .

Hence, each application Λ_i can execute with a period $h_i \in \mathbf{H}_i$, where $\mathbf{H}_i$ is the set of suggested periods application Λ_i can be executed with. However, the actual period for each control application is determined during the co-design procedure, considering the direct relation between scheduling parameters and control synthesis.

A simple example of modeling a control application as a task graph is shown in Figure 2.1. The control application Λ_i has three tasks, where τ_{is} , τ_{ic} , and τ_{ia} indicate the sensor, computation, and actuator tasks, respectively. The arrows between tasks indicate the dependencies, meaning that, for instance, the computation task τ_{ic} can be executed only after the sensor task τ_{is} completes its execution.

A special case of a task graph is a single task, which is very often sufficient to support our claims throughout this thesis. Thus, occasionally, we consider only a single task τ_i with the following parameters, which describe the timing behavior of the task:

- the *execution time*, denoted by c_i with probability mass function ξ_i ;
- the *best-case execution time*, denoted by c_i^b ;
- the *worst-case execution time*, denoted by c_i^w ; and
- the *sampling period*, denoted by h_i .

The sampling frequency is denoted by f_i and is defined as follows,

$$f_i = \frac{1}{h_i}. \quad (2.4)$$

Figure 2.1: Example of modeling a controller as a task graph

2.2 Control Performance and Controller Synthesis

In this section, we present preliminaries related to controller synthesis and also introduce control performance metrics, both the expected and worst-case.

2.2.1 Controller Synthesis

For a given sampling period h_i and a given, constant sensor–actuator delay (i.e., the time between sampling the output $\mathbf{y}_i$ and updating the controlled input $\mathbf{u}_i$), it is possible to find the control-law $\mathbf{u}_i$ that minimizes the expected cost $J_{\Lambda_i}^e$ [ÅW97]. Thus, the optimal controller can be designed if the delay is constant at each execution instance of the control application. Since the overall performance of the system is determined by the expected control performance, the controllers shall be designed for the expected (average) behavior of the system. System simulation is performed to obtain the delay distribution and the expected sensor–actuator delay and the controllers are designed to compensate for this expected amount of delay. In order to synthesize the controller we use MATLAB and the Jitterbug toolbox [LC02].

The sensor–actuator delay is, in reality, not constant at runtime due to the interference from other applications competing for the shared resources. The quality provided by the synthesized controller is degraded if the sensor–actuator delay is different from the constant one assumed during the control-law synthesis. The overall expected control quality of the controller for a given delay distribution is obtained as it will be discussed in the next section.

To provide intuition regarding the controller synthesis procedure, in the following, we discuss the Linear-Quadratic state-feedback controller (LQR), for the simple case of continuous-time plant and controller, ignoring delay, jitter, measurement noise, and disturbance.

We consider the simplified dynamical system

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}.$$

The problem formulation is as follows,

$$\begin{aligned} \min_{\mathbf{u}} \quad & J_{\Lambda}^e(\mathbf{x}_0) = \int_0^{\infty} \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u} \, dt \\ \text{s.t.} \quad & \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \\ & \mathbf{x}(0) = \mathbf{x}_0. \end{aligned}$$

The optimal, stabilizing, feedback control is given by,

$$\mathbf{u} = -\mathbf{R}^{-1} \mathbf{B}^T \mathbf{P} \mathbf{x},$$

where $\mathbf{P}$ is the solution to the following Riccati equation,

$$\mathbf{A}^T \mathbf{P} + \mathbf{P} \mathbf{A} + \mathbf{Q} - \mathbf{P} \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^T \mathbf{P} = 0.$$

It is well known that for linear systems there exists a Lyapunov function $V(t) = \mathbf{x}^T(t) \mathbf{P} \mathbf{x}(t)$, where $\mathbf{P}$ is a symmetric positive definite matrix that satisfies the algebraic Riccati equation. Since we want to have decrease along the trajectory, we would like to have $\dot{V}(t) < 0$,

$$\begin{aligned} \dot{V}(t) &= \dot{\mathbf{x}}^T \mathbf{P} \mathbf{x} + \mathbf{x}^T \mathbf{P} \dot{\mathbf{x}} \\ &= \mathbf{x}^T \mathbf{A}^T \mathbf{P} \mathbf{x} + \mathbf{x}^T \mathbf{P} \mathbf{A} \mathbf{x} + \mathbf{u}^T \mathbf{B}^T \mathbf{P} \mathbf{x} + \mathbf{x}^T \mathbf{P} \mathbf{B} \mathbf{u} \end{aligned}$$

Then, we add to both sides $\mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u}$,

$$\begin{aligned} \dot{V}(t) + \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u} &= \dot{\mathbf{x}}^T \mathbf{P} \mathbf{x} + \mathbf{x}^T \mathbf{P} \dot{\mathbf{x}} \\ &= \mathbf{x}^T \left(\mathbf{A}^T \mathbf{P} + \mathbf{P} \mathbf{A} \right) \mathbf{x} \\ &\quad + \mathbf{u}^T \mathbf{B}^T \mathbf{P} \mathbf{x} + \mathbf{x}^T \mathbf{P} \mathbf{B} \mathbf{u} \\ &\quad + \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u}. \end{aligned}$$

Let us complete the square by adding and removing $\mathbf{x}^T \mathbf{P} \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^T \mathbf{P} \mathbf{x}$ to the right-hand side. Assume $\mathbf{R} = \mathbf{U}^T \mathbf{U}$, $\mathbf{R}$ is positive definite, and

U is invertible. Then, the above can be written as follows,

$$\begin{aligned}
 \dot{V}(t) + \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u} &= \dot{\mathbf{x}}^T \mathbf{P} \mathbf{x} + \mathbf{x}^T \mathbf{P} \dot{\mathbf{x}} \\
 &= \mathbf{x}^T \left(\mathbf{A}^T \mathbf{P} + \mathbf{P} \mathbf{A} \right) \mathbf{x} \\
 &\quad + \mathbf{u}^T \mathbf{B}^T \mathbf{P} \mathbf{x} + \mathbf{x}^T \mathbf{P} \mathbf{B} \mathbf{u} \\
 &\quad + \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u}. \\
 &\quad + \mathbf{x}^T \mathbf{P} \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^T \mathbf{P} \mathbf{x} \\
 &\quad - \mathbf{x}^T \mathbf{P} \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^T \mathbf{P} \mathbf{x} \\
 &= \mathbf{x}^T \left(\mathbf{A}^T \mathbf{P} + \mathbf{P} \mathbf{A} + \mathbf{Q} - \mathbf{P} \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^T \mathbf{P} \right) \mathbf{x} \\
 &\quad + \|\mathbf{U} \mathbf{u} + \mathbf{U}^{-T} \mathbf{B}^T \mathbf{P} \mathbf{x}\|^2.
 \end{aligned}$$

The first term is independent of the control input and, therefore, the solution to the algebraic Riccati equation,

$$\mathbf{A}^T \mathbf{P} + \mathbf{P} \mathbf{A} + \mathbf{Q} - \mathbf{P} \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^T \mathbf{P} = 0.$$

Assuming state-feedback controller $\mathbf{K}$,

$$\mathbf{K} = -\mathbf{R}^{-1} \mathbf{B}^T \mathbf{P},$$

the second term will be equal to zero and we will have,

$$\begin{aligned}
 \dot{V}(t) + \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u} &= 0, \\
 \dot{V}(t) &= -\mathbf{x}^T \mathbf{Q} \mathbf{x} - \mathbf{u}^T \mathbf{R} \mathbf{u} < 0,
 \end{aligned} \tag{2.5}$$

since $\mathbf{Q}$ and $\mathbf{R}$ are positive definite.

Considering the equality in (2.5), the optimal control cost is given by,

$$\begin{aligned}
 J_{\Lambda}^e(\mathbf{x}_0) &= \int_0^{\infty} \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u} \, dt = - \int_0^{\infty} \dot{V}(t) dt \\
 &= V(0) - V(\infty) \\
 &= \mathbf{x}^T(0) \mathbf{P} \mathbf{x}(0).
 \end{aligned}$$

2.2.2 Expected Control Performance

In order to measure the expected quality of control for a control application Λ associated with plant $\mathbf{P}$, we use a standard quadratic

cost [ÅW97]

$$J_{\Lambda}^e = \lim_{T \rightarrow \infty} \frac{1}{T} \mathbb{E} \left\{ \int_0^T \begin{bmatrix} \mathbf{x} \\ \mathbf{u} \end{bmatrix}^T \mathbf{Q} \begin{bmatrix} \mathbf{x} \\ \mathbf{u} \end{bmatrix} dt \right\}. \quad (2.6)$$

The weight matrix $\mathbf{Q}$ is given by the designer, and is a positive semi-definite matrix with weights that determine how important each of the states or control inputs are in the final control cost, relative to others ($\mathbb{E} \{ \cdot \}$ denotes the expected value of a stochastic variable). To compute the expected value of the quadratic control cost J_{Λ}^e for a given delay distribution, the Jitterbug toolbox is employed [LC02].

Intuitively, the cost function captures (1) the expected deviation of the state variables from the origin and (2) the expected energy used to control the plant. This is probably more clear if we consider the Linear-Quadratic controller (LQR) case. In the case of Linear-Quadratic controller, the problem formulation is as follows,

$$\begin{aligned} \min_{\mathbf{u}} \quad & J_{\Lambda}^e(\mathbf{x}_0) = \int_0^{\infty} \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u} \, dt \\ \text{s.t.} \quad & \dot{\mathbf{x}} = \mathbf{A} \mathbf{x} + \mathbf{B} \mathbf{u}, \\ & \mathbf{x}(0) = \mathbf{x}_0. \end{aligned}$$

The optimal cost of the problem is given by

$$J_{\Lambda}^e(\mathbf{x}_0) = \mathbf{x}^T(0) \mathbf{P} \mathbf{x}(0),$$

obtained as discussed in the previous section, where $\mathbf{P}$ is the solution to the following Riccati equation,

$$\mathbf{A}^T \mathbf{P} + \mathbf{P} \mathbf{A} + \mathbf{Q} - \mathbf{P} \mathbf{B} \mathbf{R}^{-1} \mathbf{B}^T \mathbf{P} = 0.$$

In the following, we demonstrate the relation among the delay, the sampling period, and the expected control cost through a simple example. We consider an inverted pendulum with transfer function $\frac{1}{s^2-1}$. The sampling period of the inverted pendulum is varied in the interval of $[0.1, 0.5]$. The delay is also varied and is reported in terms of the percentage of the sampling period. Note that in this example, we consider constant delays and the controller is also designed to compensate for this constant delay.

Figure 2.2: The quadratic control cost based on the sampling period and delay for an inverted pendulum. Note that the delay is given in percentage of the sampling period.

We use the Jitterbug toolbox to calculate the quadratic control cost. The results are shown in Figure 2.2. Often, the control cost, for a given sampling period, increases as the delay increases. Another observation is that for longer sampling periods, an increase in the delay (in terms of the percentage of the sampling period) has a more dramatic impact on the control cost, compared to when the sampling period is short.

Let us now consider Figure 2.3, where we change the sampling period of the inverted pendulum in the interval of $[0.2, 0.8]$ seconds, and consider a constant delay of 0.1 seconds. For the blue curve, the controller is synthesized such that it considers the constant delay and compensates for it. The red curve is the control cost of the controller which ignores the information regarding the constant delay.

Note that the controller which takes into consideration the constant delay at control-law synthesis outperforms the other controller. Furthermore, it is clear that increasing the sampling period leads to

Figure 2.3: The quadratic control cost based on the sampling period for a constant delay. The blue curve shows the case when the controller is synthesized for the constant delay, whereas the red curve is the control cost for the controller which does not consider the constant delay.

worse control performance, as it is expected. This result is intuitive, since the more often the plant is controlled, the better the control performance, and the lower the control cost.

Let us now consider Figure 2.4, where we change the constant delay of the inverted pendulum in the interval of $[0.01, 0.3]$ seconds, and consider a fixed sampling period of 0.3 seconds. The blue curve is the control cost when the controller is synthesized to account for the actual constant delay. The red curve, however, is generated for a controller which is synthesized for zero input–output delay. Observe that for long delays the control cost is not finite, i.e., the control application becomes unstable, for the controller which does not compensate for the delay.

Note that, similar to the previous case, considering the delay in the control-law synthesis process improves the control performance. Further, the shorter the delay, the better the control performance, and the lower the control cost.

Figure 2.4: The quadratic control cost based on the constant delay for a fixed sampling period. The blue curve shows the case when the controller is synthesized for the constant delay, whereas the red curve is the control cost for the controller which does not consider the constant delay during synthesis.

It has been shown that there does not exist any guaranteed margins for the standard Linear-Quadratic-Gaussian (LQG) regulators [Doy78]. In addition to this, in our problem formulation the controllers are designed for a given delay distribution. Using Jitterbug, the stability of a plant can be analyzed in the mean-square sense *if* all time-varying delays are assumed to be independent stochastic variables. However, by their nature, task and message delays do not behave as independent stochastic variables and, therefore, the stability results based on the above quadratic cost are not valid as worst-case guarantees. Hence, while the above cost function is appropriate as a metric for the average quality of control, it cannot provide a *hard guarantee* of stability in the worst case.

Figure 2.5: General control problem formulation

2.2.3 Worst-Case Control Performance

The worst-case control performance of a system can be quantified by an upper bound on the gain from the uncertainty input to the plant output. For instance, let us consider Figure 2.5, where the plant is denoted by P and the controller is denoted by K . Assuming the exogenous input w to be the uncertainty input, the worst-case performance of a system can be measured by computing an upper bound on the worst-case gain G from the exogenous input w to the plant output z . The plant output is then guaranteed to be bounded by

$$\|z\| \leq G\|w\|.$$

In order to measure the worst-case control performance of a system, we use the Jitter Margin toolbox [Cer12]. The Jitter Margin toolbox provides sufficient conditions for the worst-case stability of a closed-loop system with a linear continuous-time plant and a linear discrete-time controller. Moreover, if the closed-loop system is stable, the Jitter Margin toolbox can measure the worst-case performance of the system. The worst-case control cost is captured by

$$J_{\Lambda}^w = G(P, K, L, J_s, J_a). \quad (2.7)$$

The discrete-time controller designed for sampling period h is denoted by K . The nominal sensor–actuator (input–output) delay for control application Λ is denoted by L . The *worst-case* jitters in response times of the sensor (input) and the actuator (output) tasks of application Λ are denoted by J_s and J_a (see Section 2.3, Figure 2.8), respectively. It should be noted that a finite value for the worst-case control cost J_{Λ}^w represents a guarantee of stability for control application Λ in the

Figure 2.6: The stability curves generated by Jitter Margin and their linear lower bounds (the area below the curves is the stable area).

worst case. Furthermore, a smaller value of the worst-case gain G indicates a better worst-case control performance.

A special case of the above results is when the sampling jitter is omitted, simply by using a dedicated hardware to perform the sampling strictly periodically. In this case, the Jitter Margin toolbox [KL04, CLE⁺04, Cer12] can quantify the tolerable amount of latency and jitter experienced by a control application before the instability of the plant, or before certain worst-case performance requirements are violated.

For a given controller and latency, the Jitter Margin toolbox computes the *jitter margin* (similar to the *phase margin* and *gain margin* concepts) to guarantee the required degree of performance or stability. The Jitter Margin toolbox provides the stability curve that determines the maximum tolerable response-time jitter J based on the latency L . While the curve can instead be generated for a certain required control performance, rather than stability, we use the phrase *stability curve* in this thesis to refer to the output of the Jitter Margin toolbox.

The solid curves in Figure 2.6 are examples of the *stability curves* generated by the Jitter Margin toolbox. The area below the solid

Figure 2.7: Loop transformation of the delayed system [KL04]

curve is the stable area. The green curves are generated for a DC servo process $\frac{1000}{s^2+s}$ and discrete-time Linear-Quadratic-Gaussian (LQG) controllers. The upper and lower green curves correspond to LQG controllers with sampling periods 6 ms and 12 ms, respectively. Replacing the Linear-Quadratic-Gaussian controllers with a Proportional-Integral-Derivative (PID) controller, the blue curves in Figure 2.6 are obtained. Notice the large gap between the two blue curves generated for the two sampling periods. The upper blue curve, $h = 6$ ms, provides better stability margin (the application can tolerate larger delay and jitter) compared to the lower curve, $h = 12$ ms, but requires higher bandwidth due to more frequent activation. This trade-off motivates the need for a co-design approach where the real-time parameters are optimized along with the controllers in a unified process.

In the following, we will provide intuition on how to derive such sufficient stability conditions for the simple case of a continuous-time plant and continuous-time controller, borrowed from [KL04]. The analysis is based on the *small-gain theorem*. There are three steps in the analysis: (1) the control loop is transformed as it is shown in Figure 2.7, (2) The gains of the two parts (Δ_F and G_{wv}) of the loop are calculated, and, finally, (3) based on the small-gain theory, the stability criterion is obtained.

The main step is to compute the gain Δ_F . Let us define

$$y(t) = \int_0^t v(\tau) d\tau,$$

and

$$w(t) = y(t - \delta(t)) - y(t) = \int_{t-\delta(t)}^t v(\tau) d\tau,$$

where $\delta(t)$ is the time-varying delay parameter which is bounded above by $\bar{\delta}$, i.e., $\delta(t) \leq \bar{\delta}$. Using the Cauchy–Schwarz inequality in the first inequality and the boundedness of the delay parameter in the second inequality we obtain,

$$\begin{aligned} w(t)^2 &= \left(- \int_{t-\delta(t)}^t v(\tau) d\tau \right)^2 \leq \delta(t) \int_{t-\delta(t)}^t v(\tau)^2 d\tau \\ &\leq \bar{\delta} \int_{t-\delta(t)}^t v(\tau)^2 d\tau. \end{aligned}$$

Then, the L_2 norm of $w(t)$ can be bounded as follows,

$$\begin{aligned} \|w\|_{L_2}^2 &\leq \int_0^\infty \bar{\delta} \int_{t-\delta(t)}^t v(\tau)^2 d\tau dt = \bar{\delta} \int_0^\infty \int_{-\bar{\delta}}^0 v(t+s)^2 ds dt \\ &= \bar{\delta} \int_{-\bar{\delta}}^0 \int_0^\infty v(t)^2 dt ds \\ &= \bar{\delta} \|v\|_{L_2}^2 \int_{-\bar{\delta}}^0 1 ds \\ &= \bar{\delta}^2 \|v\|_{L_2}^2. \end{aligned}$$

This basically means that the gain Δ_F is bounded,

$$\|\Delta_F\|_{L_2} = \frac{\|w\|_{L_2}}{\|v\|_{L_2}} \leq \bar{\delta}.$$

Finally, the small-gain theorem is used to derive the stability criterion,

$$\begin{aligned} \|G_{wv}\|_{L_2} \cdot \|\Delta_F\|_{L_2} &< 1, \\ \sup_{\omega \in [0, \infty]} \left| \frac{j\omega P(j\omega)K(j\omega)}{1 + P(j\omega)K(j\omega)} \right| \cdot \bar{\delta} &< 1. \end{aligned}$$

Note that the condition could be improved if the delay parameter is also bounded from below $\underline{\delta} \leq \delta(t) \leq \bar{\delta}$. This is simply done by replacing $P(s)$ by $P(s)e^{-\underline{\delta}s}$ and redefining $\bar{\delta} = \bar{\delta} - \underline{\delta}$. In this case, the constant part of the delay is moved in the plant.

2.3 Latency and Jitter Analyses

In order to apply the worst-case control performance analysis, Equation (2.7), we shall compute the three parameters mentioned in Section 2.2.3, namely, the nominal sensor–actuator delay L , worst-case

Figure 2.8: Graphical interpretation of the nominal sensor-actuator delay L , worst-case sensor jitter J_s , and worst-case actuator jitter J_a

sensor jitter J_s , and worst-case actuator jitter J_a for each control application Λ . Figure 2.8 illustrates the graphical interpretation of these three parameters. To obtain these parameters, we can apply response-time analysis as follows,

$$\begin{aligned} J_s &= R_s^w - R_s^b, \\ J_a &= R_a^w - R_a^b, \\ L &= \frac{R_{ia}^b + R_{ia}^w}{2} - \frac{R_{is}^b + R_{is}^w}{2}, \end{aligned} \quad (2.8)$$

where R_s^w and R_s^b denote the worst-case and best-case response times for the sensor task τ_s of the control application Λ , respectively. Analogously, R_a^w and R_a^b are the worst-case and best-case response times for the actuator task τ_a of the same control application Λ .

Let us now consider the simple *latency-jitter interface*, where the sampling jitter is eliminated by introducing a dedicated hardware to perform the sampling strictly periodically.

In order to apply the stability analysis discussed, the values of the latency (L) and worst-case response-time jitter (J) of the control task should be computed. In this case, we define the two metrics based on the worst-case and best-case response times as follows,

$$\begin{aligned} L &= R^b, \\ J &= R^w - R^b, \end{aligned} \quad (2.9)$$

where R^w and R^b denote the worst-case and best-case response times, respectively.

For a given sampling period, the stability curve can safely be approximated by a linear function of the latency and worst-case response-time jitter. The linear approximation is generated by a constrained

Figure 2.9: Graphical interpretation of the latency and worst-case response-time jitter.

least-squared optimization on the original curve generated by the Jitter margin toolbox, which is computationally efficient. The linear *stability condition* for a control application is of the form

$$L + aJ \leq b, \quad (2.10)$$

where $a \geq 1, b \geq 0$. The latency L identifies the constant part of the delay that the control application experiences, whereas the worst-case response-time jitter J captures the varying part of the delay (see Figure 2.9, where R^b and R^w represent the best-case and worst-case response times, respectively).

The linear lower bounds, depicted by the dashed lines, on the original curves generated by the Jitter Margin toolbox are also shown in Figure 2.6. In [CLE⁺04], Cervin et al. discussed the fact that $L + J(L)$ is an increasing function of L , where $J(L)$ is the jitter margin for the latency L . We shall show that the coefficient a is indeed always greater than 1. Let us consider two latency values of L and L' , where $L < L'$. Based on this property, we can write the following inequality,

$$L + J(L) < L' + J'(L'),$$

which can be simplified to,

$$a = -\frac{L' - L}{J'(L') - J(L)} > 1.$$

This indicates that control applications are more sensitive to the varying part of the delay than the constant part.

Often, the linear lower bound efficiently captures the stable area identified by Jitter Margin. Although we consider only a single linear function to lower bound the curve generated by the Jitter Margin

toolbox in this thesis, it is also possible to consider a piecewise linear lower bound and perform all optimizations throughout this thesis for each linear section and then choose the best solution among all. However, for the sake of presentation, here, we consider that the stability curve can be efficiently bounded from below by a single linear function.

Considering the linear lower bound on the original stability curve, the linear stability constraint can be formulated as,

$$\begin{aligned} L + aJ &\leq b, \\ R^b + a(R^w - R^b) &\leq b. \end{aligned} \tag{2.11}$$

For a given control application, the stability condition (2.11), which is based on the exact best-case and worst-case response times, determines if the application, in the worst-case, is guaranteed to be stable.

In the context of the design and optimization, as will be discussed, the presence of discontinuous operators (ceiling) in the exact expressions (2.16) and (2.14) of the worst-case and best-case response times makes them unsuitable. Hence, we use the upper/lower bound of the worst/best-case response times (see Chapter 4) and redefine the latency and the worst-case response-time jitter as follows,

$$\begin{aligned} \underline{L} &= \underline{R}^b, \\ \bar{J} &= \bar{R}^w - \underline{R}^b. \end{aligned} \tag{2.12}$$

While using the linear supply bounds involves some pessimism compared to the original supply bounds, it is safe from the stability point of view [CLE⁺04]. Then, the stability condition is as follows,

$$\underline{L} + a\bar{J} \leq b. \tag{2.13}$$

2.4 Response-Time Analysis

In this thesis, we consider fixed-priority preemptive scheduling for its simplicity. However, mainly similar conclusions can be drawn for the dynamic-priority scheduling policies, provided the response-time analysis results are available. This section gives a brief overview on computing the worst-case and best-case response times. The response time of a job is defined as the amount of time it takes for a job

to finish its execution, from the moment it is released. The best-case and worst-case response times are, respectively, the minimum and maximum response times among all jobs of a task. The jitter is defined as the variation in the response time of a task, i.e., the difference between the best-case and worst-case response times.

2.4.1 Worst-Case Response-Time Analysis

Under fixed-priority preemptive scheduling, assuming deadline $D_i \leq h_i$ and an independent taskset, the exact worst-case response time of a task τ_i occurs once all jobs of different tasks are released at the same time as the job under analysis. The exact worst-case response time, then, occurs for the first job of the task under analysis at the so-called *critical instant* and it can be computed by the following equation [JP86],

$$R_i^w = c_i^w + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i^w}{h_j} \right\rceil c_j^w, \quad (2.14)$$

where $hp(\tau_i)$ denotes the set of higher priority tasks for task τ_i . Equation (2.14) is solved by fixed-point iteration starting with, e.g., $R_i^w = c_i^w$.

Assuming arbitrary deadlines, it has been shown that the worst-case response time can occur for the later jobs in the so-called *busy period* [Leh90]. The worst-case response time for independent tasksets with arbitrary deadlines is given by [Leh90] and [TBW94],

$$\begin{aligned} w_i(q) &= (q+1)c_i^w + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{w_i(q)}{h_j} \right\rceil c_j^w, \\ R_i^w &= \max_q \{w_i(q) - qh_i\}. \end{aligned} \quad (2.15)$$

Under the assumption of arbitrary deadlines, all instances in the busy period must be considered in order to obtain the worst-case response time.

2.4.2 Best-Case Response-Time Analysis

Under fixed-priority preemptive scheduling, assuming $D_i \leq h_i$ and an independent taskset, the exact best-case response time of a task τ_i

occurs for the job which finishes its execution at the critical instant, or the so-called *favorable instant*, if we consider the best-case analysis and the terminology in [RS02]. The exact best-case response time is given by the following equation [RS02],

$$R_i^b = c_i^b + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i^b}{h_j} - 1 \right\rceil c_j^b. \quad (2.16)$$

Similar to worst-case response-time analysis, Equation (2.16) also has to be solved by iteration, but starting from an initial value of, e.g., $R_i^b = R_i^w$. The above equation is also a valid lower bound for tasksets with arbitrary deadlines.

Assuming arbitrary deadlines, although the above equation results in a safe lower bound for the best-case response time, to compute the exact best-case response time the notion of *active period* should be considered, to account for the interference from the previous jobs [BLM13]. The exact best-case response time is given by the following equation [BLM13],

$$\begin{aligned} b_i(q) &= qc_i^b + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{b_i(q)}{h_j} - 1 \right\rceil c_j^b, \\ R_i^b &= \max_q \{b_i(q) - (q-1)h_i\}, \end{aligned} \quad (2.17)$$

where q is bounded above by the number of jobs of task τ_i in the busy period.

2.5 Properties of Latency and Jitter

In this section, we investigate the effect of changing scheduling parameters, i.e., priorities and periods, on the latency and jitter a control application experiences. In particular, it will be shown that the monotonicity property does not hold for the response-time jitter obtained based on Equations (2.14) and (2.16).

2.5.1 Analysis with Respect to Priorities

The latency and worst-case response-time jitter, defined in Equation (2.9), only depend on the best-case and worst-case response times.

Figure 2.10: Non-monotonicity of response-time jitter with respect to priorities and periods

Further, the best-case and worst-case response times only depend on the *set* of higher priority tasks. Therefore, the latency and response-time jitter remain unchanged as long as the set of higher priority tasks remains the same; then, the relative priority order of the higher priority tasks (and lower priority tasks) is irrelevant.

In addition to the above, we shall consider the effect of removing a higher priority task from the set of high priority tasks. It is clear from Equation (2.16) that removing a higher priority task results in less or equal latency. As opposed to the latency, the response-time jitter, however, does not monotonically change with removing high priority tasks due to the jumps in the worst-case and best-case response times as results of the ceiling functions in Equations (2.14), (2.15), (2.16), and (2.17). This will be illustrated using a small example.

It should be reminded that the worst-case response time occurs at the critical instant, i.e., when the task under analysis is released at the same time as all other high priority tasks [JP86]. The best-case response time occurs when the task under analysis is released such that it finishes executing simultaneously with the release of all its high priority tasks, i.e., at the favorable instant [RS02].

Let us consider three tasks $\tau_1 = (\rho_1 = 3, c_1^w = 3, c_1^b = 3, h_1 = 12)$, $\tau_2 = (2, 1, 1, 9)$, $\tau_3 = (1, 9.5, 8.5, 100)$. The priority of task τ_i is denoted by ρ_i . The worst-case and best-case scenarios for task τ_3 , under analysis, are shown in Figures 2.10(a) and 2.10(b). The set of higher priority tasks is $hp(\tau_3) = \{\tau_1, \tau_2\}$. Let us further consider implicit deadlines (deadlines equal to periods) for all three tasks. The worst-case and best-case response times are $R_3^w = 9.5 + 2 \times 3 + 2 \times 1 = 17.5$ and $R_3^b = 8.5 + 1 \times 3 + 1 \times 1 = 12.5$, respectively. The worst-case jitter in the response time of task τ_3 is $J_3 = 5$. Let us remove task τ_2 from the set of higher priority tasks of task τ_3 , i.e., $hp(\tau_3) = \{\tau_1\}$. Figures 2.10(c) and 2.10(d) show the new worst-case and best-case response times. The dotted lines show the execution of task τ_2 according to the previous scenario. The worst-case and best-case response times decrease to $R_3^w = 9.5 + 2 \times 3 + 0 \times 1 = 15.5$ and $R_3^b = 8.5 + 0 \times 3 + 0 \times 1 = 8.5$. The worst-case jitter in the response time of task τ_3 is, however, increased to $J_3 = 7$ by removing high priority task τ_2 .

2.5.2 Analysis with Respect to Periods

The latency, as defined in Equation (2.9), monotonically increases with decrease in the period of high priority tasks. Increasing the periods of higher priority tasks results in less or equal interference at the favorable instant. In other words, increasing the periods of higher priority tasks may cause some instances of these tasks to fall outside the interference scenario.

While the latency monotonically decreases with increasing the periods of the higher priority tasks, the worst-case response-time jitter does not monotonically change with periods. This can be illustrated by an example as follows. Let us consider the same taskset as in the previous example, i.e., $\tau_1 = (3, 3, 3, 12)$, $\tau_2 = (2, 1, 1, 9)$, $\tau_3 = (1, 9.5, 8.5, 100)$. The deadlines are considered to be equal to the periods. The worst-case and best-case instants are shown in Figures 2.10(a) and 2.10(b) and, as shown before, we have $R_3^w = 17.5$, $R_3^b = 12.5$, and $J_3 = 5$. Now, let us increase the period of task τ_1 to $h_1 = 13$. Figures 2.10(e) and 2.10(f) show the new worst-case and best-case scenarios. While the worst-case response time remains the same, the best-case response time of task τ_3 is decreased to $R_3^b = 8.5 + 0 \times 3 + 1 \times 1 = 9.5$, leading to an increase in the worst-case response-time jitter experienced by task τ_3 ($J_3 = 8$).

The section can be summarized in the following Remark.

Remark 2.5.1 *The latency and worst-case response-time jitter defined in Equation (2.9) have the following properties,*

1. *Priorities:*

- *The latency and worst-case response-time jitter experienced by a task are independent of the priority order of other tasks as long as the set of higher priority tasks remains the same.*
- *Increasing the priority level of a task leads to a less or equal latency for that task.*
- *Increasing the priority level of a task does not necessarily lead to a less or equal worst-case response-time jitter for that task.*

2. *Periods:*

Figure 2.11: Non-monotonicity of response-time jitter with respect to priorities

- *Shorter period for a higher priority task leads to a greater or equal latency for the task under analysis.*
- *Shorter period for a higher priority task does not necessarily lead to a greater or equal worst-case response-time jitter for the task under analysis.*

Observe that in the examples in this section, different relative phasings are considered for the task under analysis in the best-case and worst-case scenarios, which might not be possible in reality. The question is whether the previous results are still valid if we consider the same phasing in both the best-case and worst-case scenarios.

It turns out that similar conclusions can be drawn even if we consider the same phasing in the best-case and worst-case scenarios. For instance, Figure 2.11 is an example of such a scenario, with respect to task priorities. Let us consider the following taskset: $\tau_1 = (3, 3, 3, 6)$, $\tau_2 = (2, 2, 2, 8)$, $\tau_3 = (1, 1, 1, 8)$. Task τ_3 is the task under analysis and we consider only the response time jitter. As it is clear in Figure 2.11(a), before removing the higher priority task τ_2 , the jitter is $J_3 = R_3^w - R_3^b = 6 - 4 = 2$ time units, while after removing the higher priority task τ_2 , the jitter increases to $J_3 = R_3^w - R_3^b = 4 - 1 = 3$ time units, as it is shown in Figure 2.11(b).

2.6 General Problem Formulation

In this section, we would like to set the basis for the control–scheduling problem. First, we discuss the effect of delay and sampling period on the control performance and stability. Secondly, we go through the scheduling problem and clarify its connection to the controller synthesis task. Finally, we provide intuition regarding the interfaces used to capture stability and control performance.

2.6.1 The Effect of Delay and Sampling Period

Let us consider a single control task, which samples the state of the plant strictly periodically, computes the control input, and applies the control input to the plant. Figure 2.12 shows such a scenario. The upward arrows indicate the moment the state of the plant is sampled and the control input is applied when the execution of the task is finished. The distance between two sampling instants, i.e., sampling period, is constant and is denoted by h . The calculation of control input involves some computations and, therefore, there is a delay from the sampling instant to the actuation instant. Let us denote the constant delay between sampling and actuation by L . Note that the control input is kept constant by a hold circuit between two actuation instants.

To clarify, in Figure 2.12, the state of the plant is sampled at time $k \cdot h$ and this state information is used to compute the control input. While the control task is calculating the control input, the previous control input, based on the sampling at time $(k - 1) \cdot h$, is being applied. At time $k \cdot h + L$, the computation of control input based on the new sampled state is done and the new input is applied until the next actuation point. At time $(k + 1) \cdot h$, the state is sampled again and the control task computes the next control input and it will be applied at time $(k + 1) \cdot h + L$.

We consider the simplified dynamical system

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

In this case, the system can be modeled as follows,

$$x[k + 1] = \bar{A}x[k] + \bar{B}_L u[k - 1] + \bar{B}_{h-L} u[k],$$

Figure 2.12: The schedule for a single task with periodic sampling and constant delay.

where

$$\begin{aligned}\bar{\mathbf{A}} &= e^{\mathbf{A}h}, \\ \bar{\mathbf{B}}_L &= e^{\mathbf{A}(h-L)} \int_0^L e^{\mathbf{A}t} dt \mathbf{B}, \\ \bar{\mathbf{B}}_{h-L} &= \int_0^{h-L} e^{\mathbf{A}t} dt \mathbf{B}.\end{aligned}$$

Assuming the state feedback controller $\mathbf{u}[k] = \mathbf{K}\mathbf{x}[k]$, the dynamics can be written as follows,

$$\mathbf{x}[k+1] = \bar{\mathbf{A}}\mathbf{x}[k] + \bar{\mathbf{B}}_L \mathbf{K}\mathbf{x}[k-1] + \bar{\mathbf{B}}_{h-L} \mathbf{K}\mathbf{x}[k].$$

This basically means that while the new control input is being computed based on the new sampled state (i.e., for the first L time units after each sampling instant) the last control input computed based on the previous sampled state is applied.

Finally, let us extend the state vector as follows,

$$\mathbf{z}[k] = \begin{bmatrix} \mathbf{x}[k] \\ \mathbf{x}[k-1] \end{bmatrix}. \quad (2.18)$$

In this case, the system can be written as follows,

$$\mathbf{z}[k+1] = \mathbf{\bar{A}}\mathbf{z}[k],$$

Figure 2.13: The schedule for two periodic tasks introduces variation in the delay experienced by the low priority task.

where the closed-loop matrix $\mathring{\mathbf{A}}$ is given by,

$$\mathring{\mathbf{A}} = \begin{bmatrix} \bar{\mathbf{A}} + \bar{\mathbf{B}}_{h-L}\mathbf{K} & \bar{\mathbf{B}}_L\mathbf{K} \\ \mathbf{I} & \mathbf{0} \end{bmatrix}.$$

This shows that the closed-loop dynamics depend on both the sampling period h and constant delay L . For instance, the system is guaranteed to be stable if all the eigenvalues of matrix $\mathring{\mathbf{A}}$ are inside the unit circle.

Note that the control gain $\mathbf{K}$ is synthesized for a sampling period h and zero sensor–actuator delay. However, if the model of the dynamical system is available and the delay is constant and known, it is possible to compensate for this delay.

To account for the delay, based on the dynamical model, the constant and known delay, and the state of the plant at the sampling instant, we compute the state of the plant at the actuation instant. Then, we simply synthesize the control gain $\mathbf{K}$ for sampling period h and zero delay, but the control input is calculated based on the predicted state of the plant at the actuation instant.

Observe that we can only account for the delay if the delay between sampling and actuation is constant and known. This is not the case when the platform is shared among several tasks, as it is considered in this thesis. The competition among tasks for execution results in a time-varying time delay for the low priority task. This has been shown in Figure 2.13, where we introduce a higher priority task

which competes for execution on the shared platform. In the example in Figure 2.13, it is still possible to design the controller to take into consideration the deterministic delay pattern [XÅC⁺15]. The idea is to consider all the jobs in one hyper-period instead of only a single job. This is only possible due to the deterministic pattern in the input-output delays. Nevertheless, the execution-time of each task varies due to different platform states or input data, and this leads to varying and unpredictable interference patterns. Moreover, the delay compensation is not straightforward in the presence of complex timing behaviors. Similar conclusions can be drawn if we consider that the sampling is not strictly periodic.

To sum up, the sensor-actuator (input-output) delay and sampling period are among the most important factors which determine the control performance and stability margins of a control application. Further, it has been discussed that it is not straightforward to deal with complex timing behaviors introduced as a result of resource sharing.

2.6.2 Task Scheduling and Controller Synthesis

Early in this section, it has been discussed that the delay characteristics in the control-law synthesis process have a dramatic impact on the performance (see Figure 2.4) and stability (see Figure 2.6) of control applications. In particular, the sampling period which may be used for the controller to guarantee certain performance level depends on the delay experienced by the controller (see Figures 2.6 and 2.3). The sampling period obviously has a direct impact on task scheduling. Finally, the delay experienced by a control task is determined by the task scheduling process. This clarifies the interdependency of the controller synthesis and task scheduling.

The problem becomes more complex if we consider that changing the parameters of a task affects the performance of other tasks. In addition, intuitively better choices of the parameters of a task might not necessarily be better from the control performance and stability points of view. For example, increasing the priority level of a task might mean larger jitter and, thus, instability for the corresponding control application (see Section 2.5).

To sum up, the interdependence between controller synthesis and task scheduling motivates the need for a co-design methodology. The

traditional approaches based on the *separation of concerns* principle treat these two processes separately and often obtain a suboptimal solution.

2.6.3 Delay Distribution, Latency, and Jitter

It is important to realize that the most rigorous interface between control synthesis and task scheduling is the exact delay patterns. This is, however, often not available except for the case of very restricted scheduling policies. The next closest option is the delay distribution. The delay distribution basically determines the control performance of the application. In fact, the Jitterbug toolbox uses the delay distribution to compute the expected performance of a control application.

A considerably less expressive interface is the latency–jitter interface. The latency–jitter interface abstracts the exact delay patterns by considering only the extreme values. Note that the latency–jitter interface says very little about the distribution of delay. Therefore, it is not an appropriate metric for measuring control quality. In other words, similar values of latency and jitter might lead to completely different control qualities. However, when it comes to providing hard stability guarantees, we have to consider the worst-case scenario and this interface is simple enough to capture sufficient conditions for stability. The Jitter Margin toolbox uses this interface to investigate stability. Note that, as shown in Section 2.3, the coefficient a is always greater than or equal to one. This basically means that the worst-case stability condition is more sensitive to an increase in the jitter than the latency.

CONTROL-QUALITY DRIVEN DESIGN WITH STABILITY GUARANTEES

As discussed in the previous chapters, the resource sharing among several control applications leads to complex timing behaviors that degrades the quality of control, and more importantly, can jeopardize stability in the worst-case, if not properly taken into account during design. The two main requirements of control applications are: (1) robustness and, in particular, stability and (2) high control quality. Although stability of the control applications is absolutely essential, a design flow driven by the worst-case scenario often leads to poor control quality since the design is tuned towards the worst-case scenario that may occur very rarely. On the other hand, designing the system merely based on control quality, determined by the expected (average-case) behavior, does not guarantee the stability of control applications in the worst case. Therefore, both control quality and worst-case stability have to be considered during the design process, i.e., period assignment, task scheduling, and control-synthesis. In this chapter, we present an integrated control-scheduling co-design approach to optimize the expected control performance, while guaranteeing stability.

3.1 Introduction and Related Work

The design of embedded control systems involves two main tasks, synthesis of the controllers and implementation of the control ap-

plications on a given execution platform, as discussed before. Controller synthesis comprises period assignment, delay compensation, and control-law synthesis. Implementation of the control applications, on the other hand, is mostly concerned with allocating computational resources to these applications (e.g., mapping and scheduling).

Traditionally, the problem of designing embedded control systems has been dealt with in two independent steps, where first the controllers are synthesized and, then, applications are implemented on a given platform. However, this approach often leads to either resource under-utilization or poor control performance and, in the worst case, may even jeopardize the stability of control applications because of timing problems which can arise due to certain implementation decisions [WNT95], [ÅCES00]. As shown in the previous chapter, in order to achieve high control performance while guaranteeing stability even in the worst case, it is essential to consider the temporal properties extracted from the system schedule during control synthesis and to keep in view control performance and stability during system scheduling.

Rehbinder and Sanfridson [RS00] studied the integration of offline scheduling and optimal control. They found the static-cyclic schedule [Kop97] which minimizes a quadratic cost function. Although the stability of control applications is guaranteed, the authors mentioned the intractability of their method and its applicability only for systems limited to a small number of controllers. Majmudar et al. [MSZ11] proposed a performance-aware static-cyclic scheduler synthesis approach for control systems. The optimization objective is based on the $\mathcal{L}_\infty$ to RMS gain. Goswami et al. [GLSC12] proposed a time-triggered implementation method for mixed-criticality automotive software. The optimization is performed considering an approximation of a quadratic cost function, in the absence of noise and disturbance. Considering time-triggered implementation, it is straightforward to guarantee stability, as a consequence of removing the element of jitter. However, to completely avoid jitter, the periods of applications are constrained to be related to each other (harmonic relationship) or the data are unnecessarily buffered for the worst-case delay. Therefore, such approaches can lead to resource under-utilization and, possibly, poor control performance due to long sampling periods or long delays [RS00], [ÅC05]. The approaches in [RS00], [MSZ11], and [GLSC12] are restricted to static-cyclic and

time-triggered scheduling.

Seto et al. [SLSS96] found the optimal periods for a set of controllers on a uniprocessor platform with respect to a given performance index. Bini and Cervin [BC08] proposed a delay-aware period assignment approach to maximize the control performance measured by a standard quadratic cost function on a uniprocessor platform. Ben Gaid et al. [BCH06] considered the problem of networked control systems and the objective is to minimize a quadratic cost function. Cervin et al. [CLE⁺04] proposed a control–scheduling co-design procedure to yield the same relative performance degradation for each control application. Zhang et al. [ZSWM08] considered the control–scheduling co-design problem where the objective function to be optimized is the sum of the H_∞ norm of the sensitivity functions of all controllers. In [SCEP09] and [ASEP11], we proposed optimization methodologies for integrated mapping, scheduling, and control synthesis, to maximize control performance by minimizing a standard quadratic cost, on a distributed platform.

In order to capture control performance, two kinds of metrics are often used: (1) stochastic control performance metrics and (2) robustness (stability-related) metrics. The former identify the expected (*mathematical expectation*) control performance of a control application, whereas the latter are considered to be a measure of the worst-case control performance. Although considering both the expected control performance and worst-case control performance during the design process is crucial, previous work only focuses on one of the two aspects. The main drawback of such approaches, e.g., based solely on the expected control performance, is that the resulting high-quality design solution does not necessarily satisfy the stability requirements in the worst-case scenario. On the other hand, considering merely the worst case, often results in a system with poor expected control performance. This is due to the fact that the design is solely tuned to a scenario that occurs rarely. Thus, even though the overall design optimization goal should be the expected control performance, taking the worst-case control stability into consideration during the design space exploration is indispensable.

In this chapter, we propose an integrated control–scheduling approach to design embedded control systems with guarantees on the worst-case robustness, where the optimization objective is the ex-

pected control performance.

In the next section, we present the system model, i.e., plant, platform, and application models. Section 3.3 discusses the metrics for the expected and the worst-case control performance, and control synthesis. Analysis of real-time attributes is outlined in Section 3.4. In Section 3.5, we present motivational examples and in Section 3.6, we formulate our control-scheduling co-design problem. The proposed design flow is discussed in Section 3.7. Section 3.8 contains the experimental setup and results. Finally, the chapter will be concluded in Section 3.9.

3.2 System Model

As discussed in the previous chapter, the system model consists of the plant model, the platform model, and the application model.

In this chapter, we assume that the plant output is sampled periodically with certain delays after the sampling instants. Further, the control signal will be updated periodically with certain delays and is held constant between two updates.

The platform considered in this chapter is a uniprocessor. However, the results in this chapter are also valid in the case of distributed platforms [AEPC13a].

For each plant $\mathbf{P}_i \in \mathbf{P}$ there exists a corresponding control application denoted by $\Lambda_i \in \mathbf{\Lambda}$, where $\mathbf{\Lambda}$ indicates the set of applications in the system. Each application Λ_i is modeled as a task graph. Each node corresponds to a task and the dependencies are captured by edges between two tasks. We denote the j -th task of application Λ_i by τ_{ij} . The execution time, c_{ij} , of the task τ_{ij} is modeled as a stochastic variable with probability function ξ_{ij} , bounded by the best-case execution time c_{ij}^b and the worst-case execution time c_{ij}^w .

Control applications can typically provide a satisfactory control performance within a range of sampling periods [ÅW97]. Hence, each application Λ_i can execute with a period $h_i \in \mathbf{H}_i$, where $\mathbf{H}_i$ is the set of suggested periods application Λ_i can be executed with. However, the actual period for each control application is determined during the co-design procedure, considering the direct relation between scheduling parameters and control synthesis.

3.3 Control Performance and Synthesis

In this section, we introduce control performance metrics both for the expected and worst-case. We also present preliminaries related to control synthesis.

3.3.1 Expected Control Performance

In order to measure the expected quality of control for a controller Λ_i , we use, as discussed in Section 2.2.2, a standard quadratic cost [ÅW97]

$$J_{\Lambda_i}^e = \lim_{T \rightarrow \infty} \frac{1}{T} \mathbb{E} \left\{ \int_0^T \begin{bmatrix} \mathbf{x}_i \\ \mathbf{u}_i \end{bmatrix}^\top Q_i \begin{bmatrix} \mathbf{x}_i \\ \mathbf{u}_i \end{bmatrix} dt \right\}. \quad (3.1)$$

To compute the expected value of the quadratic control cost $J_{\Lambda_i}^e$ for a given delay distribution, the Jitterbug toolbox is employed [LC02].

The stability of a plant can be investigated using the Jitterbug toolbox in the mean-square sense if all time-varying delays can be modeled by independent stochastic variables. However, since it is not generally possible to model all time-varying delays by independent stochastic variables, this metric (Equation (3.1)) is not appropriate as a guarantee of stability in the worst-case scenario.

3.3.2 Worst-Case Control Performance

As discussed in Section 2.2.3, the worst-case control performance of a system can be quantified by an upper bound on the gain from the uncertainty input to the plant output. For instance, assuming the disturbance d to be the uncertainty input, the worst-case performance of a system can be measured by computing an upper bound on the worst-case gain G from the disturbance d to the plant output y . The plant output is then guaranteed to be bounded by

$$\|y\| \leq G\|d\|. \quad (3.2)$$

In order to measure the worst-case control performance of a system, we use the Jitter Margin toolbox [Cer12]. The Jitter Margin toolbox provides sufficient conditions for the worst-case stability of a closed-loop system. Moreover, if the closed-loop system is stable, the

Jitter Margin toolbox can measure the worst-case performance of the system.

The worst-case control cost $J_{\Lambda_i}^w$ is determined by gain G and it depends on the plant, discrete-time controller, nominal sensor–actuator (input–output) delay L_i , worst-case sensor (input) jitter J_{is} , and worst-case actuator (output) jitter J_{ia} (see Section 3.4, Figure 3.1). It should be noted that a finite value for the worst-case control cost $J_{\Lambda_i}^w$ represents a guarantee of stability for control application Λ_i in the worst case.

3.3.3 Control Synthesis

For a given sampling period h_i and a given, constant sensor–actuator delay (i.e., the time between sampling the output $\mathbf{y}_i$ and updating the controlled input $\mathbf{u}_i$), it is possible to find the control-law $\mathbf{u}_i$ that minimizes the expected cost $J_{\Lambda_i}^e$ [ÅW97]. Thus, the optimal controller can be designed if the delay is constant at each execution instance of the control application. Since the overall performance of the system is determined by the expected control performance, the controllers shall be designed for the expected (average) behavior of the system. System simulation is performed to obtain the delay distribution and the expected sensor–actuator delay and the controllers are designed to compensate for this expected amount of delay. In order to synthesize the controller we use MATLAB and the Jitterbug toolbox [LC02].

The sensor–actuator delay is in reality not constant at runtime due to the interference from other applications competing for the shared resources. The quality of the synthesized controller is degraded if the sensor–actuator delay distribution is different from the constant one assumed during the control-law synthesis. The overall expected control quality of the controller for a given delay distribution is obtained according to Section 3.3.1.

3.4 Delay and Jitter Analyses

In order to apply the worst-case control performance analysis, we shall compute the three parameters mentioned in Section 3.3.2, namely, the nominal sensor–actuator delay L_i , worst-case sensor jitter J_{is} , and worst-case actuator jitter J_{ia} for each control application Λ_i . Figure

Figure 3.1: Graphical interpretation of the nominal sensor-actuator delay L , worst-case sensor jitter J_s , and worst-case actuator jitter J_a

3.1 illustrates the graphical interpretation of these three parameters. To obtain these parameters, we can apply response-time analysis as follows,

$$\begin{aligned} J_{is} &= R_{is}^w - R_{is}^b, \\ J_{ia} &= R_{ia}^w - R_{ia}^b, \\ L_i &= \frac{R_{ia}^b + R_{ia}^w}{2} - \frac{R_{is}^b + R_{is}^w}{2}, \end{aligned} \quad (3.3)$$

where R_{is}^w and R_{is}^b denote the worst-case and best-case response times for the sensor task τ_{is} of the control application Λ_i , respectively. Analogously, R_{ia}^w and R_{ia}^b are the worst-case and best-case response times for the actuator task τ_{ia} of the same control application Λ_i .

In this chapter, we consider fixed-priority scheduling. In Section 2.4, we gave a brief overview on computing the worst-case and best-case response times.

Under fixed-priority scheduling, assuming deadline $D_i \leq h_i$ and an independent taskset, the worst-case response time of task τ_{ij} can be computed by the following equation [JP86],

$$R_{ij}^w = c_{ij}^w + \sum_{\tau_{ab} \in hp(\tau_{ij})} \left\lceil \frac{R_{ij}^w}{h_a} \right\rceil c_{ab}^w, \quad (3.4)$$

where $hp(\tau_{ij})$ denotes the set of higher priority tasks for the task τ_{ij} . Equation (3.4) is solved by fixed-point iteration starting with e.g., $R_{ij}^w = c_{ij}^w$.

For the best-case response time analysis under fixed priority scheduling, we use the bound given by [PGGGGH98]

$$R_{ij}^b = c_{ij}^b + \sum_{\tau_{ik} \in pr(\tau_{ij})} c_{ik}^b, \quad (3.5)$$

where the set $pr(\tau_{ij})$ is the set of predecessors for task τ_{ij} in the task graph model of application Λ_i .

3.5 Motivational Example

The examples in this section motivate the need for a proper design space exploration in order to be able to design a high expected performance embedded control system with guaranteed worst-case stability. The first example illustrates that considering only the expected control performance could jeopardize the stability of a control system. The second example illustrates that the design space exploration should be done considering the expected control cost (Equation (3.1)) as the objective function, subject to the constraints on worst-case cost (Equation (3.2)).

We consider (see Equation (3.1)) the weight matrix $Q_i = \text{diag}(1, 0, 0.001)$ for each application Λ_i . All time quantities are given in units of 10 milliseconds throughout this section. We synthesize the discrete-time Linear-Quadratic-Gaussian (LQG) controllers for a given period and constant expected sensor-actuator delay using the Jitterbug toolbox [LC02] and MATLAB. We consider the continuous-time disturbance $\mathbf{v}_i$ to have a covariance $R_{1i} = 1$ and discrete-time measurement noise $\mathbf{e}_i$ to have the covariance $R_{2i} = 0.01$. The total expected control cost, for a set of plants $\mathbf{P}$, is $J_{\text{total}}^e = \sum_{\mathbf{P}_i \in \mathbf{P}} J_{\Lambda_i}^e$, whereas the total worst-case control cost is defined to be $J_{\text{total}}^w = \sum_{\mathbf{P}_i \in \mathbf{P}} J_{\Lambda_i}^w$.

3.5.1 Example 1: System Design Driven by Expected Control Quality

In this example, we consider two plants $\mathbf{P}_1$ and $\mathbf{P}_2$ ($\mathbf{P} = \{\mathbf{P}_1, \mathbf{P}_2\}$). The corresponding applications Λ_1 and Λ_2 are two controllers modeled as two task graphs each consisting of a sensor task τ_{is} , a computation task τ_{ic} , and an actuator task τ_{ia} as shown in Figure 3.2(a). The numbers in parentheses specify the execution times of the corresponding tasks. The execution time of the control task τ_{2c} is uniformly distributed in the interval of $(0, 6)$. The other tasks, on the other hand, have constant execution times. Moreover, let us consider that both applications are released synchronously.

The applications Λ_1 and Λ_2 have periods $h_1 = h_2 = 30$. Consid-

Figure 3.2: Motivational examples

ering the control application Λ_2 to be the higher priority application, the total expected control cost for control applications Λ_1 and Λ_2 , J_{total}^e , computed by the Jitterbug toolbox, is minimized and is equal to 14.0. However, if we use the Jitter Margin toolbox [Cer12], we realize that there is no guarantee that the plant $\mathbf{P}_1$ will be stable (the cost $J_{\Lambda_1}^w$ is infinite). The values of the nominal sensor–actuator delay L_1 , sampling jitter J_{1s} , and actuator jitter J_{1a} are 6.0, 6.0, and 6.0, respectively.

Inverting the priority order of the applications, we can guarantee the stability of control application Λ_1 as a result of decrease in the sampling jitter J_{1s} and actuator jitter J_{1a} , without jeopardizing the stability of control application Λ_2 (since the nominal sensor–actuator delay L_2 , sampling jitter J_{2s} , and actuator jitter J_{2a} remain the same). For this priority order, the values of the nominal sensor–actuator delay L_1 , sampling jitter J_{1s} , and actuator jitter J_{1a} are 6.0, 0.0, and 0.0, respectively. This solution guarantees the stability of both control applications, although the total expected control cost J_{total}^e for control applications Λ_1 and Λ_2 is slightly larger than before, i.e., 15.2. Thus, considering only the maximization of the expected control performance during the design process can jeopardize the worst-case stability of applications. We also observe that the guaranteed worst-case stability could be obtained with only marginal decrease in the overall expected control quality.

3.5.2 Example 2: Stability-Aware System Design with Expected Control Quality Optimization

We consider three plants P_1 , P_2 , and P_3 ($\mathbf{P} = \{P_1, P_2, P_3\}$) to be controlled, where task graph models of the corresponding control applications Λ_1 , Λ_2 , and Λ_3 are shown in Figure 3.2(b). Let us assume that application Λ_i has a higher priority than application Λ_j iff $i > j$. Our objective is to find a stable solution with high control performance.

Considering the initial period assignment to be $h_1 = 60$, $h_2 = 60$, and $h_3 = 20$, the stability of the plant P_1 is not guaranteed (the worst-case control cost $J_{\Lambda_1}^w$ is not finite) even though this design solution provides high expected control performance, $J_{\text{total}}^e = 2.7$. For this period assignment, we obtain $L_1 = 12$, $J_{1s} = 16$, $J_{1a} = 22$ for application Λ_1 .

In order to decrease the amount of interference experienced by low priority application Λ_1 from high priority applications Λ_2 and Λ_3 , we increase the period of application Λ_3 to $h_3 = 60$. Further, since a smaller period often leads to higher control performance, we decrease the period of the control application Λ_1 to $h_1 = 30$. Applying delay and jitter analyses for this period assignment, we compute the values of the nominal sensor-actuator delay $L_1 = 9$, worst-case sensor jitter $J_{1s} = 16$, and worst-case actuator jitter $J_{1a} = 16$, which are smaller than the corresponding values for the initial period assignment. However, this modification does not change the nominal sensor-actuator delay, worst-case sensor jitter, and worst-case actuator jitter for the control applications Λ_2 and Λ_3 . As a result, we can guarantee the stability of all applications in the worst-case scenario since the worst-case control costs for all three applications is finite. Moreover, the total worst-case and expected control costs for this period assignment are $J_{\text{total}}^w = 2.3$ and $J_{\text{total}}^e = 20.4$, respectively.³ We observe that stability in the worst-case has been achieved with a severe degradation of the expected control performance.

Another possible solution would be $h_1 = 60$, $h_2 = 60$, and $h_3 = 60$, that leads to a total worst-case control cost equal to $J_{\text{total}}^w = 5.7$ and

³It is good to mention that the metrics for the worst-case (J^w) and expected case (J^e) are expressing different properties of the system and, thus, are not comparable with each other ($J_{\text{total}}^w = 2.3$, and $J_{\text{total}}^e = 20.4$, does not mean that the worst-case control cost is better than the expected one.).

a total expected cost $J_{\text{total}}^e = 3.0$. It should be mentioned that the values of the nominal sensor-actuator delay, worst-case sensor jitter, and worst-case actuator jitter remain the same as the previous period assignment for all three applications. Considering the applications are released simultaneously, as for the previous period assignment, the low priority application Λ_1 experiences interference by high priority applications every other time (since it executes twice as frequently as the high priority applications). Therefore, the decrease in the expected control cost is due to removing the variation in the response-time of the control application Λ_1 .

Having finite total worst-case costs for both period assignments indicates the worst-case stability of both design solutions. However, although the total worst-case control cost (J_{total}^w) is smaller in the former period assignment, the latter is the desirable design solution since the overall performance of the system is determined by the expected control performance (J_{total}^e). While it is of critical importance to guarantee stability, it is highly desirable to keep the inherent quality degradation in the expected control performance as small as possible.

We conclude that, just optimizing the expected control quality can lead to unsafe solutions which are unstable in the worst-case. Nonetheless, focusing only on stability, potentially, leads to low overall quality. Therefore, it is essential and possible to achieve both safety (worst-case stability guarantees) and high level of expected control quality.

3.6 Problem Formulation

The inputs for our co-design problem are

- a set of plants $\mathbf{P}$ to be controlled,
- a set of control applications $\mathbf{\Lambda}$,
- a set of sampling periods $\mathbf{H}_i$ for each control application Λ_i ,
- execution-time probability functions of the tasks (including the best-case and worst-case execution times).

The outputs of our co-design tool are the period h_i for each control application Λ_i , unique priority ρ_i for each application Λ_i , and the

control law $\mathbf{u}_i$ for each plant $\mathbf{P}_i \in \mathbf{P}$ (the tasks within an application have the same priority equal to the application priority). The outputs related to the controller synthesis are the period h_i and the control law $\mathbf{u}_i$ for each plant $\mathbf{P}_i$.

As mentioned before, there exists a control application $\Lambda_i \in \Lambda$ corresponding to each plant $\mathbf{P}_i \in \mathbf{P}$ and the final control quality is captured by the weighted sum of the individual control costs $J_{\Lambda_i}^e$ (Equation (3.1)) of all control applications $\Lambda_i \in \Lambda$. Hence, the cost function to be minimized is

$$\sum_{\mathbf{P}_i \in \mathbf{P}} w_{\Lambda_i} J_{\Lambda_i}^e, \quad (3.6)$$

where the weights w_{Λ_i} are determined by the designer.

To guarantee stability of control application Λ_i , the worst-case control cost $J_{\Lambda_i}^w$ (Equation (3.2)) must have a finite value. However, in addition to stability, the designer may require an application to satisfy a certain degree of robustness which is captured by the following criteria,

$$J_{\Lambda_i}^w < \bar{J}_{\Lambda_i}^w, \quad \forall \mathbf{P}_i \in \mathbf{P}, \quad (3.7)$$

where $\bar{J}_{\Lambda_i}^w$ is the limit on tolerable worst-case cost for control application Λ_i and is decided by the designer. If the requirement is for an application Λ_i only to be stable in the worst-case, the constraint on the worst-case cost $J_{\Lambda_i}^w$ is to be finite.

3.7 Co-design Approach

The overall flow of our proposed optimization approach is illustrated in Figure 3.3. In each iteration, each application is assigned a period by our period optimization method (see Section 3.7.1). Having assigned a period to each application, we proceed with determining the priorities of the applications (see Section 3.7.2). This process takes place in two main steps. In the first step, we analyze the worst-case sensitivity (worst-case sensitivity is illustrated in Section 3.7.2.1) of control applications and at the same time we synthesize the controllers (see Section 3.7.2.2). The analysis of worst-case sensitivity is done based on the worst-case control performance constraints for control applications. The higher the sensitivity level of a control application,

Figure 3.3: Overall flow of our approach

the smaller the amount of delay and jitter it can tolerate before hitting the worst-case control performance limit. Therefore, in this step, the applications are clustered in several groups according to their sensitivity and the priority level of a group is identified by the sensitivity level of its applications. In addition, in the first step, we can figure out if there exists, at all, any possible priority assignment which satisfies the worst-case robustness requirements with the current period assignment. The outputs of this step are a sequence of groups, in ascending order of sensitivity, and the synthesized controllers. While the first step has grouped applications according to their sensitivity, the second step is concerned with assigning priorities to each individual application (see Section 3.7.2.3). This will be taken care of by an inside group optimization, where we take the expected control performance into account during the priority optimization step. Having found the periods and priorities, we perform system simulation to obtain the delay distributions for all sensor and actuator tasks. These delay distributions are then used by the Jitterbug toolbox to compute the total expected control cost. The optimization will be terminated once a satisfactory design solution is found.

3.7.1 Period Optimization

The solution space exploration for the periods of control applications is done using a modified coordinate search [NW99] combined with the direct search method [HJ61]. Both methods belong to the class of derivative-free optimization methods, where the derivative of the objective function is not available or it is time consuming to obtain. These methods are desirable for our optimization since the objective function is the result of an inside optimization loop (i.e., the objective function is not available explicitly) and it is time consuming to approximate the gradient of the objective function using finite differences [NW99]. We describe our period optimization approach in two steps, where the first step identifies a promising region in the search space and the second step performs the search in the region identified by the first step, to achieve further improvements.

In the first step, we utilize a modified coordinate search method to acquire information regarding the search space and to drive the optimization towards a promising region. The coordinate search method cycles through the coordinate directions (here, the coordinates are

along the application periods) and in each iteration performs a search along only one coordinate direction. Our modified coordinate search method is guided by the expected control performance, taking into consideration the worst-case requirements of the applications. In the initial step, all control applications are assigned their longest periods in their period sets. If there exists a control application violating its worst-case control performance criterion, its period is decreased to the next longest period in the period set; otherwise, the period of the control application with the highest expected control cost is decreased to the next longest period in the period set. This process is repeated until the expected control performance cannot be further improved under the worst-case control performance requirements of the applications.

The solution obtained in the first step (period assignment), places us in a promising region of the solution space. In the second step, the direct search method is utilized to achieve further improvements. This method employs two types of search moves, namely, exploratory and pattern. The exploratory search move is used to acquire knowledge concerning the behavior of the objective function. The pattern move, however, is designed to utilize the information acquired by the exploratory search move to find a promising search direction. If the pattern move is successful, the direct search method attempts to perform further steps along the pattern move direction with a series of exploratory moves from the new point, followed by a pattern move.

Figure 3.4 illustrates the coordinate and direct search methods using a simple example consisting of two control applications. Therefore, in this example, the search space is two-dimensional, where each dimension corresponds to the period of one of the control applications. The dashed curves are contours of the objective function and the minimizer of the objective function in this example is $x^* = (h_1^*, h_2^*)$. The green arrows are the moves performed by the coordinate search method in the first step. The blue and red arrows are exploratory and pattern moves, respectively, performed by the direct search method in the second step.

In this section, we have considered the period optimization loop in Figure 3.3. Inside the loop, application priorities have to be determined and controllers have to be synthesized such that the design goals are achieved. This will be described in the following section.

Figure 3.4: The coordinate and direct search methods

3.7.2 Priority Optimization and Control Synthesis

The problem of priority assignment has previously been addressed in the context of real-time applications with hard deadlines. The priority assignment problem for hard real-time applications focuses on assigning priorities such that all tasks are schedulable. Optimality of rate monotonic priority assignment for independent synchronous tasksets with implicit deadlines is shown by Serlin [Ser72] and Liu and Layland [LL73]. In the case of constrained deadlines and synchronous tasksets, it is shown that deadline monotonic priority assignment is the optimal policy [LW82]. Audsley [Auds91] proposed an optimal priority assignment algorithm for independent asynchronous tasksets. The algorithm is also applicable to tasksets with arbitrary deadlines. Davis and Burns [DB07] proposed a robust priority assignment algorithm based on Audsley's priority assignment. Note that *optimality* is defined with regard to the respective schedulability test, i.e., a priority assignment policy is referred to as optimal if, considering the given schedulability test, there are no tasksets that are schedulable by another priority assignment policy, that are not schedulable by the optimal priority assignment [DB07]. Recently, Mancuso et al. [MBP14] proposed a control-quality driven priority assignment for control applications considering a linearization of the original control cost function.

While optimal priority assignment for hard real-time applications has been discussed to a great extent, it has gained less attention in the context of control applications. Our priority optimization approach consists of two main steps. The first step makes sure that the worst-case control performance constraints are satisfied, whereas the second step improves the expected control performance of the control applications.

3.7.2.1 Worst-Case Sensitivity and Sensitivity Groups

The notion of worst-case sensitivity is closely related to the amount of delay and jitter a control application can tolerate before violating the robustness requirements. The sensitivity level of an application is captured by Algorithm 1. Further, Algorithm 1 clarifies the relation between sensitivity level and priority level of a group and identifies the sensitivity groups ($\mathbf{G}_i$). The notion of sensitivity is linked to priority by the fact that high priority applications experience less delay and jitter, as defined in Section 3.4. This is, however, not always the case (see Section 2.5).

3.7.2.2 Sensitivity-Based Application Clustering

The algorithm for the worst-case sensitivity analysis and application clustering is outlined in Algorithm 1. The main idea behind this algorithm is to cluster the applications which have the same level of sensitivity in the same group.

To cluster applications, we look for the set of applications which can satisfy their worst-case control performance requirements even if they are at the lowest priority level. This group ($\mathbf{G}_1$) of applications can be considered the least sensitive set of applications. We remove these applications from the set of applications (Line 24) and proceed with performing the same procedure for the remaining applications. This process continues until either the set of remaining applications ($\mathbf{S}$) is empty (Line 17) or none of the remaining applications can satisfy its requirements if it is assigned the lowest priority among the remaining applications (Line 20), which indicates that there does not exist any priority assignment for the current assigned periods which can guarantee the worst-case control performance requirements for all applications.

In order to figure out whether an application meets its worst-case robustness requirements, we perform the best-case and worst-case response-time analyses (Equations (3.4) and (3.5)) for the sensor and actuator tasks considering $hp(\Lambda_i) = \mathbf{S} \setminus \{\Lambda_i\}$ (Line 7) and compute the nominal sensor-actuator delay, worst-case sensor jitter, and worst-case actuator jitter (Equation (3.3)) for the application under analysis (Line 8). Moreover, we shall synthesize the control-law and compensate for the proper amount of delay. Therefore, we design an LQG controller and compensate for the expected sensor-actuator delay using MATLAB and Jitterbug (Line 10). The expected sensor-actuator delay is extracted from the schedule in our system simulation environment, where the remaining applications constitute the set of higher priority applications ($hp(\Lambda_i) = \mathbf{S} \setminus \{\Lambda_i\}$) for the application under analysis (Line 9). Having the controller and the values of the nominal sensor-actuator delay, worst-case sensor jitter, and worst-case actuator jitter, we can calculate the worst-case control cost using the Jitter Margin toolbox (Equation (3.2)) (Line 11) and check if the worst-case requirements (Equation (3.7)) are fulfilled (Line 12).

3.7.2.3 Inside Group Optimization

Based on Algorithm 1 in the previous section, we have grouped the applications according to their sensitivity level. In this section, we shall assign a unique priority to each application such that the expected control performance is improved. The grouping realized by Algorithm 1 guarantees the worst-case control performance and robustness requirements (Equation (3.7)) of an application Λ_i assigned to group $\mathbf{G}_j$, as long as all applications assigned to a group $\mathbf{G}_k$, $k < j$, have a priority smaller than the priority of application Λ_i . Considering the grouping, we can make the following observation: priority order of applications inside a group can be assigned arbitrarily without jeopardizing the worst-case performance and stability guarantees.

As mentioned before, an intrinsic property of Algorithm 1 is that the priority order of the applications inside a group can be changed arbitrarily. Therefore, our proposed approach optimizes the priority order of the applications within each group with respect to the expected control performance without the need for rechecking the worst-case requirements of applications. Thus, the combinatorial optimization problem is divided into several smaller problems, inside

Algorithm 1 Worst-Case Sensitivity Analysis

```

1: %  $\mathbf{S}$ : remaining applications set;
2: %  $\mathbf{G}_i$ : the  $i$ -th sensitivity group;
3: Initialize set  $\mathbf{S} = \mathbf{\Lambda}$ ;
4: for  $n = 1$  to  $|\mathbf{\Lambda}|$  do
5:    $\mathbf{G}_n = \emptyset$ ;
6:   for all  $\Lambda_i \in \mathbf{S}$  do
7:     • Response-time analysis for sensor and actuator (Equations (3.4) and
       (3.5)), considering  $hp(\Lambda_i) = \mathbf{S} \setminus \{\Lambda_i\}$ ;
8:     • Delay and jitter analyses  $J_{is}, J_{ia}, L_i$  (Equation (3.3));
9:     • Simulation considering  $hp(\Lambda_i) = \mathbf{S} \setminus \{\Lambda_i\}$  to find the expected sensor–
       actuator delay for  $\Lambda_i$ ;
10:    • Control-law synthesis and delay compensation;
11:    • Worst-case control performance  $J_{\Lambda_i}^w$  analysis (Equation (3.2));
12:    if  $J_{\Lambda_i}^w < \bar{J}_{\Lambda_i}^w$  then
13:       $\mathbf{G}_n = \mathbf{G}_n \cup \{\Lambda_i\}$ ;
14:    end if
15:  end for
16:
17:  if  $\mathbf{S} == \emptyset$  then
18:    % Terminate!
19:    return  $\langle \mathbf{G}_1, \mathbf{G}_2, \dots, \mathbf{G}_n \rangle$ ;
20:  else if  $\mathbf{G}_n == \emptyset$  then
21:    % No possible solution meets the requirements!
22:    return  $\langle \rangle$ ;
23:  else
24:     $\mathbf{S} = \mathbf{S} \setminus \mathbf{G}_n$ ;
25:  end if
26: end for

```

each group, which are less time consuming to solve.

The priorities could be assigned considering the dynamics of the control applications. For instance, the bandwidth of a closed-loop control system indicates the speed of system response—the larger the bandwidth, the faster the response. Further, a higher bandwidth implies that the system is more sensitive to a given amount of delay. Analogous to the rate-monotonic priority assignment principle, we hence can assign higher priorities to control applications with larger bandwidth, leading to smaller induced delays due to interference from other applications. In this work we choose to assign the priorities based on the eigenvalues of the closed-loop system, which are proper representatives of the dynamics of the plants.

Table 3.1: Experimental Results

Number of control applications	NO_OPT approach	WST approach	EXP approach	
	Improvement %	Improvement %	Difference %	Percentage of
	$\frac{J_{\text{NO_OPT}}^e - J_{\text{EXP_WST}}^e}{J_{\text{NO_OPT}}^e}$	$\frac{J_{\text{WST}}^e - J_{\text{EXP_WST}}^e}{J_{\text{WST}}^e}$	$\frac{J_{\text{EXP}}^e - J_{\text{EXP_WST}}^e}{J_{\text{EXP}}^e}$	invalid solutions ⁴
	$J_{\text{NO_OPT}}^e$	J_{WST}^e	J_{EXP}^e	
2	74%	10%	-2.4%	42%
3	72%	49%	4.6%	20%
4	44%	25%	-5.4%	44%
5	55%	37%	-0.4%	40%
6	52%	27%	-6.1%	30%
7	58%	33%	4.4%	33%
8	56%	23%	-4.1%	44%
9	54%	28%	-4.9%	44%
10	46%	20%	-4.9%	37%
11	51%	24%	-5.5%	50%
12	43%	13%	-1.6%	44%
13	65%	28%	1.9%	71%
14	46%	24%	-2.4%	60%
15	38%	18%	-5.5%	60%
Average	53%	26%	-2.3%	44%

3.8 Experimental Results

We have performed several experiments to investigate the efficiency of our proposed design approach. We compare our approach (EXP-WST) against three other approaches (NO_OPT, WST, and EXP). We have 125 benchmarks with varying number of plants. The number of plants varies from 2 to 15. The plants are taken from a database with inverted pendulums, ball and beam processes, DC servos, and harmonic oscillators [ÅW97]. Such benchmarks are representative of realistic control problems and are used extensively for experimental evaluations. For each plant, there exists a corresponding control application modeled as a task graph with 2 to 5 tasks. The set of suggested periods of each control applications includes 6 periods generated based on common rules of thumb [ÅW97]. Without loss of generality, we wish to find a high-quality stable design solution, i.e., the constraint on the maximum tolerable worst-case control cost is that the cost is finite (Equation (3.7)).

3.8.1 Efficiency of Our Proposed Approach

As for the first comparison, we run the same algorithm as our proposed approach, however, it terminates as soon as it finds a stable

design solution. Therefore, this approach, called NO_OPT, does not involve any expected control performance optimization but guarantees worst-case stability. We calculate the relative expected control cost improvements $\frac{J_{\text{NO_OPT}}^e - J_{\text{EXP_WST}}^e}{J_{\text{NO_OPT}}^e}$, where $J_{\text{EXP_WST}}^e$ and $J_{\text{NO_OPT}}^e$ are the expected control costs produced by our approach and the NO_OPT approach, respectively. The second column of Table 3.1 shows the result of this set of experiments. As it can be observed, our approach produces solutions with guaranteed stability and an overall control quality improvement of 53% on average, compared to an approach which only addresses worst-case stability.

The second comparison is made with an optimization approach driven by the worst-case control performance. The approach, called WST, is exactly the same as our approach but the objective function to be optimized is the worst-case control cost given in Equation (3.2). Similar to the previous experiment, we are interested in the relative expected control cost improvements $\frac{J_{\text{WST}}^e - J_{\text{EXP_WST}}^e}{J_{\text{WST}}^e}$, where J_{WST}^e is the expected control cost of the final solution obtained by the WST approach. Our proposed approach, while still guarantees worst-case stability, has an average improvement of 26% in the expected control cost as shown in the third column of Table 3.1.

The third comparison is performed against an optimization approach, called EXP, which ONLY takes into consideration the expected control performance. The priority assignment in this approach is done by a genetic algorithm approach similar to our previous work [SCEP09]. Since the worst-case control performance constraints are ignored, the search space is larger, and the algorithm should be able to find a superior design solution in terms of the expected control performance. The comparison has been made considering the relative expected control cost difference $\frac{J_{\text{EXP}}^e - J_{\text{EXP_WST}}^e}{J_{\text{EXP}}^e}$, where J_{EXP}^e is the expected control cost of the final solution found by the EXP approach. The results of the comparison are shown in the forth column of Table 3.1. Since the worst-case control performance constraints are relaxed, the final solution of this approach can turn out to be unstable. The percentage of designs produced by the EXP approach for which the worst-case stability was not guaranteed is reported in the fifth column of Table 3.1. The first observation is that, on average,

⁴Percentage of design solutions produced by the EXP approach for which

Figure 3.5: Runtime of our approach (EXP-WST) and comparison with the relaxed optimization approach (EXP)

for 44% of the benchmarks this algorithm ended up with a design solution for which the stability could not be guaranteed. The second observation is that our approach is on average 2.3% away from the relaxed optimization approach exclusively guided by expected control performance. This clearly states that we are able to guarantee worst-case stability with a very small loss on expected control quality. As discussed before, the relaxed optimization approach should in principle outperform our proposed approach. However, our proposed approach performs slightly better in a few cases which is due to the fact that heuristics with different constraints can be guided to different regions of the huge search space.

3.8.2 Runtime of Our Proposed Approach

We measured the runtime of our proposed approach on a PC with a quad-core CPU running at 2.83 GHz with 8 GB of RAM and Linux operating system. The average runtime of our approach based on the number of control application is shown in Figure 3.5. It can be seen that our approach could find a high-quality stable design solution for large systems (15 control applications) in less than 7 minutes. Also, we report the timing of the relaxed optimization approach, EXP, which only considers the expected control performance (no guarantee

stability is not guaranteed.

for stability), where, as mentioned earlier, the priority assignment is performed by a genetic algorithm. For large systems (15 control applications), it takes 178 minutes for this approach to terminate.

3.9 Conclusions

The design of embedded control systems requires special attention due to the complex interrelated nature of timing properties connecting scheduling and control synthesis. In order to address this problem, not only the control performance but also the robustness requirements of control applications have to be taken into consideration during the design process since having high control performance is not a guarantee for worst-case stability. On the other hand, a design methodology solely grounded on worst-case scenarios leads to poor control performance due to the fact that the design is tuned to a scenario which might occur very rarely. We proposed an integrated design optimization method for embedded control systems with high requirements on robustness and optimized expected control performance. Experimental results have validated that both control quality and worst-case stability of a system have to be taken into account during the design process.

OPTIMAL DESIGN OF STABILIZING CONTROL SERVERS

IN the previous chapter, we discussed that while guaranteeing stability in the worst-case scenario is essential, the main optimization objective in the co-design of embedded control systems should be the expected control performance. In this chapter, we propose exploiting a virtualization framework which facilitates the design and optimization of embedded control systems, while also supporting compositionality and isolation. Note that the contributions in this chapter and the previous chapter are orthogonal and can be combined.

As previously discussed, guaranteeing the stability of control applications in embedded systems is perhaps the most fundamental requirement while implementing such applications. This is different from the classical hard real-time systems where often the acceptance criterion is meeting some deadline. In other words, in the case of control applications, guaranteeing stability is considered to be a main design goal, which is linked to the amount of delay and jitter a control application can tolerate before instability.

In this chapter, the analysis and design of such systems, considering a server-based resource reservation mechanism, are addressed. The benefits of employing a server-based approach are manifold: providing a compositional and scalable framework, protection against other tasks' misbehaviors, and systematic control server design and controller-server co-design. We propose a methodology for designing bandwidth-optimal servers to stabilize control tasks. The pessimism

involved in the proposed methodology is both discussed theoretically and evaluated experimentally.

4.1 Introduction and Related Work

In typical approaches [SLSS96, RS00, CLE⁺04, NPAG06, BC08, ZSWM08, NH09, MSZ11, KGC⁺12, ASE⁺12], the control tasks are all designed together in a way that some global cost (function of the control cost of the individual tasks) is minimized. By following this approach, however, the design of each control task is affected by the other control tasks, which makes the design task all the more challenging. The design, hence, suffers from complexity and lack of compositionality, temporal isolation, and scalability. The proposed approach in the previous chapter follows the same principle.

In this chapter, we propose to use the resource reservation mechanism and run each controller within its own server, which then isolates each control task in the execution environment (see Figure 4.1). The usage of servers for control tasks presents the following advantages:

- it provides compositionality that is essential for systematic system design methodologies;
- the complexity of the design scales linearly with the number of applications;
- it protects each controller from possible misbehaviors, which may occur within other tasks and then possibly jeopardize the entire system;
- the bandwidth assignment, rather than the priority assignment, may constitute a more accurate instrument to allocate the available computing resources;
- the simple interface provided by the resource reservation mechanism facilitates the controller–server co-design process [CE05], [ABEP14];
- running the controller over a dedicated server, may reduce significantly the jitter of the controller, especially if the server period is smaller than the period of the controller. In short, this

Figure 4.1: Overview of the proposed approach.

is due to the fact that the server guarantees the control task a certain resource bandwidth. This is important since it is often possible to compensate for the constant part of delay, while the process of coping with the jitter is more involved.

Over the past decade, the analysis and design of real-time servers have widely developed. Feng and Mok [FM02] introduced the *bounded delay resource* model to facilitate hierarchical resource sharing. The schedulability analysis and server design problems for real-time applications under the *periodic resource* model have been addressed by [SRLK02, LB03, SL03, AP04]. Easwaran et al. [EAL07] extended the periodic resource model to the *explicit deadline periodic* model (EDP). Similar to what we do in this chapter, Fisher and Dewan [FD12] described a method to minimize the bandwidth of a server. They developed a fully-polynomial-time approximation scheme (FPTAS) to solve the problem. However, as the majority of the work in this area, they considered the task deadlines as constraints rather than the stability of the controllers.

More relevant to this work, Cervin and Eker [CE05] proposed the control server approach which provides a simple interface used for control-scheduling co-design of real-time systems. Fontanelli et al. [FPG13] addressed the problem of bandwidth allocation for a set of control tasks, assuming a fluid execution model and monotonicity of the considered cost function.

Recently, Fontanelli et al. proposed a new model for real-time control applications [FPA13] to investigate stochastic stability, but ignoring the dependencies among stochastic variables. In [ABEP14], we extend the work in this chapter towards a different direction, presenting a controller-server co-design approach where the controller is determined in a unified process along with the server parameters.

While the analysis and design problems of real-time servers have been discussed to a considerable degree, the server-based approach has gained less attention in the case of control applications which are fundamentally different from real-time applications with hard deadlines. In particular, as opposed to hard real-time applications, the notion of deadline is considered to be artificial for control applications. In contrast to hard real-time systems, control stability is the main property to be guaranteed for control applications. Therefore, in the case of control applications, worst-case control performance and stability should be considered instead of worst-case response time and deadline.

To approach the problem of designing stabilizing servers, the first step is to capture the stability of the controllers in terms of real-time parameters, which is facilitated by the Jitter Margin toolbox [KL04], [CLE⁺04], [Cer12]. The stability of control applications, hence, depends not only on the amount of latency, but also on the amount of jitter the application experiences [WNT95]. The second step is to derive analysis methods for the servers to compute the discussed real-time metrics, i.e., latency and jitter. To this end, we consider the explicit deadline periodic model and develop the worst-case and best-case response times for tasks with arbitrary deadlines within explicit deadline periodic servers with arbitrary deadlines. Having the worst-case and best-case response times, it is, then, possible to compute the latency and jitter and investigate if a control application within a given server is guaranteed to be stable.

In addition to the analysis, we also provide analytic results that can drive the design of a server towards solutions which can guarantee the stability of the controller. The aim of such a design procedure is bandwidth minimization. Since such a solution is derived using a linear upper and lower bound of the server supply function, we also evaluate the amount of pessimism introduced by our technique, both theoretically and experimentally.

Figure 4.2: Graphical interpretation of the latency and worst-case response-time jitter.

4.2 System Model

The system is composed of n plants. Each plant is controlled by a control task which is executing within a server. Below we describe the model of the plant and the control task. Note that since each plant is considered in isolation, we do not report the index i of the control application among all the control applications.

Throughout this chapter, we assume that the controller is given. The plant output is sampled in a strictly periodic manner with period h . The control signal is computed by a control task τ . Such a control signal is updated any time the control task completes and is held constant between two consecutive updates. The instants when the input is applied to the plant do then depend on the way the task τ is scheduled. The task parameters, are the best-case execution time c^b , the worst-case execution time c^w , and sampling period h .

In addition, the task scheduling process determines the best-case response time R^b , the worst-case response time R^w , the latency $L = R^b$, and the worst-case response-time jitter (jitter) $J = R^w - R^b$.

The terminology and the notation are illustrated in Figure 4.2 (reproduced, for convenience, from Chapter 2). The latency captures the constant part of the delay, while the jitter corresponds to the variation in the delay experienced by all instances (jobs) of a task. Note that we do not consider any deadlines for control tasks.

4.3 Server Model

As discussed above, to isolate controllers from one another, each control task is bound to execute over a dedicated server. The periodic server S is described by:

- the *server budget* Q ;

(a) Worst-case resource allocation scenario. (b) Best-case resource allocation scenario.

Figure 4.3: Worst-case and best-case resource allocation scenarios.

- the *server period* P , and
- the *server deadline* D .

This model was also called EDP (Explicit Deadline Periodic) model [EAL07]. Every period P the server is activated. Then, it allocates Q amount of time to the task, before the server deadline expires.

The latency and jitter experienced by a task are tightly connected to the best-case and worst-case response times. To compute these two quantities, it is then necessary to determine the worst and best case scenarios with regard to the computational resource supplied by the server.

To perform worst-case analysis for the tasks running within a server, a classic approach [FM02, LB03, SL03, AP04, EAL07] is to define the *supply lower bound function* $\text{slbf}(t)$, which is formally defined as follows.

Definition 4.1 *The supply lower bound function $\text{slbf}(t)$ of a server S is the minimum amount of resource provided in any interval of length t .*

The exact expression of $\text{slbf}(t)$ of a periodic server, is

$$\text{slbf}(t) = \max\{0, kQ, t - P - D + 2Q - k(P - Q)\} \quad (4.1)$$

with $k = \left\lfloor \frac{t-(D-Q)}{P} \right\rfloor$, and it is depicted in Figure 4.3(a) by a solid line (please refer to the related literature [FM02, LB03, SL03, AP04, EAL07] for details on its computation). As the expression of (4.1) may be difficult to be managed, especially when the server parameters are the variables subject to optimization (as we do here), it is often

convenient to lower bound the $\text{slbf}(t)$ by the *linear supply lower bound function* $\text{lslbf}(t)$, defined as

$$\text{lslbf}(t) = \max \{0, \alpha(t - \Delta)\}, \quad (4.2)$$

with, using Feng-Mok's notation [FM02], the server bandwidth α and delay Δ , defined as

$$\alpha = \frac{Q}{P}, \quad (4.3)$$

$$\Delta = P + D - 2Q. \quad (4.4)$$

The lslbf is depicted in Figure 4.3(a) by a dashed line.

Analogously, for the best-case analysis it is possible to compute the *supply upper bound function* $\text{subf}(t)$, defined as follows.

Definition 4.2 *The supply upper bound function $\text{subf}(t)$ of a server S is the maximum amount of resource provided in any interval of length t .*

In strict analogy to the worst case examined earlier, the expression of the subf of a periodic server is

$$\text{subf}(t) = \min\{t, kQ, t + P + D - 2Q - k(P - Q)\} \quad (4.5)$$

with $k = \left\lceil \frac{t+D-Q}{P} \right\rceil$, while the *linear supply upper bound function* is

$$\text{lsubf}(t) = \min \{t, \alpha(t + \Delta)\} \quad (4.6)$$

with α and Δ as in (4.3) and (4.4), respectively. Figure 4.3(b) shows the subf (by a solid line) as well as the lsubf (by a dashed line).

4.4 Server-Based Analysis of Control Tasks

In this section, we determine the best-case and worst-case response times of the control task running within a server, as functions of the server parameters P , Q , and D . The analysis is performed with the exact slbf/subf functions of (4.1) and (4.5) (in Section 4.4.1) as well as with the linear bounds $\text{lslbf}/\text{lsubf}$ of (4.2) and (4.6) (Section 4.4.2).

4.4.1 Exact Characterization

In this section, the exact real-time analysis for a control task is derived. To derive the worst-case response time of a task τ , we must consider the minimum amount of resource time available to the task, which is described by $\text{slbf}(t)$.

The worst-case response time R^w of the first job of the control task (released at time 0) is equal to the first instant when the server has necessarily provided at least c^w amount of time, that is

$$R^w = \min \{t : \text{slbf}(t) \geq c^w\}. \quad (4.7)$$

By computing the pseudo-inverse of $\text{slbf}(t)$, such a value can be computed explicitly and it is equal to

$$R^w = D - Q + \left\lceil \frac{c^w}{Q} \right\rceil (P - Q) + c^w. \quad (4.8)$$

The proof is similar to [BB06].

Unfortunately, the longest response time may occur even at the later jobs, and not necessarily at the first job. This is the case since, as mentioned before, we do not enforce any task deadline, thus, response times are allowed to be longer than the sampling periods h . Therefore, we must evaluate the response times of all jobs within the busy period, as indicated by Lehoczky [Leh90] for the arbitrary deadline case.

The worst-case response time of the control task within a server $S = (Q, P, D)$ is obtained as follows,

$$R^w = \sup_{q \in \mathbb{N}} \left\{ D - Q + \left\lceil \frac{qc^w}{Q} \right\rceil (P - Q) + qc^w - (q - 1)h \right\}. \quad (4.9)$$

We remind that (for example, see the proof of Lemma 1 in [BHCNRB09]) the supremum of (4.9) has a finite solution only when

$$\alpha = \frac{Q}{P} \geq \frac{c^w}{h}. \quad (4.10)$$

In analogy with (4.7), the best-case response time R^b is defined through the subf function as follow

$$R^b = \min \{t : \text{subf}(t) \geq c^b\}, \quad (4.11)$$

which can also be computed explicitly, and it is equal to

$$R^b = \max \left\{ 0, 2Q - D - P + \left\lceil \frac{c^b}{Q} \right\rceil (P - Q) \right\} + c^b. \quad (4.12)$$

The proof is similar to the proof of Theorem 1 in [LB03].

4.4.2 Characterization with Linear Bounds

The main obstacle in using the exact response time for finding the optimal server parameters (see Section 4.6) is that Equations (4.9) and (4.12) involve ceiling functions. Hence, we propose to compute an upper bound $\bar{R}^w$ to the R^w and a lower bound $\underline{R}^b$ of R^b using, respectively, the `slbf` and `lsbf` functions, rather than the exact ones, i.e., `slbf` and `subf`.

Observe that while this approximation involves pessimism, it is safe from the stability point of view.

By replacing the `slbf` in (4.7) with the `lsbf` of (4.2), we can readily compute the response time upper bound, which is

$$\bar{R}^w = \frac{c^w}{\alpha} + \Delta. \quad (4.13)$$

As shown in [BHCNRB09], such an upper bound to the response time is finite only if the server bandwidth is not smaller than the worst-case utilization of the control task, that is

$$\alpha = \frac{Q}{P} \geq \frac{c^w}{h}. \quad (4.14)$$

Similarly, by replacing `subf` in (4.11) by `lsbf` of (4.6), the lower bound to the best-case response time is given by,

$$\underline{R}^b = \max \left\{ c^b, \frac{c^b}{\alpha} - \Delta \right\}. \quad (4.15)$$

4.5 Stability Constraint

In Chapter 2 it has been discussed that the latency and jitter in the execution of the control applications are decisive factors in the performance and stability of the plants associated with them. To quantify the tolerable amount of latency and jitter by a control application before the instability of the plant, or to guarantee a certain degree of performance, we use the Jitter Margin toolbox [KL04], [CLE⁺04],

[Cer12]. It provides sufficient stability conditions for a closed-loop system with a linear continuous-time plant and a linear discrete-time controller.

As discussed in Section 2.3, for a given sampling period, the stability curve can safely be approximated by a linear function of the latency and worst-case response-time jitter. In this case, the linear *stability condition* for a control application is of the form

$$L + aJ \leq b, \quad (4.16)$$

where $a \geq 1, b \geq 0$. The latency L identifies the constant part of the delay that the control application experiences, whereas the worst-case response-time jitter J captures the varying part of the delay (see Figure 4.2, where R^b and R^w represent the best-case and worst-case response times, respectively).

In order to apply the stability analysis discussed, the values of the latency (L) and worst-case response-time jitter (J) of the control task should be computed. The two metrics are defined based on the worst-case and best-case response times as follows,

$$\begin{aligned} L &= R^b, \\ J &= R^w - R^b, \end{aligned} \quad (4.17)$$

where R^w and R^b denote the worst-case and best-case response times, respectively. The stability constraint, hence, can be formulated as,

$$\begin{aligned} L + aJ &\leq b, \\ R^b + a(R^w - R^b) &\leq b. \end{aligned} \quad (4.18)$$

For a given server, the stability condition (4.18), which is based on the exact best-case and worst-case response times, determines if the server, in the worst-case, can guarantee the stability of the control task associated with it (analysis problem).

In the context of the optimization problem, as will be discussed in Section 4.6, however, the presence of discontinuous operators (ceiling) in the exact expressions (4.9) and (4.12) of the worst-case and best-case response times makes them unsuitable. Hence, we use the upper/lower bound of the worst/best-case response times and redefine the latency and the worst-case response-time jitter as follows,

$$\begin{aligned} \underline{L} &= \underline{R}^b, \\ \bar{J} &= \bar{R}^w - \underline{R}^b. \end{aligned} \quad (4.19)$$

While using the linear supply bounds involves some pessimism compared to the original supply bounds, it is safe from the stability point of view. Nonetheless, the amount of introduced pessimism is discussed in Section 4.7.

The stability constraint based on the linear bounds is given in the following,

$$\begin{aligned} b &\geq \underline{L} + a\bar{J}, \\ b &\geq \underline{R}^b + a(\bar{R}^w - \underline{R}^b), \\ b &\geq a\left(\frac{c^w}{\alpha} + \Delta\right) - (a-1) \max\left\{c^b, \frac{c^b}{\alpha} - \Delta\right\}, \\ &= a\left(\frac{c^w}{\alpha} + \Delta\right) + (a-1) \min\left\{-c^b, -\left(\frac{c^b}{\alpha} - \Delta\right)\right\}, \end{aligned}$$

which we rewrite as

$$\min\left\{\frac{a(c^w - c^b) + c^b}{\alpha} + (2a-1)\Delta - b, \frac{ac^w}{\alpha} + a\Delta - (a-1)c^b - b\right\} \leq 0. \quad (4.20)$$

Hence, Equation (4.20) describes the constraint on the server parameters (the bandwidth α and the delay Δ , see Section 4.3), which guarantees the stability of the controller running within such a server.

4.6 Optimal Design of Stabilizing Servers

In this section, we describe the procedure to design optimal stabilizing servers. The objective of the optimization is to minimize the utilization required in order to guarantee the stability of all control applications, that is

$$U = \sum_{i=1}^n \left(\alpha_i + \frac{\epsilon}{P_i}\right), \quad (4.21)$$

where ϵ denotes the switching overhead for the server and is considered to be strictly positive. If no overhead is considered, then the solution would be with $P \rightarrow 0$, making this an impractical server period.

We consider the implicit deadline server design, in which all server deadlines are set equal to the periods, $D_i = P_i$.

Thanks to the isolation provided by the resource allocation mechanism, the stability of each control task is guaranteed through the parameters (α and Δ) of the server running the task only (Equation (4.20)). Hence, the minimization of the total server utilization of (4.21) can be broken down into one bandwidth minimization problem for each server, rather than a more complex minimization which involves all task parameters together.

If we assume $D = P$ for all servers, we can perform the following optimization for each control application and conclude based on the obtained results,

$$\begin{aligned} \min_{\alpha, \Delta} \quad & \alpha + \frac{2\epsilon(1-\alpha)}{\Delta} \\ \text{s.t.} \quad & \min \left\{ \frac{a(c^w - c^b) + c^b}{\alpha} + (2a-1)\Delta - b, \right. \\ & \left. \frac{ac^w}{\alpha} + a\Delta - (a-1)c^b - b \right\} \leq 0. \end{aligned} \quad (4.22)$$

Notice that in the above cost the period P is replaced by $\frac{\Delta}{2(1-\alpha)}$, as it follows from (4.3)–(4.4) for $D = P$.

The solution to the above problem is the minimum bandwidth (including the overhead) required to guarantee stability of control task τ .

Let us proceed with finding the global optimum of the problem (4.22), which is concerned with a single control task in isolation. The stability constraint in (4.22) can be written as

$$\min\{g_I(\alpha, \Delta), g_{II}(\alpha, \Delta)\} \leq 0,$$

which is equivalent to

$$(g_I(\alpha, \Delta) \leq 0) \quad \vee \quad (g_{II}(\alpha, \Delta) \leq 0),$$

with $\vee$ denoting the *logical or* between two propositions. Thus, the problem (4.22) can be solved by solving individually the following two problems

$$\begin{aligned} \min_{\alpha, \Delta} \quad & \alpha + \frac{2\epsilon(1-\alpha)}{\Delta} \\ \text{s.t.} \quad & \frac{a(c^w - c^b) + c^b}{\alpha} + (2a-1)\Delta - b \leq 0, \end{aligned} \quad (4.23)$$

and,

$$\begin{aligned} \min_{\alpha, \Delta} \quad & \alpha + \frac{2\epsilon(1-\alpha)}{\Delta} \\ \text{s.t.} \quad & \frac{ac^w}{\alpha} + a\Delta - (a-1)c^b - b \leq 0. \end{aligned} \quad (4.24)$$

and then select the best solution between the two produced by (4.23) and (4.24). Moreover, in order for the response time to be finite, the server bandwidth α has to satisfy $\alpha \geq \frac{c^w}{h}$, which leads to an additional constraint in each of the problems (4.23) and (4.24).

To solve problems (4.23) and (4.24), we use the KKT (Karush-Kuhn-Tucker) necessary conditions for optimality [BSS06]. According to the KKT condition, the optimum $\mathbf{x}^*$ of the problem

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0, \quad i = 1 \dots m, \end{aligned} \quad (4.25)$$

must necessarily satisfy the following conditions

$$\begin{aligned} \nabla f(\mathbf{x}^*) + \sum_{i=1}^m \mu_i^* \nabla g_i(\mathbf{x}^*) &= \mathbf{0}, \\ \mu_i^* g_i(\mathbf{x}^*) &= 0, \quad i = 1 \dots m, \\ \mu_i^* &\geq 0, \quad i = 1 \dots m. \end{aligned} \quad (4.26)$$

For the case of our problem, it is assumed that $g_1(\mathbf{x})$ is associated with the stability constraints shown in problems (4.23) and (4.24), whereas $g_2(\mathbf{x})$ is associated with inequality (4.10).

Let us proceed with solving problem (4.23). From the KKT condition of the gradient, if we differentiate w.r.t. α and then Δ , we find

$$1 - \frac{2\epsilon}{\Delta} - \mu_1 \frac{a(c^w - c^b) + c^b}{\alpha^2} - \mu_2 = 0 \quad (4.27)$$

$$-\frac{2\epsilon(1-\alpha)}{\Delta^2} + \mu_1(2a-1) = 0 \quad (4.28)$$

We consider two cases: $\mu_2 = 0$ and $\mu_2 > 0$.

$\mu_2 = 0$: Let us first assume there is no constraint on the server bandwidth α , i.e., $\mu_2 = 0$. Since $a \geq 1$ and $\alpha < 1$, from (4.28), we immediately find the multiplier μ_1 , that is:

$$\mu_1 = \frac{2\epsilon(1-\alpha)}{\Delta^2(2a-1)} > 0,$$

hence the constraint of (4.23) is active and must hold with the equal sign.

If we set, to have a more compact notation,

$$x_1 = a(c^w - c^b) + c^b, \quad y_1 = \epsilon(2a - 1), \quad z_1 = b, \quad (4.29)$$

then the equality constraint of (4.23) can be rewritten as

$$\frac{x_1}{\alpha} + y_1 \frac{\Delta}{\epsilon} = z_1, \quad (4.30)$$

from which we find

$$\frac{\Delta}{\epsilon} = \frac{\alpha z_1 - x_1}{\alpha y_1}, \quad (4.31)$$

and then the multiplier μ_1 is

$$\mu_1 = \frac{2(1 - \alpha)}{y_1} \left(\frac{\alpha y_1}{\alpha z_1 - x_1} \right)^2. \quad (4.32)$$

By replacing (4.31) and (4.32) in the condition (4.27), we find:

$$\begin{aligned} 1 - 2 \frac{\alpha y_1}{\alpha z_1 - x_1} - \frac{2(1 - \alpha)}{y_1} \frac{\alpha^2 y_1^2}{(\alpha z_1 - x_1)^2} \frac{x_1}{\alpha^2} &= 0 \\ z_1(z_1 - 2y_1)\alpha^2 - 2x_1(z_1 - 2y_1)\alpha + x_1(x_1 - 2y_1) &= 0 \\ \alpha^2 - 2 \frac{x_1}{z_1} \alpha + \frac{x_1(x_1 - 2y_1)}{z_1(z_1 - 2y_1)} &= 0 \\ \alpha &= \alpha_1 (1 \pm \delta_1) \end{aligned}$$

where we set

$$\alpha_\ell = \frac{x_\ell}{z_\ell}, \quad \delta_\ell = \sqrt{1 - \frac{z_\ell(x_\ell - 2y_\ell)}{x_\ell(z_\ell - 2y_\ell)}} \quad (4.33)$$

with $\ell = 1$. The values α_1 and δ_1 represent, respectively, the consumed bandwidth in absence of overhead and the increase of bandwidth needed due to overhead. Among the two solutions, the smaller one makes the corresponding value of Δ negative. Hence, the only acceptable solution for the server bandwidth is given by $\alpha_1(1 + \delta_1)$ and there is no need to check the second-order sufficient conditions.

The solution identified here corresponds to the case where there is no constraint on server bandwidth α (i.e., $\mu_2 = 0$). Therefore, if this solution satisfies constraint (4.10), i.e., $\alpha \geq \frac{c^w}{h}$, then there is no need to consider the case where $\mu_2 > 0$ (because the solution in the larger

search space is valid even considering this constraint); otherwise, this case has to be taken into account.

$\mu_2 > 0$: Now let us consider the case where the solution found does not satisfy constraint (4.10), i.e., $\alpha < \frac{c^w}{h}$. From Equation (4.27), we have,

$$\mu_2 = 1 - \frac{2\epsilon}{\Delta} - \mu_1 \frac{a(c^w - c^b) + c^b}{\alpha^2}. \quad (4.34)$$

Substituting μ_2 in the second equality of (4.26), i.e., $\mu_2 g_2(\mathbf{x}) = \mu_2 \left(-\alpha + \frac{c^w}{h}\right) = 0$, we obtain,

$$\underbrace{\left(1 - \frac{2\epsilon}{\Delta} - \mu_1 \frac{a(c^w - c^b) + c^b}{\alpha^2}\right)}_{\text{first term}} \underbrace{\left(-\alpha + \frac{c^w}{h}\right)}_{\text{second term}} = 0. \quad (4.35)$$

The above equation has at most three solutions. The solution found so far is equivalent to considering the first term to be equal to zero, i.e., $\mu_2 = 0$ (see Equation (4.34)). As discussed before, the solutions obtained considering the first term are invalid since they do not satisfy constraint (4.10). Therefore, the only valid solution is $\frac{c^w}{h}$. In other words, the optimal solution α_1^* is equal to $\alpha_1(1 + \delta_1)$ except when this solution does not satisfy constraint (4.10). The final solution is then given by,

$$\alpha_1^* = \max \left\{ \alpha_1(1 + \delta_1), \frac{c^w}{h} \right\}, \quad (4.36)$$

in which we also account for the constraint (4.10). The corresponding optimal value of the server delay Δ_1^* can be computed from (4.31).

To solve the second problem (4.24), we simply observe that by setting

$$x_{II} = ac^w, \quad y_{II} = a\epsilon, \quad z_{II} = b + (a - 1)c^b. \quad (4.37)$$

the constraint can be rewritten as in (4.30) by replacing x_I , y_I , and z_I , with x_{II} , y_{II} , and z_{II} of (4.37). Since the cost functions and the constraints of the two problems are the same, it follows that the solution is exactly the same as (4.36), with the corresponding replacements.

Since the two problems have to be considered in logical or, the minimal bandwidth α^* and delay Δ^* which can guarantee the stability of the control task (within the assumption of server deadline D equal

to the server period P) is given by the better solution of the two problems, i.e.,

$$\min \left\{ \alpha_I^* + \frac{2\epsilon(1 - \alpha_I^*)}{\Delta_I^*}, \alpha_{II}^* + \frac{2\epsilon(1 - \alpha_{II}^*)}{\Delta_{II}^*} \right\}. \quad (4.38)$$

After performing the above procedure for all servers and having found the minimum resource utilization required for stability of all control applications, we should now check if the resource demand is less than or equal to the resource supply. In the case of the implicit deadline servers running on a uniprocessor, the solution found is valid if and only if the utilization is less than or equal to one, i.e.,

$$\sum_{i=1}^n \left(\alpha_i^* + \frac{2\epsilon(1 - \alpha_i^*)}{\Delta_i^*} \right) \leq 1. \quad (4.39)$$

If condition (4.39) is not satisfied, there is no solution, with the given assumptions, that guarantees the stability of all control applications.

4.7 Theoretical Guarantees

We shall now discuss the degree of pessimism introduced in the proposed approach by using the linear bounds instead of the exact response times. Towards this, we need to define the notion of optimistic supply function. Note that the optimistic supply functions are *unsafe* and are only used to quantify the amount of pessimism introduced in our approach.

The *optimistic supply lower bound function* $\text{olslbf}(t)$ of a server is a linear upper bound on the supply lower bound function (as shown by the dotted line in Figure 4.3(a)). To obtain this, we notice that the optimistic supply functions are only $\frac{Q}{P}(P - Q)$ different from the linear supply bound functions, as it is shown in Figure 4.4. Since we have the linear supply lower bound function $\text{slbf}(t)$, the optimistic supply lower bound function is obtained as follows,

$$\begin{aligned} \text{olslbf}(t) &= \max \left\{ 0, \frac{Q}{P}(t - (P + D - 2Q)) + \frac{Q}{P}(P - Q) \right\} \\ &= \max \left\{ 0, \frac{Q}{P}(t - (D - Q)) \right\}. \end{aligned}$$

Figure 4.4: The relation between linear, optimistic, and exact supply functions.

Similarly, we define the *optimistic supply upper bound function* $\text{olsubf}(t)$ as follows (shown by the dotted line in Figure 4.3(b)),

$$\begin{aligned} \text{olsubf}(t) &= \min \left\{ t, \frac{Q}{P}(t + (P + D - 2Q)) - \frac{Q}{P}(P - Q) \right\} \\ &= \min \left\{ t, \frac{Q}{P}(t + (D - Q)) \right\}. \end{aligned}$$

Computing the pseudo-inverse of these optimistic supply functions, we obtain optimistic bounds for the best-case and worst-case response times,

$$\begin{aligned} \underline{R}^w &= \frac{c^w}{\alpha} + \underline{\Delta}, \\ \overline{R}^b &= \max \left\{ c^b, \frac{c^b}{\alpha} - \underline{\Delta} \right\}, \end{aligned} \tag{4.40}$$

where $\underline{\Delta} = D - Q$.

The next two subsections discuss the theoretical results on the amount of pessimism involved in our design method. We shall first restrict our attention to the stability of one single controller. Then, we focus on both stability and schedulability of the set of all servers.

4.7.1 Stability of Controllers

In this subsection, we shall focus on the stability of a single controller. The following lemma clarifies the relation between using the optimistic and exact supply functions for stability guarantees.

Lemma 4.7.1 *If the stability constraint (4.20) of a control task is satisfied within a server $S = (Q, P, D)$ with the exact supply functions, it is also satisfied within the same server, but considering the optimistic supply functions.*

Proof: From the definition of the optimistic supply functions,

$$\begin{aligned}\forall t, \quad \text{subf}(t) &\geq \text{olsubf}(t), \\ \forall t, \quad \text{slbf}(t) &\leq \text{olslbf}(t).\end{aligned}$$

This, in turn, leads to the following inequalities among the exact and optimistic response times,

$$\begin{aligned}R^b &= \min\{t : \text{subf}(t) \geq c^b\} \leq \min\{t : \text{olsubf}(t) \geq c^b\} = \bar{R}^b, \\ R^w &= \min\{t : \text{slbf}(t) \geq c^w\} \geq \min\{t : \text{olslbf}(t) \geq c^w\} = \underline{R}^w.\end{aligned}$$

This indicates that considering the optimistic supply functions results in a lower bound for the worst-case response time $\underline{R}^w$ and an upper bound for the best-case response time $\bar{R}^b$. Note that since $a \geq 1$, $\underline{R}^w \leq R^w$, $\bar{R}^b \geq R^b$, we have,

$$L + aJ = aR^w + (1 - a)R^b \geq a\underline{R}^w + (1 - a)\bar{R}^b = \bar{L} + a\underline{J},$$

where $\bar{L} = \bar{R}^b$ and $\underline{J} = \underline{R}^w - \bar{R}^b$. Hence, if there exists a server with the exact supply functions that can satisfy inequality (4.18), then the stability constraint (4.18) is also satisfied considering the optimistic linear supply functions, i.e.,

$$L + aJ \leq b \xrightarrow{L+aJ \geq \bar{L}+a\underline{J}} \bar{L} + a\underline{J} \leq b.$$

□

Lemma 4.7.2 *If the stability constraint (4.20) of a control task is satisfied within a server $S = (Q, P, D)$ with the linear supply functions, it is also satisfied within the same server, but considering the exact supply functions.*

Proof: The following relations hold for the response times,

$$\begin{aligned}R^b &= \min\{t : \text{subf}(t) \geq c^b\} \geq \min\{t : \text{lsubf}(t) \geq c^b\} = \underline{R}^b, \\ R^w &= \min\{t : \text{slbf}(t) \geq c^w\} \leq \min\{t : \text{lsbf}(t) \geq c^w\} = \bar{R}^w.\end{aligned}$$

Figure 4.5: Worst-case and best-case resource allocation scenarios for implicit deadline server.

Since $a \geq 1$, we have the following inequalities,

$$\begin{aligned} aR^w + (1-a)R^b &\leq a\bar{R}^w + (1-a)\underline{R}^b, \\ L + aJ &\leq \underline{L} + a\bar{J}, \end{aligned}$$

from which the theorem follows,

$$\underline{L} + a\bar{J} \leq b \quad \xRightarrow{L+aJ \leq \underline{L}+a\bar{J}} \quad L + aJ \leq b.$$

□

The next theorem establishes one important relation between stabilizing controllers based on the linear supply functions and the exact supply functions.

Theorem 4.7.3 *If the stability constraint (4.20) of a control task is satisfied within a server $S = (Q, P, D)$ with the exact supply functions, it is also satisfied within a server $S' = (\frac{Q}{k}, \frac{P}{k}, \frac{D}{k})$ with the linear supply functions and $k \geq 1 + \frac{P-Q}{D-Q}$.*

Proof: Let us first prove the following inequalities,

$$\begin{aligned} \forall t, \quad \text{subf}(t) &\geq \text{lsbf}'(t), \\ \forall t, \quad \text{slbf}(t) &\leq \text{slbf}'(t). \end{aligned} \tag{4.41}$$

To prove $\text{subf}(t) \geq \text{lsbf}'(t)$, we derive the optimistic linear lower bound $\text{olsbf}(t)$ on the exact supply upper bound function $\text{subf}(t)$. If we can prove that this optimistic linear lower bound $\text{olsbf}(t)$ is

always greater than or equal to $\text{lsubf}'(t)$, considering that it is a lower bound of $\text{subf}(t)$, we have $\text{subf}(t) \geq \text{lsubf}'(t)$. The linear lower bound on $\text{subf}(t)$ is given by (according to the definition of the optimistic supply upper bound functions),

$$\text{olsubf}(t) = \min \left\{ t, \frac{Q}{P}(t + (D - Q)) \right\}.$$

Let us also derive the $\text{lsubf}'(t)$ for the implicit deadline server $S' = (\frac{Q}{k}, \frac{P}{k}, \frac{D}{k})$,

$$\text{lsubf}'(t) = \min \left\{ t, \frac{Q}{\frac{P}{k}}(t + \frac{1}{k}(P + D - 2Q)) \right\}.$$

Our goal is to find k such that the optimistic linear lower bound on the exact supply upper bound function $\text{olsubf}(t)$ is always above the linear upper bound $\text{lsubf}'(t)$. Note that, if $x \geq y$ and $x' \geq y'$, then $\min\{x, x'\} \geq \min\{y, y'\}$. Since $t \geq t$, we only need to focus on the second terms inside the min-functions in $\text{olsubf}(t)$ and $\text{lsubf}'(t)$, i.e.,

$$\frac{Q}{P}(t + (D - Q)) \geq \frac{Q}{\frac{P}{k}}(t + \frac{1}{k}(P + D - 2Q)).$$

Re-arranging the terms, we obtain,

$$k \geq 1 + \frac{P - Q}{D - Q}.$$

Analogously, to prove $\text{slbf}(t) \leq \text{slbf}'(t)$, we start by showing that the optimistic linear upper bound on $\text{slbf}(t)$ is given by (according to the definition of the optimistic supply lower bound functions),

$$\text{olslbf}(t) = \max \left\{ 0, \frac{Q}{P}(t - (D - Q)) \right\}.$$

The linear lower bound on server S' is given by,

$$\text{slbf}'(t) = \max \left\{ 0, \frac{Q}{\frac{P}{k}}(t - \frac{1}{k}(P + D - 2Q)) \right\}.$$

Observe that, if $x \leq y$ and $x' \leq y'$, then $\max\{x, x'\} \leq \max\{y, y'\}$. Since $0 \leq 0$, it is enough to focus on the second terms inside the max-functions. We would like to find k such that the optimistic linear

upper bound on $\text{slbf}(t)$, denoted by $\text{olslbf}(t)$, is always below $\text{slbf}'(t)$, i.e.,

$$\frac{Q}{P}(t - (D - Q)) \leq \frac{\frac{Q}{k}}{\frac{P}{k}}(t - \frac{1}{k}(P + D - 2Q)),$$

which simplifies to,

$$k \geq 1 + \frac{P - Q}{D - Q},$$

which is the same as the previous condition on k .

As a result of the inequalities in (4.41), the following relations hold for the response times,

$$\begin{aligned} R^b &= \min\{t : \text{subf}(t) \geq c^b\} \leq \min\{t : \text{lsbf}'(t) \geq c^b\} = \underline{R}^{b'}, \\ R^w &= \min\{t : \text{slbf}(t) \geq c^w\} \geq \min\{t : \text{slbf}'(t) \geq c^w\} = \overline{R}^{w'}. \end{aligned}$$

Since $a \geq 1$, we have the following inequalities,

$$\begin{aligned} aR^w + (1 - a)R^b &\geq a\overline{R}^{w'} + (1 - a)\underline{R}^{b'}, \\ L + aJ &\geq \underline{L}' + a\overline{J}', \end{aligned}$$

from which the theorem follows,

$$L + aJ \leq b \quad \xLeftrightarrow{L+aJ \geq \underline{L}' + a\overline{J}'} \quad \underline{L}' + a\overline{J}' \leq b.$$

Note that the bound is tight when $k = 1 + \frac{P-Q}{D-Q}$ since the linear lower bound on $\text{subf}(t)$ is the same as $\text{lsbf}'(t)$ and the linear upper bound on $\text{slbf}(t)$ is the same as $\text{slbf}'(t)$. The tightness is in the sense that, for server $S' = (\frac{Q}{k}, \frac{P}{k}, \frac{D}{k})$, decreasing k by a small positive value, violates the inequalities in (4.41). $\square$

The important message of Theorem 4.7.3 is that, if a server $S = (Q, P, D)$ (with the exact supply functions) with bandwidth $\alpha = \frac{Q}{P}$ is identified that satisfies the stability constraint (4.18) for the control task associated with it, then there exists a server $S' = (\frac{Q}{k}, \frac{P}{k}, \frac{D}{k})$ (with the linear supply functions) that can also satisfy the stability constraint (4.18) for the control task *and the required bandwidth is the same*, i.e., $\alpha' = \frac{\frac{Q}{k}}{\frac{P}{k}} = \frac{Q}{P}$.

The theorem also states that, *in the worst-case*, the server S' has to be run k times more frequently compared to S . In practice, of

course, this might be a disadvantage, if the context-switch overhead is significant.

The following corollary discusses the particular case of implicit deadline servers (i.e., $D = P$) and it will be used in the next section.

Corollary 4.7.3.1 *If the stability constraint (4.20) of a control task is satisfied within an implicit deadline server $S = (Q, P)$ with the exact supply functions, it is also satisfied within an implicit deadline server $S' = (\frac{Q}{2}, \frac{P}{2})$ with the linear supply function.*

Proof: The proof follows by substituting $D = P$ in $k \geq 1 + \frac{P-Q}{D-Q}$ in Theorem 4.7.3, i.e., $k \geq 2$. $\square$

For clarification see Figure 4.5. In Figure 4.5(a), the linear supply lower bound function lsbf' is both a lower bound for the exact supply lower bound function slbf' and the optimistic upper bound on slbf , i.e., olsbf . This implies that the amount of resource provided by slbf' in the worst case is more than or equal to the amount of resource provided by slbf in the worst case. Similarly, Figure 4.5(b) shows that the linear supply upper bound function lsubf' is both an upper bound for the exact supply upper bound function subf' and the optimistic lower bound of subf , olsubf .

The bound is tight in the sense that the linear supply lower bound functions lsbf' is not only the tightest linear lower bound on the exact supply lower bound function slbf' , but also the tightest linear upper bound on the exact supply lower bound function slbf . Similar results can be derived for the linear supply upper bound function lsubf' .

4.7.2 Schedulability of Servers

Thus far in this section, we have limited our attention only to the stability of a single server. However, for a system to be *implementable*, not only should the stability constraint be satisfied, but also the system should be schedulable. The next theorem provides an important analytical result to bound the pessimism involved in using the linear supply functions.

Note that, in many situations, as shown in Section 2.5, the monotonicity property does not hold. This is, for example, the case with respect to task periods and priorities. The following Lemma discusses the monotonicity property with respect to processor speed.

Lemma 4.7.4 *If the stability constraint (4.20) is guaranteed for a task running within a server $S = (Q, P, D)$ on a processor and considering the linear supply functions, it is also guaranteed within the same server and on a higher speed processor.*

Proof: It can be shown that if stability constraint (4.20) is satisfied, then, on a processor with speed augmented by a factor λ ($\lambda \geq 1$), the following stability constraint is also satisfied,

$$\min \left\{ \frac{a(c^w - c^b) + c^b}{\lambda\alpha} + (2a - 1)\Delta - b, \right. \\ \left. \frac{1}{\lambda} \left(\frac{ac^w}{\alpha} - (a - 1)c^b \right) + a\Delta - b \right\} \leq 0. \quad (4.42)$$

The above inequality holds since the terms multiplied by the factor $\frac{1}{\lambda}$ are non-negative considering $\alpha \leq 1$ and $a \geq 1$.

Since the linear stability constraint is satisfied on a processor which is λ times faster, so is the exact stability condition (according to Lemma 4.7.2). $\square$

Theorem 4.7.5 *If a set of controllers is guaranteed to be implemented (i.e., the system is schedulable and all the plants are guaranteed to be stable) using implicit deadline servers $S_i = (Q_i, P_i)$ and considering the exact supply functions over a unit-speed processor, then the same set is guaranteed to be implemented using implicit deadline servers $S'_i = (\frac{Q_i}{2}, \frac{P_i}{2})$ considering the linear supply functions over a λ -speed processor with $\lambda = \sum_{i=1}^n \frac{Q_i + 2\epsilon}{P_i}$, ϵ being the switching overhead.*

Proof: According to Corollary 4.7.3.1, if the stability constraint of a control application can be satisfied within an implicit deadline server $S_i = (Q_i, P_i)$ with the exact supply functions, then it is also satisfied within an implicit deadline server $S'_i = (\frac{Q_i}{2}, \frac{P_i}{2})$ with the linear supply functions. Therefore, server S'_i with the linear supply functions provides guarantees from the stability point of view. However, in addition to stability, the schedulability of the controllers should also be investigated.

Note that, considering the exact supply functions, the system is schedulable if and only if,

$$U = \sum_{i=1}^n \left(\alpha_i + \frac{\epsilon}{P_i} \right) = \sum_{i=1}^n \left(\frac{Q_i}{P_i} + \frac{\epsilon}{P_i} \right) \leq 1. \quad (4.43)$$

Now, let us consider the case where the linear supply functions are used,

$$U' = \sum_{i=1}^n \left(\alpha_i + \frac{\epsilon}{\frac{P_i}{2}} \right) = U + \sum_{i=1}^n \frac{\epsilon}{P_i}. \quad (4.44)$$

The system based on the linear supply functions is schedulable if,

$$\lambda \geq U + \sum_{i=1}^n \frac{\epsilon}{P_i}, \quad (4.45)$$

where λ is the relative speed of the processor. Having considered a processor which is λ times faster, it is now required to discuss the impact of this choice on stability. This is addressed by Lemma 4.7.4, which states that the stability guarantees (with regard to stability constraint (4.20)) are preserved on faster processors. It can also be shown that if constraint (4.10) is satisfied on a processor (i.e., $\frac{c^w}{h} \leq \alpha$), then it is also satisfied on a processor which is faster (i.e., $\frac{1}{\lambda} \frac{c^w}{h} \leq \frac{c^w}{h} \leq \alpha$).

This result indicates the following: if the stability of the controllers cannot be guaranteed on a processor with speed $\lambda = U + \sum_{i=1}^n \frac{\epsilon}{P_i} = \sum_{i=1}^n \frac{Q_i + 2\epsilon}{P_i}$ considering the linear supply functions, then it for sure cannot be guaranteed considering the exact supply functions on a processor with speed 1, since it implies $U > 1$. $\square$

Corollary 4.7.5.1 *The factor λ in Theorem 4.7.5 is bounded from above by 2.*

Proof: Let us assume the system is implementable considering the exact supply functions. This, in turn, implies that the utilization U is

$$U = \sum_{i=1}^n \left(\alpha_i + \frac{\epsilon}{P_i} \right) \leq 1.$$

Since $\alpha_i \geq 0$, we obtain

$$\sum_{i=1}^n \frac{\epsilon}{P_i} = U - \sum_{i=1}^n \alpha_i \leq U.$$

Observe that λ is given by,

$$\lambda = U + \sum_{i=1}^n \frac{\epsilon}{P_i} \leq 2 \cdot U.$$

Since $U \leq 1$ in an implementable system, the bound on λ follows: $\lambda \leq 2$. $\square$

4.8 Asymptotic Analysis

As discussed before, the optimal values of server parameters cannot be obtained efficiently, when the exact supply functions are used. Therefore, we consider the notion of optimistic supply functions, for which the server parameters may be computed efficiently (see Section 4.8).

In this section, we shall identify a lower bound on the minimum achievable utilization by the approach discussed in the previous sections, i.e., the implicit deadline servers. We will use this bound in our experiments (Section 4.9.2) in order to evaluate the efficiency of our optimization technique discussed in Section 4.6. To obtain a tight lower bound on the minimum utilization required for guaranteeing stability of a plant associated with an implicit deadline server, optimistic linear supply functions are considered in this section.

Let us consider the optimistic upper bound on the $\text{slbf}(t)$, denoted by $\text{olslbf}(t)$, and the optimistic lower bound on $\text{subf}(t)$, denoted by $\text{olsubf}(t)$ (see Section 4.7). Lemma 4.7.1 states that if there exists a server with the exact supply functions that can satisfy inequality (4.18), then it is also possible to find a solution using the optimistic linear supply functions that satisfies inequality (4.18).

Forming the stability constraint based on the optimistic response times, defined in Equation (4.40), we realize that the stability constraint (4.20) remains exactly the same, but assuming $\underline{\Delta}$ instead of Δ . Now let us focus on the special case of implicit deadline server

($D = P$) that leads to $\frac{\epsilon}{P} = \frac{\epsilon(1-\alpha)}{\underline{\Delta}}$. The optimization problem then will be as follows,

$$\begin{aligned} \min_{\alpha, \underline{\Delta}} \quad & \alpha + \frac{\epsilon(1-\alpha)}{\underline{\Delta}} \\ \text{s.t.} \quad & \min \left\{ \frac{a(c^w - c^b) + c^b}{\alpha} + (2a-1)\underline{\Delta} - b, \right. \\ & \left. \frac{ac^w}{\alpha} + a\underline{\Delta} - (a-1)c^b - b \right\} \leq 0. \end{aligned} \quad (4.46)$$

Notice that this optimization problem is the same as problem (4.22), except for the factor 2 in the term that captures switching overhead ($\frac{\epsilon(1-\alpha)}{\underline{\Delta}}$ versus $\frac{2\epsilon(1-\alpha)}{\underline{\Delta}}$). Therefore, the solution to this problem can be obtained using the approach in Section 4.6, simply by substituting the overhead $\frac{\epsilon}{2}$ instead of ϵ .

While the objective function of the above optimization problem is the same for both the exact and optimistic supply functions (i.e., $\alpha + \frac{\epsilon}{P}$), the stability constraint uses optimistic response times instead of the exact results. However, according to Lemma 4.7.1, if there exists a solution that guarantees the stability constraint when the exact supply functions are used, then there is also a solution considering the optimistic supply functions (i.e., the search space of this problem contains the search space of the exact problem). Therefore, the total utilization found using the asymptotic approach in this section is lower than or equal to the one that could possibly be found for the implicit deadline server considering the exact supply functions.

4.9 Evaluation

We will first illustrate and evaluate our proposed approach in Section 4.9.1 by a small example and later in Section 4.9.2 by a large set of experiments.

4.9.1 Illustrative Example

In this section, the server design approach discussed in Section 4.6 will be illustrated using a small example. Further, we also compare the results to the asymptotic bound developed for the case of implicit deadline servers in Section 4.8.

Table 4.1: Example: taskset data

i	c_i^b	c_i^w	h_i	a_i	b_i	$F(s)$
1	30	60	600	1.18	831	$\frac{1000}{s^2+s}$
2	92	184	920	1.16	826	$\frac{98.1}{s^2-98.1}$
3	427	854	2847	1.14	2697	$\frac{9.81}{s^2-9.81}$

Let us consider a set of three controllers whose parameters are reported in Table 4.1. In the table we report best-case and worst-case execution times (c_i^b and c_i^w), the period (h_i), the coefficients of the linear stability constraint (a_i and b_i of the constraint in (4.18)), and the transfer function of the plant to be controlled. We assume a server switching overhead of $\epsilon = 0.3$. All time quantities are given in units of 0.01 *ms* throughout this section.

The server parameters obtained after optimization are reported in Table 4.2. In the first column group of the table (labelled by “Proposed Design”) we report server budgets Q_i , periods P_i , bandwidth α_i , delay Δ_i , and overhead due to switching $O_i = \frac{\epsilon}{P_i}$ for the proposed design strategy (PD) proposed in Section 4.6. In the second column group of the table (labelled by “Asymptotic Analysis”), the corresponding results for the asymptotic analyses of the implicit deadline server (AA) in Section 4.8 are reported.

The total utilization obtained by the asymptotic analysis for the proposed approach in Section 4.8 is $U_{AA} = 0.71$, while the total utilization in the case of the implicit deadline servers obtained by our design approach (Section 4.6) is slightly higher, i.e., $U_{PD} = 0.72$. The detailed calculation is given in the following,

$$\begin{aligned}
 U_{AA} &= \sum_{i=1}^3 \left(\alpha_i^* + \frac{\epsilon(1-\alpha_i^*)}{\Delta_i^*} \right) = \left(0.100 + \frac{0.3(1-0.100)}{130} \right) \\
 &\quad + \left(0.249 + \frac{0.3(1-0.249)}{23.6} \right) + \left(0.345 + \frac{0.3(1-0.345)}{34.4} \right) = 0.71, \\
 U_{PD} &= \sum_{i=1}^3 \left(\alpha_i^* + 2 \frac{\epsilon(1-\alpha_i^*)}{\Delta_i^*} \right) = \left(0.100 + 2 \frac{0.3(1-0.100)}{130} \right) \\
 &\quad + \left(0.253 + 2 \frac{0.3(1-0.253)}{32.8} \right) + \left(0.347 + 2 \frac{0.3(1-0.347)}{48.3} \right) = 0.72.
 \end{aligned}$$

Observe that the solution found by the asymptotic analysis for the implicit deadline servers is not guaranteed to be stable (it does not guarantee stability considering the exact stability condition). However, the solution obtained by our proposed approach is guaranteed to be stable (valid considering the exact stability condition), while

Table 4.2: Example: Solution to the server design problem.

i	Proposed Design					Asymptotic Analysis				
	Q_i^*	P_i^*	α_i^*	Δ_i^*	O_i^*	Q_i^*	P_i^*	α_i^*	Δ_i^*	O_i^*
1	7.25	72.5	0.100	130	0.004	14.5	145	0.100	130	0.002
2	5.56	22.0	0.253	32.8	0.010	7.82	31.4	0.249	23.6	0.010
3	12.8	37.0	0.347	48.3	0.008	18.1	52.5	0.345	34.4	0.006
Σ	0.700				0.022	0.694				0.018

only 1% away from the asymptotic analysis, in terms of resource utilization. This indicates that, for the discussed example, the solution obtained by our approach is less than 1% away from the actual optimum.

4.9.2 Experimental Results

To further evaluate our proposed server design approach we compare against the asymptotic analysis discussed in this chapter:

- **Proposed control servers:** the stabilizing control server approach (PD) proposed in Section 4.6, which is based on the implicit deadline servers.
- **Implicit deadline asymptotic analysis:** the asymptotic analysis of implicit deadline servers (AA) is discussed in Section 4.8 and produces solutions that are not guaranteed to be stable, but their resource utilization is less than or equal to the actual optimum.

Note that the asymptotic analysis (AA) is an approach that outperforms the optimal, in terms of total bandwidth usage, if the class of implicit deadline servers is considered. In other words, the asymptotic analysis (AA) produces a lower bound on the total bandwidth usage, but does not guarantee stability. Therefore, this asymptotic analysis (AA) is considered as the baseline for the comparison in this section. We compare the proposed design approach for stabilizing control servers (PD) to the asymptotic analysis (AA), also developed in this paper. The difference captures the efficiency of the proposed approach (PD), assuming the asymptotic analysis (AA) produces solutions that are often close to the optimal.

We have generated 700 benchmarks with a number of control applications from 2 to 10. The plants considered are chosen from a database consisting of inverted pendulums, ball and beam processes, DC servos, and harmonic oscillators [ÅW97, CLE⁺04]. To generate a set of random control tasks for a given utilization, the UUniFast algorithm is used [BB05]. The periods are chosen based on common rules of thumb [ÅW97]. Having the period and task utilization, the worst-case execution time can be computed. The switching overhead is given by $\epsilon = r \cdot \min_{i=1 \dots n} \{c_i^b\}$, where r is randomly chosen with a uniform distribution in the interval of $[0.01, 0.10]$.

The experiments are repeated for several values of total task utilization ($\sum_{i=1}^n \frac{c_i^w}{h_i}$) and the results are shown in Figure 4.6. The metric used for this comparison is the relative quality, defined as $\left(\frac{N_{AA} - N_{PD}}{N_{AA}} \times 100 \right)$, where N_{PD} and N_{AA} are the number of benchmarks for which the proposed approach (PD) and asymptotic analysis (AA), respectively, could find a valid solution. Therefore, the metric states the quality of the proposed approach (PD) compared to the asymptotic analysis (AA). For each value of utilization, we evaluate the percentage of benchmarks for which the stability could not be guaranteed, and we call it “invalid solutions”.¹

It can be observed that the number of invalid solutions by the proposed design approach (PD) compared to the asymptotic analysis (AA) increases with the taskset utilization. It is also noteworthy that the gap between the proposed design approach (PD) and the asymptotic analysis (AA) is always less than 10%. In other words, the proposed design approach (PD) is less than 10% away from the theoretical optimum, for the benchmarks considered here.

4.10 Conclusions

Providing guarantees for stability of control applications is perhaps the most important requirement while implementing embedded control systems. The fundamental difference between the control systems and what we classically understand by hard real-time systems advocates the need for new analysis and design techniques. In this chapter, we have proposed the use of resource reservation mechanisms

¹Note that, as mentioned, “valid” solutions with AA are not guaranteed to be stable, as opposed to those produced with PD.

Figure 4.6: The percentage of the benchmarks for which stability of the control tasks could not be guaranteed using the proposed control server design approach (PD), compared to the asymptotic analysis (AA).

for designing embedded control systems. Exploiting the server mechanism provides not only compositionality, scalability, and isolation, but also a simple interface between the control stability and real-time scheduling aspects which facilitates the design process. Finally, we have addressed the analysis and design of stabilizing servers and demonstrated the efficiency of our proposed approaches both theoretically and experimentally.

A SCHEDULING POLICY TO STABILIZE CONTROL APPLICATIONS

IN the previous chapters we discussed offline methodologies for designing embedded control systems. An online scheduling policy to stabilize control applications will be proposed in this chapter.

It has previously been discussed that jitter is typically an important factor for control applications. This chapter investigates whether it is possible to guarantee that the jitter will not exceed a certain amount, for a given set of applications on a shared platform. The effect of jitter on the stability of control applications and its relation with the latency will be discussed. The importance of our technique arises from the fact that it is considerably easier in controller design to manage the constant part of the delay (known as latency), while coping with the varying part of the delay (known as jitter) is more involved. The proposed solution guarantees certain jitter limits, and at the same time does not lead to overly pessimistic latency values. The results are later used in a design optimization problem to minimize resource utilization.

5.1 Introduction and Related Work

As it has been discussed in the previous chapters, over the past decade, there has been considerable amount of research on offline methodologies for the control–scheduling co-design problem [SLSS96, RS00, CLE⁺04, NPAG06, BC08, ZSWM08, NH09, MSZ11, GLSC12,

KGC⁺12, ASE⁺12, ABEP13]. Typically [CLE⁺04, ASE⁺12, ABEP13], response-time analysis is used to calculate the latency and jitter introduced as a result of resource sharing and, then, this information is utilized to investigate the stability of the plants [ÅW97, Cer12].

The approach in this chapter is of another nature: given a set of applications and the maximum amount of jitter which can be tolerated by each application, is it possible to schedule the system such that all applications satisfy their jitter constraints? Here, we focus on the notion of *lag* [BCPV96], i.e., the difference between the amount of time allocated to a task on a shared processor and a dedicated processor with speed equal to the task utilization. Towards this, we model the schedule as a system where the state of the system is the lag of all tasks. To bound the jitter of a task, the lag should be kept within a specified limit, which is obtained by considering the relation between the jitter and lag. It will be shown that it is possible to limit the lags for a set of applications with a total utilization not exceeding one, for arbitrary positive lag limits. In order to limit the lags, and in turn the jitters, we propose a simple online scheduling policy. In other words, our task is to design a scheduling policy to guarantee the required lag limits. The complexity and the context switch overhead of the proposed scheduling policy will be discussed. In particular, it is shown that, in the worst case, the number of preemptions by our proposed policy is at most three times the number of preemptions by the optimal policy. Finally, the bound on the jitter for control applications is translated into stability guarantees. It will also be demonstrated how to assign processor shares and select lag limits in designing such systems in order to guarantee stability for a set of control applications.

An important feature of the algorithm is that it requires very little information about the taskset at design time. An engineering solution to avoid jitter is to buffer the last part of the task execution until before the worst-case response time of the tasks [VZF91], which requires considerably more information at design time. Moreover, such a technique would unnecessarily extend the latency towards the worst-case response time.

The Pfair algorithm [BCPV96] addresses the problem of proportional progress for tasks execution. In other words, it guarantees that the difference between the amount of time allocated to each task

and the corresponding value in the fluid model of execution does not exceed one time unit of execution. The schedule may, however, experience preemption every time unit. As opposed to Pfair, in the proposed approach, the constraint on the lag limits for each application could be individually and arbitrarily selected. The concept of *regularity*, similar to the lag concept, is discussed in [MF01]. The authors also discuss the schedulability conditions when the concept of regularity is considered. A closely related technique is the server-based resource reservation mechanism and the use of the (α, Δ) model [FM02] to design servers to bound the output jitter.

In [BBGL99], Baruah et al. develop two algorithms to minimize the output jitter, as opposed to limiting the jitter. Di Natale and Stankovic [NS00] propose a framework based on simulated annealing to synthesize static schedules with minimum output jitter for non-preemptive tasks and messages, in distributed systems. In [WBSS11a], Westmijze et al. study the effectiveness of a number of scheduling heuristics intended to reduce the latency and jitter taking into account the execution times of tasks as well as dependencies between the tasks, the data structures accessed by the tasks, and the memory hierarchy. However, the complexity of the model considered in [WBSS11b] renders it impossible to provide any kind of guarantees. As opposed to our approach, the jitter minimization requires the complete information about the tasks parameters during the design phase [BBGL99, NS00, WBSS11b].

Hong et al., in [HHL10], propose a heuristic based on elastic scheduling to reduce the jitter through adaptive deadline adjustment. In [PL13], Phan et al. propose to reduce the output jitter of the task for fixed-priority policy using shapers. Essentially the authors propose to delay the already released job for a certain amount of time to bound the resource demands. However, the solutions proposed in [HHL10] and [PL13] are similar to the buffering mechanism discussed previously. Mochocki et al. [MHRE05] address the problem of guaranteeing jitter constraints, but using the dynamic voltage scaling (DVS) technique.

The *fairness* concept is defined by [BCPV96]. A policy is considered *fair* if tasks progress roughly proportional to their allocated resource share. However, as discussed before, the algorithm can only guarantee *equal* lag limits. As opposed to Pfair, our proposed algo-

rithm provides guarantees on the independent amount of *jitter* an application can tolerate and therefore the algorithm is referred to as *Jfair*. Note that fairness does not necessarily translate into equality as applications may tolerate different amount of variations in the response time in order to remain stable.

The interplay between the latency and jitter of online scheduling policies and the connection to the stability of control applications has not been discussed in the literature. The most important results presented in this chapter are the following:

- We prove that it is possible to guarantee any (positive) jitter limits for tasks as long as the utilization is not exceeding one.
- The proof is constructive, i.e., a scheduling policy is proposed to synthesize such a schedule.
- It is also proved that the number of preemptions needed by the proposed scheduling policy will be at most three times that of any other valid (feasible) scheduling policy, including the optimal one.
- We address the analysis problem to select the lag limits, given the constraints on the latency and jitter required to guarantee stability.
- Based on the proposed online scheduling policy, a design optimization problem is formulated to minimize the resource utilized to guarantee the stability of the control applications. The design parameters are the lag limits and processor shares.

5.2 System Model and Background

We are given n independent periodic tasks, each of which denoted by τ_i with computation time (execution time) c_i and period h_i . Each task τ_i has a utilization, denoted by $u_i = \frac{c_i}{h_i}$, which is defined as the portion of resource allocated to this task. The j -th instance of τ_i is referred to as job and is denoted by $\tau_{i,j}$. The *Jfair* algorithm divides each job $\tau_{i,j}$ into several subjobs. The k -th subjob of $\tau_{i,j}$ is denoted by $\tau_{i,j,k}$.

We denote by $r_{i,j}$ and $f_{i,j}$ the release and finishing time instants of the j -th job of τ_i , respectively. The jitter J_i of τ_i is defined as

$$J_i = \max_j \{f_{i,j} - r_{i,j}\} - \min_j \{f_{i,j} - r_{i,j}\}. \quad (5.1)$$

Such a quantity is also called output jitter, as it measures the variation in the response time of a task.

Let us define the execution function $e_i(t)$,

$$e_i(t) = \begin{cases} 1 & \text{if } \tau_i \text{ is executing at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

For a given task schedule, the *lag* of task τ_i is defined [BCPV96] as,

$$\delta_i(t) = u_i \cdot t - \int_0^t e_i(x) dx. \quad (5.2)$$

The definition captures the difference between the amount of time that should have been allocated to task τ_i until time t according to its utilization u_i (i.e., $u_i \cdot t$) and the amount that is actually allocated (i.e., $\int_0^t e_i(x) dx$).

The lag at any instant t_2 may also be written as a function of the lag at any previous instant t_1 ($\leq t_2$), as follows

$$\begin{aligned} \delta_i(t_2) &= u_i \cdot t_2 - \int_0^{t_2} e_i(x) dx \\ &= u_i \cdot t_1 + u_i \cdot (t_2 - t_1) - \int_0^{t_1} e_i(x) dx - \int_{t_1}^{t_2} e_i(x) dx \\ &= \delta_i(t_1) + u_i \cdot (t_2 - t_1) - \int_{t_1}^{t_2} e_i(x) dx. \end{aligned} \quad (5.3)$$

The *lag limit* is defined as the maximum allowed deviation in the absolute value of the lag for task τ_i , and is denoted by $\bar{\delta}_i$. A schedule is *feasible* if the lag limits are satisfied. That is, a feasible schedule guarantees that,

$$|\delta_i(t)| \leq \bar{\delta}_i, \quad \forall t \geq 0, \quad i = 1, \dots, n.$$

The relation between the jitter and the lag will be clarified in the following. Assuming that lag $\delta_i(t_2)$ for task τ_i is within the lag limits

at time t_2 we obtain,

$$\begin{aligned}
 |\delta_i(t_2)| &\leq \bar{\delta}_i, \\
 |\delta_i(t_1) + u_i \cdot (t_2 - t_1) - k \cdot c_i| &\leq \bar{\delta}_i, \\
 \frac{k \cdot c_i - \delta_i(t_1) - \bar{\delta}_i}{u_i} &\leq t_2 - t_1 \leq \frac{k \cdot c_i - \delta_i(t_1) + \bar{\delta}_i}{u_i}, \\
 t_1 + \frac{k \cdot c_i - \delta_i(t_1) - \bar{\delta}_i}{u_i} &\leq t_2 \leq t_1 + \frac{k \cdot c_i - \delta_i(t_1) + \bar{\delta}_i}{u_i},
 \end{aligned}$$

where t_2 is the instant at which the k -th job of task τ_i may finish its execution and t_1 is the release time of the 1-st job of task τ_i . Therefore, we have $\int_{t_1}^{t_2} e_i(x) dx = k \cdot c_i$. The second inequality is obtained using Equation (5.3). Therefore, the output jitter J_i , that is equivalent to the variation in t_2 , is bounded as follows,

$$J_i \leq 2 \cdot \frac{\bar{\delta}_i}{u_i}. \quad (5.4)$$

This indicates that the limit on the output jitter for task τ_i translates into a constraint on lag,

$$|\delta_i(t)| \leq \bar{\delta}_i = \frac{J_i}{2} \cdot u_i, \quad (5.5)$$

assuming J_i is the maximum amount of jitter a task τ_i can tolerate (i.e., jitter margin).

We shall also define the notion of *pending work* $\sigma_i(t)$ for a periodic task τ_i at t ,

$$\sigma_i(t) = \left\lfloor \frac{t}{h_i} + 1 \right\rfloor c_i - \int_0^t e_i(x) dx. \quad (5.6)$$

By definition, the pending work may not be negative at any time. Clearly, a task τ_i may not execute when $\sigma_i(t) = 0$.

5.3 Motivational Example

In this section, we shall motivate the need for the scheduling policy proposed. Let us consider a set of three tasks $\mathbf{T} = \{\tau_1, \tau_2, \tau_3\}$. The execution times of all tasks are $c_i = 5$. The sampling period of task τ_1 is $h_1 = 10$, while for tasks τ_2 and τ_3 the periods are $h_2 = h_3 = 20$.

(a) Pfair

(b) Server-based

(c) Jfair

Figure 5.1: Example: three tasks with execution times $c_i = 5$, periods $h_1 = 10$, $h_2 = h_3 = 20$, lag limits $\bar{\delta}_i = 1$.

Figure 5.1 illustrates the task schedule under several scheduling policies: the Pfair algorithm [BCPV96], a server-based algorithm [AEP15], and our proposed Jfair algorithm. To compare our proposed approach against the Pfair algorithm, we set the lag limits $|\delta_i(t)| \leq \bar{\delta}_i = 1$, although our policy does not require such a condition. The results are summarized in Table 5.1. The preemption density for a taskset $\mathbf{T} = \{\tau_i \mid i = 1 \dots n\}$ is defined as,

$$\rho = \sum_{i=1}^n \frac{m_i}{h_i}, \quad (5.7)$$

where m_i is the number of preemptions of task τ_i in one period h_i .

The three tasks τ_1 , τ_2 , and τ_3 are depicted in green, red, and blue, respectively. The x-axis is the time axis and the lags are shown on the y-axis. The upward arrows indicate the release time of the corresponding jobs. The lags are also depicted with the piecewise linear lines in the same color as the task. However, for the sake of illustration, the lags are displaced such that $|\delta_i(t) - i| \leq 1$. For example, the lag for the red task τ_2 is drawn by red and lag $\delta_2(t) = 0$ at time t if it intersects with line $y = 2$. Therefore, the lags are valid as long as $\delta_2(t)$ is between $y = 1$ and $y = 3$, i.e., $|\delta_2(t) - 2| \leq 1$.

Figure 5.1(a) shows the schedule by the Pfair algorithm [BCPV96]. In every time unit, the task executing is preempted, which leads to 20 preemptions in total.

The schedule by the server-based approach (resource reservation mechanism) is shown in Figure 5.1(b). As in the previous case, the servers (or shapers) are designed to guarantee the lag limits of $|\delta_i(t)| \leq 1$ (a more detailed discussion will be presented in Section 5.4). It can be seen that the number of preemptions is 24, i.e., more than the Pfair algorithm, for this particular example.

The proposed Jfair algorithm (discussed in Section 5.4) is shown in Figure 5.1(c). Notice that while the lag limits are still respected, the number of preemptions is reduced to 14. As it will be discussed later, the number of preemptions by the optimal algorithm, for this example, is bounded from below by 8.

This example demonstrates that the schedule generated by the Jfair policy, while respecting the lag limits, will lead to a fewer number of preemptions, when compared to Pfair or servers.

In the next section, our goal is to design a scheduling policy to guarantee the lag limits for all tasks.

Table 5.1: Summary of the characteristics of the schedule.

policy	preemption density ρ	lag limit $\bar{\delta}$
Pfair	1.0	1
Server	1.2	1
Jfair	0.7	1

5.4 Jfair Scheduling Policy

In this section, we discuss the Jfair scheduling policy and the theoretical guarantees provided by the proposed approach. Without loss of generality, we assume that the resource is fully utilized, i.e., $\sum_{i=1}^n u_i = 1$. This is simply possible by introducing an extra task with utilization $u_0 = 1 - \sum_{i=1}^n u_i$ and implicit deadline. For instance, by choosing the period and deadline equal to the hyper period and the execution time in such a way that the task utilization is equal to u_0 .

5.4.1 Scheduling Policy

Our proposed algorithm divides each job $\tau_{i,j}$ of task τ_i into several smaller subjobs $\tau_{i,j,k}$, for each of which the execution time and artificial deadline are denoted by $c_{i,j,k}$ and $d_{i,j,k}$, respectively. In fact, our objective is to assign the execution times $c_{i,j,k}$ and deadlines $d_{i,j,k}$ in such a way that the lag limits are not violated.

The scheduling policy is as follows:

- The scheduler is an EDF-based scheduler, i.e., the subjob with the earliest deadline has the highest priority.
- The deadline $d_{i,j,k}$ of a subjob is set as follows,

$$d_{i,j,k} = \frac{\bar{\delta}_i}{u_i \cdot (1 - u_i)}. \quad (5.8)$$

In the case where $\sigma_i(t) < d_{i,j,k} \cdot u_i$, where t is deadline of the previous subjob $\tau_{i,j,k-1}$, the deadline is modified to

$$d_{i,j,k} = \frac{\sigma_i(t)}{u_i}. \quad (5.9)$$

- The subjob's execution time at time t is assigned according to

$$c_{i,j,k} = d_{i,j,k} \cdot u_i. \quad (5.10)$$

To have a better understanding of the algorithm, let us first have a closer look at how our simple scheduling policy works. Intuitively, the scheduler breaks a job into several subjobs.

Starting with $\delta_i(0) = 0$, the deadline and execution-time are $d_{i,j,1} = \frac{\bar{\delta}_i}{u_i \cdot (1-u_i)}$ and $c_{i,j,1} = \frac{\bar{\delta}_i}{1-u_i}$, respectively. At time $t = d_{i,j,1}$, the lag $\delta_i(t) = 0$ and the relative deadline and execution-time are $d_{i,j,2} = \frac{\bar{\delta}_i}{u_i \cdot (1-u_i)}$ and $c_{i,j,2} = \frac{\bar{\delta}_i}{1-u_i}$, respectively. Therefore, the schedule is similar to a quasi-periodic task τ'_i with period and deadline $h'_i = d'_i = \frac{\bar{\delta}_i}{u_i \cdot (1-u_i)}$ and execution-time $c'_i = \frac{\bar{\delta}_i}{1-u_i}$. The last subjob of each job might have shorter deadline and execution time. However, note that the utilization for the quasi-periodic task is kept the same as the original task $u'_i = \frac{c'_i}{h'_i} = u_i$.

An example is shown in Figure 5.1(c). Let us consider task τ_1 shown in green. At time $t = 0$, the first subjob is assigned $d_{1,1,1} = \frac{\bar{\delta}_1}{u_1 \cdot (1-u_1)} = \frac{1}{0.5(1-0.5)} = 4$ and $c_{1,1,1} = d_{1,1,1} \cdot u_1 = 2$. At time $t = d_{1,1,1} = 4$, the next subjob is assigned $d_{1,1,2} = \frac{1}{0.5(1-0.5)} = 4$ and $c_{1,1,2} = d_{1,1,2} \cdot u_1 = 2$, but since the subjob of task τ_3 has shorter deadline, this subjob may not execute immediately. At time $t = 4 + d_{1,1,2} = 8$, only 1 unit of the job execution is left and therefore the subjob has $d_{1,1,3} = \frac{\sigma_1(t)}{u_1} = 2$ and $c_{1,1,3} = d_{1,1,3} \cdot u_1 = 1$.

5.4.2 Theoretical Guarantees

We shall now show that the system could be scheduled by our algorithm with any positive lag limits, as long as the utilization of the taskset is not exceeding one.

5.4.2.1 Non-Violation of Upper Lag Limit

In order to avoid violating the upper lag limits, the concept of deadline is introduced and the scheduler should allocate the proper amount of resource share before the deadline of the subjob is expired.

Before formally defining the deadline for each subjob, let us define the time instant $t + y(t)$ (where $y(t)$ is relative to time t and is

Figure 5.2: The notion of deadline for a subjob.

calculated at time t) for a subjob as the time before (or at) which the task must execute in order to respect the lag constraint defined in Equation (5.5).

The relative deadline $d_{i,j,k}$ for a subjob $\tau_{i,j,k}$ (which is relative to time t) is the point in time after $t + y(t)$, at which the lag $\delta_i(t + d_{i,j,k})$ becomes zero, assuming contiguous execution after instant $t + y(t)$. The deadline defined as such guarantees that the resource allocated in the interval of $[0, t + d_{i,j,k}]$ is $u_i \cdot (t + d_{i,j,k})$. Figure 5.2 illustrates the concept of deadline.

Note that starting with $\delta_i(0) = 0$, the lag $\delta_i(t)$ will always be zero at the assigned deadlines. This is an important property which we will use throughout this section.

Let us assume that we would like to find the deadline at time t , with lag $\delta_i(t) = 0$. Therefore, the objective is to find the time instant t' at which lag $\delta_i(t')$ is at the upper lag limit $\bar{\delta}_i$, if the task does not execute in the interval of $[t, t']$ (i.e., $\int_t^{t'} e_i(x) dx = 0$),

$$\begin{aligned} \delta_i(t') &= \delta_i(t) + u_i \cdot (t' - t) - \int_t^{t'} e_i(x) dx, \\ \bar{\delta}_i &= 0 + u_i \cdot (t' - t) + 0. \end{aligned} \quad (5.11)$$

Note that $y(t) = t' - t$, which means $y(t)$ at time t is given by,

$$y(t) = \frac{\bar{\delta}_i}{u_i}. \quad (5.12)$$

Of significant importance, the instant $t + y(t)$ identifies the time at which the lag limits will be violated if the subjob does not execute (at all) before or at that instant. Then, the relative deadline $d_{i,j,k}$ is defined as follows,

$$d_{i,j,k} = \frac{\bar{\delta}_i}{u_i} + \frac{\bar{\delta}_i}{1 - u_i} = \frac{\bar{\delta}_i}{u_i \cdot (1 - u_i)}, \quad (5.13)$$

where the execution time to reach to the zero lag is given by $c_{i,j,k} = \frac{\bar{\delta}_i}{1 - u_i}$.

If the subjobs of a task do not violate their deadlines $d_{i,j,k}$, the lag constraint $\delta_i(t) \leq \bar{\delta}_i$ will not be violated. In other words, as long as the deadlines are met, the lag will not exceed the upper limit, i.e., $\delta_i(t) \leq \bar{\delta}_i$.

5.4.2.2 Non-Violation of Lower Lag Limit

To avoid violating the lower lag limits $-\bar{\delta}_i$, the scheduling policy should limit the execution times of subjobs. Assuming at time t a task τ_i has a lag $\delta_i(t) = 0$, as discussed in the previous section, the maximum execution time before hitting the lower lag limit $-\bar{\delta}_i$, is given by,

$$\begin{aligned} -\bar{\delta}_i &\leq \delta_i(t) + u_i \cdot c_{i,j,k} - c_{i,j,k}, \\ -\bar{\delta}_i &\leq 0 + c_{i,j,k} \cdot (u_i - 1). \end{aligned} \quad (5.14)$$

This indicates that the lower lag limit will not be violated, as long as

$$c_{i,j,k} \leq \frac{\bar{\delta}_i}{1 - u_i}. \quad (5.15)$$

5.4.2.3 Scheduling Properties

It should now be clear that the lag limits will not be violated as long as the subjob execution-time does not exceed the limit we discussed in the previous section and the deadlines are met. The next theorem guarantees that all lag limits will be respected, provided $\sum_{i=1}^n u_i \leq 1$.

Theorem 5.4.1 *Given a periodic taskset and the positive and independent lag limits for each task, there exists a scheduling policy that satisfies the lag limits, iff the utilization of the taskset does not exceed 100%. The Jfair algorithm is an instance of such a scheduling policy.*

Proof: We shall first focus on the sufficient part. It has been shown that the lag constraints are satisfied as long as all subtasks scheduled satisfy Equation (5.14) and all deadlines are met. It has been shown, in the previous subsection, that the former property is satisfied implicitly under Jfair scheduling. The latter is proved by contradiction in the following, for the proposed policy.

The proof is similar to the earliest-deadline-first (EDF) scheduling. Let us assume that we have the first deadline miss at time t_1 , while $\sum_{i=1}^n u_i \leq 1$. Let us further consider the longest contiguous busy interval before time t_1 where all the subjobs executed in the interval of $[t_0, t_1]$ should have their release times (or activation times) and deadlines in this interval. Let us denote the start time of this interval by t_0 and its length by $l = t_1 - t_0$. Since we considered the longest contiguous busy interval ending at t_1 , just before time t_0 , the resource is not processing subjobs with deadlines less than or equal to t_1 . This means that t_0 should be the release time (or activation time) of a subjob with deadline before t_1 . Since the assigned execution times in Jfair are computed based on $c_{i,j,k} = d_{i,j,k} \cdot u_i$, the demand of each task τ_i in this interval, denoted by x_i , is less than or equal to $l \cdot u_i$, i.e., $l \cdot u_i \geq x_i$. Taking all tasks (with both activation and deadline in this interval) into consideration, we obtain $\sum_i l \cdot u_i \geq \sum_i x_i$. Having a deadline miss at time t_1 indicates that $\sum_i x_i > l$. From $\sum_i l \cdot u_i \geq \sum_i x_i$ and $\sum_i x_i > l$, we conclude $\sum_i l \cdot u_i > l$. This indicates that $\sum_{i=1}^n u_i > 1$, which is a contradiction.

The necessary part is simply because no taskset with utilization above one is schedulable (the deadlines will be violated). $\square$

The next theorem discusses the number of preemptions when using the Jfair algorithm, compared to the optimal scheduling policy. We consider the finishing of a job execution as one preemption.

Theorem 5.4.2 *The number of preemptions by the proposed scheduling policy is at most three times the number of preemptions by any feasible schedule under the given lag limits.*

Proof: First, we shall find a lower bound on the number of preemptions experienced by task τ_i in one period h_i , considering any feasible scheduling policy. Note that the set of feasible scheduling policies includes also the optimal scheduling policy, where the optimality is defined in terms of number of preemptions.

The proof is based on the observation that an upper bound on the maximum possible contiguous execution time, denoted by x , for each task can be computed. The upper bound is obtained in the following scenario: the task has lag $\delta_i(t) = \bar{\delta}_i$ and it executes without preemption until the lag is $\delta_i(t+x) = -\bar{\delta}_i$,

$$-\bar{\delta}_i = \bar{\delta}_i + u_i \cdot x - x \quad \Rightarrow \quad x = \frac{2 \cdot \bar{\delta}_i}{1 - u_i}. \quad (5.16)$$

From this, it is clear that the optimal number of preemptions for a task may not be less than $\lceil \frac{c_i}{x} \rceil = \lceil \frac{c_i \cdot (1-u_i)}{2 \cdot \bar{\delta}_i} \rceil$ in a period h_i . This is a lower bound on the minimum number of preemptions for task τ_i in one period h_i .

Secondly, let us compute the number of preemptions caused by task τ_i in one period h_i . The proof is based on the fact that the number of preemptions in a scheduling policy based on EDF is not more than the number of jobs. In short, this is due to the fact that a task is only preempted when a job of another task is released.

We shall now discuss the number of subjobs generated by our proposed policy for task τ_i in one period h_i . Considering the quasi-periodic pattern of execution, the number of subjobs in one period h_i will be $\lceil \frac{c_i \cdot (1-u_i)}{\bar{\delta}_i} \rceil$. This is an upper bound on the maximum number of preemptions caused by task τ_i in one period h_i , when Jfair is used. Further, we shall consider one extra preemption when the job finishes its execution, i.e., $\lceil \frac{c_i \cdot (1-u_i)}{\bar{\delta}_i} \rceil + 1$.

Finally, observe that the following inequality holds for a lower bound on the minimum number of preemptions of task τ_i by any algorithm (including the optimal one) and an upper bound on the maximum number of preemptions caused by task τ_i in our Jfair algorithm, in one period h_i ,

$$\frac{\lceil \frac{c_i \cdot (1-u_i)}{\bar{\delta}_i} \rceil + 1}{\lceil \frac{c_i \cdot (1-u_i)}{2 \cdot \bar{\delta}_i} \rceil} \leq 3. \quad (5.17)$$

Therefore, the number of preemptions by our algorithm may not be more than three times that of the optimal policy, where optimality is defined in terms of number of preemptions. $\square$

5.5 Stability, Analysis, and Design

As it has been discussed previously, the latency and jitter in the execution of the control tasks can lead to poor performance and may even jeopardize the stability of the control applications. As discussed in Chapter 2, to quantify the impact of the latency and jitter on the stability of the plant, we use the Jitter Margin toolbox, which provides sufficient stability conditions for a closed-loop system with a linear continuous-time plant and a linear discrete-time controller.

Given a sampling period, the stability curve can safely and efficiently be approximated by a linear function of the latency and worst-case response-time jitter. The linear *stability condition* for a control application is of the form,

$$L_i + a_i \cdot J_i \leq b_i,$$

where $a_i \geq 1, b_i \geq 0$. The latency L_i is the constant part of the delay that the control application experiences, whereas the worst-case response-time jitter J_i captures the varying part of the delay.

5.5.1 Control Stability and Jfair

In order to use the stability condition introduced above, the values of the latency (L_i) and worst-case response-time jitter (J_i) of the control task should be computed. As mentioned before, the stability constraint is formulated as,

$$L_i + a_i \cdot J_i \leq b_i. \tag{5.18}$$

The latency and jitter are given by (see Equation (5.4)),

$$\begin{aligned} L_i &= \frac{c_i}{u_i} - \frac{J_i}{2} = \frac{c_i - \bar{\delta}_i}{u_i}, \\ J_i &= \frac{2 \cdot \bar{\delta}_i}{u_i}. \end{aligned} \tag{5.19}$$

Given the control and real-time parameters of an application, the

lag limit is constrained by

$$\begin{aligned}
 L_i + a_i \cdot J_i &\leq b_i, \\
 \frac{c_i - \bar{\delta}_i}{u_i} + a_i \cdot \frac{2 \cdot \bar{\delta}_i}{u_i} &\leq b_i, \\
 \bar{\delta}_i &\leq \frac{b_i \cdot u_i - c_i}{2 \cdot a_i - 1},
 \end{aligned} \tag{5.20}$$

and the control task should be scheduled for this lag limit in order to guarantee the stability of the plant.

It is worth noting that reducing the lag limit $\bar{\delta}_i$ of an already stable task leads to an increase in the latency L_i , but a decrease in the jitter J_i (see Equation (5.19)). This does not lead to instability as shown in the following,

$$\begin{aligned}
 L_i + a_i \cdot J_i &\leq b_i, \\
 \frac{c_i - \bar{\delta}_i}{u_i} + a_i \cdot \frac{2 \cdot \bar{\delta}_i}{u_i} &\leq b_i, \\
 \frac{c_i}{u_i} + \frac{(2 \cdot a_i - 1)}{u_i} \cdot \bar{\delta}_i &\leq b_i,
 \end{aligned} \tag{5.21}$$

since the coefficient of the lag limit $\bar{\delta}_i$, i.e., $\frac{(2 \cdot a_i - 1)}{u_i}$, is positive.

5.5.2 Design for Stability and Jfair

In this section, we shall formulate the design for stability problem in the context of Jfair. Given a set of control tasks with the execution times c_i and periods h_i , we would like to guarantee stability of the plants associated with the control tasks, using the minimum amount of resource. We do this by achieving an optimal combination of latency and jitter for each controller.

The latency and jitter experienced by each task τ_i when scheduled by Jfair are connected to u_i and $\bar{\delta}_i$ for the task. In order to formulate the optimization problem, we introduce the processor share inflation factors s_i for each task τ_i . The Jfair policy takes the inflated utilization $u_i \cdot s_i$ into account to schedule the tasks. Our optimization problem finds the optimal values of the processor share inflation factors s_i and lag limits $\bar{\delta}_i$ for all applications. The stability constraint

considering the inflation factor s_i is

$$\begin{aligned} L_i + a_i \cdot J_i &\leq b_i, \\ \frac{c_i - \bar{\delta}_i}{u_i \cdot s_i} + a_i \cdot \frac{2 \cdot \bar{\delta}_i}{u_i \cdot s_i} &\leq b_i, \\ c_i + (2 \cdot a_i - 1) \cdot \bar{\delta}_i - b_i \cdot u_i \cdot s_i &\leq 0. \end{aligned} \quad (5.22)$$

There are numerous optimization objectives that could be discussed. One common objective is to minimize the processor utilization required to guarantee the stability of the plants. This could simply be formulated as follows for the proposed scheduling policy,

$$\begin{aligned} \min_{s_1, \dots, s_n, \bar{\delta}_1, \dots, \bar{\delta}_n} \quad & \sum_{i=1}^n u_i \cdot s_i + \epsilon \cdot \frac{c_i \cdot (1 - u_i \cdot s_i)}{\bar{\delta}_i} \\ \text{s.t.} \quad & c_i + (2 \cdot a_i - 1) \cdot \bar{\delta}_i - b_i \cdot u_i \cdot s_i \leq 0, \quad i = 1 \dots n, \end{aligned}$$

where ϵ is the switching overhead. We assume that the overhead is proportional to $\frac{c_i}{c_i'} = \frac{c_i \cdot (1 - u_i)}{\bar{\delta}_i}$ (see Section 5.4), i.e., the execution time of each subjob.

The problem could then be reformulated as n smaller problems for each task τ_i ,

$$\begin{aligned} \min_{s_i, \bar{\delta}_i} \quad & u_i \cdot s_i + \epsilon \cdot \frac{c_i \cdot (1 - u_i \cdot s_i)}{\bar{\delta}_i} \\ \text{s.t.} \quad & c_i + (2 \cdot a_i - 1) \cdot \bar{\delta}_i - b_i \cdot u_i \cdot s_i \leq 0. \end{aligned}$$

Since the focus is on task τ_i , for the sake of presentation, we drop the index i in the following.

To solve each problem, the KKT (Karush-Kuhn-Tucker) necessary conditions for optimality [BSS06] are considered. The optimum solution, denoted by $\mathbf{x}^*$, of the problem

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g(\mathbf{x}) \leq 0, \end{aligned} \quad (5.23)$$

must necessarily satisfy the following conditions

$$\begin{aligned} \nabla f(\mathbf{x}^*) + \mu^* \nabla g(\mathbf{x}^*) &= \mathbf{0}, \\ \mu^* g(\mathbf{x}^*) &= 0, \\ \mu^* &\geq 0. \end{aligned} \quad (5.24)$$

The following equalities are obtained based on the KKT conditions,

$$u - \frac{\epsilon \cdot c \cdot u}{\bar{\delta}} - \mu \cdot b \cdot u = 0 \quad (5.25)$$

$$-\frac{\epsilon \cdot c \cdot (1 - u \cdot s)}{\bar{\delta}^2} + \mu \cdot (2 \cdot a - 1) = 0 \quad (5.26)$$

$$\mu \cdot (c + (2 \cdot a - 1) \cdot \bar{\delta} - b \cdot u \cdot s) = 0 \quad (5.27)$$

Considering Equation (5.26), if $\mu = 0$, then $u \cdot s = 1$ which is not a valid solution when the platform is shared. Therefore, without loss of generality, we consider $\mu > 0$, which leads to,

$$c + (2 \cdot a - 1) \cdot \bar{\delta} - b \cdot u \cdot s = 0, \quad (5.28)$$

if we consider Equation (5.27). This leads to

$$s = \frac{c + (2 \cdot a - 1) \cdot \bar{\delta}}{b \cdot u}. \quad (5.29)$$

From Equation (5.25), we find μ ,

$$\mu = \frac{\bar{\delta} - \epsilon \cdot c}{b \cdot \bar{\delta}}. \quad (5.30)$$

Substituting μ and s in Equation (5.26), we obtain a quadratic equation with its positive root as the optimal value of $\bar{\delta}$,

$$\bar{\delta}^* = \sqrt{\frac{\epsilon \cdot c \cdot (b - c)}{2 \cdot a - 1}}. \quad (5.31)$$

Substituting $\bar{\delta}^*$ in Equation (5.29), we obtain the optimal value of s ,

$$s^* = \frac{c}{b \cdot u} + \frac{(2 \cdot a - 1)}{b \cdot u} \sqrt{\frac{\epsilon \cdot c \cdot (b - c)}{2 \cdot a - 1}}. \quad (5.32)$$

Finally, the taskset is schedulable and all plants are guaranteed to be stable on a uniprocessor platform if

$$\sum_{i=1}^n u_i \cdot s_i^* + \epsilon \cdot \frac{c_i \cdot (1 - u_i \cdot s_i^*)}{\bar{\delta}_i^*} \leq 1. \quad (5.33)$$

Figure 5.3: Comparison between Jfair and server approaches.

5.6 Experimental Results

In this section, we present experimental results to demonstrate the efficiency of the proposed algorithm. In particular, the comparison between the Jfair algorithm and a server-based approach is performed.

To evaluate our proposed algorithm experimentally, we have generated 500 benchmarks with a number of control applications from 2 to 10. The plants considered are chosen from a database consisting of inverted pendulums, ball and beam processes, DC servos, and harmonic oscillators [ÅW97], [CLE⁺04]. The UUniFast algorithm [BB05] is used to generate a set of random tasks for a given utilization. The switching overhead is $\epsilon = r \cdot \min_{i=1 \dots n} \{c_i\}$, where r is a random variable uniformly distributed in the interval of $[0.01, 0.10]$.

We compare our proposed approach against a server-based approach based on (α, Δ) model [FM02], which maximizes the period of the server to reduce the number of preemptions. However, the approach based on the resource reservation mechanism essentially minimizes the bandwidth required for guaranteeing the stability of control applications. The experiments are repeated for several values of total task utilization ($\sum_{i=1}^n u_i$) and the results are shown in Figure 5.3. The metric used for this comparison is the relative quality, defined as $\left(\frac{N_{\text{Jfair}} - N_{\text{Server}}}{N_{\text{Jfair}}} \times 100 \right)$, where N_X is the number of benchmarks for which the approach X could find a valid solution, i.e., a solution for which all control applications are stable and the system is schedulable.

For the low task utilization, our approach outperforms the server-

based approach only in 10% of the benchmarks, in terms of number of valid solutions. This percentage increases to 59% for high task utilization, indicating that in 59% of the benchmarks, our proposed Jfair approach guarantees a stable and schedulable solution, while the approach based on the resource reservation mechanism fails to do so. The trend is also clearly visible, i.e., increasing the task utilization leads to an increase in the number of invalid solutions for the server-based approach, compared to Jfair.

5.7 Conclusions

It is well known that temporal behavior of the control tasks affects the control quality and stability of the plants. In particular, the latency and jitter are two decisive factors in analyzing and designing embedded control systems or cyber-physical systems. We have proposed an online scheduling policy to limit the variation in the response time of the tasks, while considering also the latency. The correctness of the algorithms is formally proved and the efficiency of the algorithms is evaluated both theoretically and experimentally. Finally, the results are used in a design optimization problem in order to guarantee the stability of the control applications, while minimizing the resource usage.

SELF-TRIGGERED CONTROLLERS AND HARD REAL-TIME GUARANTEES

IN the previous chapters, we focused on the widely-used periodic control paradigm. In this chapter, we extend the scope of the thesis and discuss the implementation issues of the novel self-triggered control scheme, i.e., whether any guarantees can be provided when the platform is shared with self-triggered controllers.

It is well known that event-triggered and self-triggered controllers implemented on dedicated platforms can provide the same performance as the traditional periodic controllers, while consuming considerably less bandwidth. However, since the majority of controllers are implemented by software tasks on shared platforms, on one hand, it might no longer be possible to grant access to the event-triggered controller upon request. On the other hand, due to the seemingly irregular requests from self-triggered controllers, other applications, while in reality schedulable, may be declared unschedulable, if not carefully analyzed. The schedulability and response-time analysis in the presence of self-triggered controllers is still an open problem and the topic of this chapter.

6.1 Introduction and Related Work

Self-triggered and event-triggered controllers are being actively considered as substitutes of traditional periodic controllers. As opposed to the traditional controllers where sampling is periodic in the time

domain, typically, the event-triggered and self-triggered controllers execute when the expected performance is about to be violated. This, in turn, leads to less resource usage compared to the traditional periodic controllers since the controllers execute only if it is necessary in order to guarantee the expected performance.

Today, many control applications are implemented on shared platforms, alongside other hard real-time applications. There has been considerable amount of research on event-triggered and self-triggered control mechanisms [ÅB99, Årz99, VFM03, HJC08, HSB08, MAT09, WL09, BMS13, VMB15]. In the case of self-triggered controllers, however, *the lack of clear execution patterns* has been the main obstacle in efficiently implementing such applications alongside hard real-time applications on shared platforms. Due to the seemingly irregular requests from self-triggered controllers, current practice often leads to under-utilized resources and over-provisioned designs, which defeats the purpose of the self-triggered control, i.e., less resource usage.

In this chapter, we discuss the fact that self-triggered controllers actually exhibit certain execution patterns when carefully examined. Note that the next triggering instant for the self-triggered controllers depends on the state of the plant. The core idea here is to capture the dependency between the states of the plant at each triggering point. This means that, for each initial state, the following states are not arbitrary, and exploiting this fact results in a less pessimistic analysis. This, in particular, is important from the schedulability point of view.

A naive approach to schedulability analysis is to consider the least inter-execution time with respect to all initial states the plant can be in. To perform schedulability analysis, then it is safe to consider the self-triggered controller as a periodic task with the least inter-execution time. However, this is an overly pessimistic analysis since in every step it considers the worst-case possible state for the plant (with respect to inter-execution time). In this case, the calculated interference from the self-triggered controller is considerably larger than what occurs in reality, since always the worst-case scenario is considered, eliminating the potential advantage of self-triggered controllers versus the periodic controllers. And, essentially, this leads to a pessimistic analysis method.

Over the last few years, schedulability analysis of self-triggered controllers has gained attention. Velasco et al. [VMB08] consid-

ered this problem under both fixed-priority and earliest-deadline-first scheduling policies. However, the problem of finding the worst-case triggering pattern was left open. Lemmon et al. [LCHZ07] considered online scheduling of self-triggered controllers using elastic scheduling, but no stability guarantees were provided. Anta and Tabuada [AT09] discussed the benefits of relaxing periodicity constraint over communication networks. However, to provide schedulability guarantees, the authors considered the minimum inter-arrival time for all possible initial states, which is extremely pessimistic and defeats the purpose of the self-triggered control scheme. Antunes and Heemels [AH14] found the optimal sampling instants and control inputs in a given interval with respect to quadratic cost functions. However, they did not attempt to address the schedulability and response-time analysis problem. Finally, it has been shown that in certain self-triggered schemes no positive minimum inter-event time can be guaranteed [BH14].

In this chapter, we focus on the self-triggered approach proposed by Donkers et al. [DTH14], adapted for real-time analysis. We address the response-time and schedulability analysis for a mixed set of periodic hard real-time tasks and self-triggered control tasks. The basic idea is to make use of the fact that there actually exist certain patterns in the execution of self-triggered controllers. To our knowledge, this is the first attempt to perform offline schedulability analysis in the presence of self-triggered control tasks that allows to leverage the potential advantages of self-triggered control compared to periodic control.

6.2 System Model

6.2.1 Task Model

Given is an independent taskset, where each task is denoted by τ_i . The computation time (execution time) and priority of task τ_i are denoted by c_i and ρ_i , respectively. Task τ_i has higher priority than task τ_j iff $\rho_i > \rho_j$. The set of higher priority tasks for task τ_i is denoted by $hp(\tau_i)$.

The j^{th} instance (job) of task τ_i is denoted by $\tau_{i,j}$. The inter-arrival time (or inter-execution) between the two instances $\tau_{i,j}$ and $\tau_{i,j+1}$ is denoted by $h_{i,j}$. It is clear that for a periodic task τ_i , $h_{i,j} = h_{i,k}$, $\forall j, k$, which means that the inter-arrival time is constant for the

periodic tasks. Therefore, for the periodic task τ_i , we drop the index j for the period $h_{i,j}$ when convenient and denote the period by h_i .

The worst-case interference (in terms of the number of instances) caused by a task τ_i in an interval of length t is denoted by $I_i(t)$.

6.2.2 Plant Model

The plant associated with a self-triggered control task τ_i is modeled by a continuous-time system of differential equations [ÅW97],

$$\dot{\mathbf{x}}_i = \mathbf{A}_i \mathbf{x}_i + \mathbf{B}_i \mathbf{u}_i, \quad (6.1)$$

where $\mathbf{x}_i$ and $\mathbf{u}_i$ are the plant state and the control signal, respectively.

6.2.3 Self-Triggered Controller

In event-based control, the plant is constantly monitored and a new control input is applied only if the performance requirements of the plant are about to be violated. This is as opposed to the periodic scheme, where the plant is controlled uniformly in the time domain. The self-triggered control was first introduced in [VFM03]. In self-triggered control, as opposed to constant monitoring of the plant, at each execution instant, in addition to computing the control input, the controller also computes the latest instant at which a new control input should be applied in order to guarantee the required control performance. And, this is the next execution instant for the self-triggered controller.

The inter-execution time for a self-triggered controller depends on the current state of the plant, the dynamics of the plant, and the performance metrics used.

6.3 Problem Formulation

Given a mixed set of self-triggered control tasks and periodic hard real-time tasks, we would like to find out if all hard real-time tasks meet their deadlines and all plants associated with the self-triggered controllers are guaranteed to be stable, under the fixed-priority scheduling policy.

The main step towards schedulability analysis is to find the worst-case scenario of triggering of a single self-triggered controller. The worst-case scenario, in this context, refers to the triggering scenario of the controller that produces the maximum interference on other tasks.

6.4 The Self-Triggered Controller

In this section, we shall briefly discuss how our self-triggered scheme works. The approach comprises of an offline design time step which is prior to the actual execution of self-triggered control task, and an online step, where the next execution instant and the control input are determined at runtime.

6.4.1 Offline Step

At design time, the state space of the plant is partitioned into several convex polytopes. For each polytope, we shall calculate the maximum time that the plant could run in open-loop before instability (i.e., violating the expected performance), considering that the initial state could be anywhere in the polytope. Moreover, we shall find the polytopes in which the plant state could end up after it runs in open-loop for this amount of time. This information will be encoded in the form of a transition graph, where each node corresponds to one polytope. The transitions among the polytopes are captured by edges and the weight associated with each edge is the maximum time the plant could run in open-loop, assuming the initial state is within the source polytope of the edge. For instance, an edge from node p to node q with weight h indicates that: if the initial state of the plant is in polytope p , the plant can run in open-loop for h time units and the final state of the plant after h time units could be in polytope q . We discuss these techniques further in Sections 6.5 and 6.6.

6.4.2 Online Step

At runtime, the current state $\mathbf{x}(0)$ is known. Every time the self-triggered controller is executed, there are two procedures to be performed: (1) to compute the next time the self-triggered controller

needs to execute, and (2) to compute the constant control input until the next execution.

To this end, we shall first find the polytope to which the current state $\mathbf{x}(0)$ belongs. Note that there exist efficient algorithms to determine if a point is inside a convex polytope.

The corresponding control input and the next time the controller needs to execute for an initial state inside the polytope are obtained based on a slightly modified version of the self-triggered controller approach in [DTH14]. From the transition graph, we know that if the initial state is in a particular polytope, the trajectory can only end up in a subset of polytopes. Then, we shall solve the minimum attention control problem [DTH14], but enforcing extra constraints such that the final state of the plant after running in open-loop is guaranteed to be within this subset of polytopes. The problem remains a linear feasibility problem and, therefore, is of the same complexity order as the proposed approach in [DTH14].

It is of significant importance to observe that the maximum time h the plant can run in open-loop, which is calculated at runtime, may be longer or equal to what is indicated in the transition graph. That is, the actual interference of the self-triggered controller at runtime may not be larger than the interference found in the offline step, and hence the safety of our offline real-time analysis is preserved.

6.5 The Big Picture

Under fixed-priority preemptive scheduling, assuming constrained deadlines (deadlines less than or equal to the period) and an independent taskset with periodic tasks, the exact worst-case response time of a task τ_i , denoted by R_i^w , is computed by the following equation [JP86],

$$R_i^w = c_i^w + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{R_i^w}{h_j} \right\rceil c_j^w, \quad (6.2)$$

where $hp(\tau_i)$ denotes the set of higher priority tasks for task τ_i . Since Equation (6.2) cannot be solved analytically, it has to be solved by fixed-point iteration (starting with, e.g., $R_i^w = c_i^w$) and has pseudo-polynomial complexity. The above can be extended to periodic tasks with arbitrary deadlines [Leh90].

In Equation (6.2), since only periodic tasks are considered, we have $I_j(R_i^w) = \left\lceil \frac{R_i^w}{h_j} \right\rceil$. To consider the self-triggered tasks as well, Equation (6.2) can be rewritten as follows,

$$R_i^w = c_i^w + \sum_{\tau_j \in hp(\tau_i)} I_j(R_i^w) \cdot c_j^w, \quad (6.3)$$

where $I_j(t)$ is the maximum interference of the higher priority task τ_j in an interval of length t .

It should now be clear that the schedulability problem is reduced to finding the worst-case interference scenario in an interval of length t for a single self-triggered task τ_i , i.e., $I_i(t)$. In other words, we would like to find the *request-bound function* for each self-triggered controller. For the sake of presentation, in the next section, we shall only consider one single self-triggered task and, therefore, we can drop the index identifying the task.¹

6.6 Finding Request-Bound Function

In this section, we shall discuss the design time procedure to find the request-bound function for a single self-triggered control task. Towards this, first we shall divide the state space into several subregions. Then, it is determined if at runtime a transition from one subregion to another subregion is possible, and this information is modeled as a graph (see Section 6.6.1). The second step is to use dynamic programming in order to find the shortest interval of time with at least k triggering events, from which we compute the request-bound function (see Section 6.6.2).

6.6.1 Extraction of the Transition Graph

To find the transition graph, we shall take three steps described in the following subsections:

¹Note that there are no limiting assumptions on the number of self-triggered controllers or the priorities assigned to them.

6.6.1.1 Partitioning the State Space

In this step, we shall partition the state space of the plant into m convex polytopes. The main idea is to partition the state space such that each component of the state space has the same sign in the entire polytope and the dominating (maximum in terms of absolute value) component of the state space remains the same. For example, for the two-dimensional state space we have,

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{P}_{11} & \mathbf{P}_{12} \\ \mathbf{P}_{21} & \mathbf{P}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \mathbf{P}\mathbf{x}.$$

The polytopes are identified by the following lines:

$$\begin{aligned} \mathbf{y}_1 &= 0 \Rightarrow \mathbf{P}_{11}\mathbf{x}_1 + \mathbf{P}_{12}\mathbf{x}_2 = 0, \\ \mathbf{y}_2 &= 0 \Rightarrow \mathbf{P}_{21}\mathbf{x}_1 + \mathbf{P}_{22}\mathbf{x}_2 = 0, \\ \mathbf{y}_1 &= \mathbf{y}_2 \Rightarrow (\mathbf{P}_{11} - \mathbf{P}_{21})\mathbf{x}_1 + (\mathbf{P}_{12} - \mathbf{P}_{22})\mathbf{x}_2 = 0, \\ \mathbf{y}_1 &= -\mathbf{y}_2 \Rightarrow (\mathbf{P}_{11} + \mathbf{P}_{21})\mathbf{x}_1 + (\mathbf{P}_{12} + \mathbf{P}_{22})\mathbf{x}_2 = 0. \end{aligned}$$

The first two lines make sure that the sign of each component of vector $\mathbf{y}$ remains the same in each polytope, whereas the next two lines partition the state space such that in each polytope, the infinity norm of $\|\mathbf{y}\|_\infty$ always depends on one component of vector $\mathbf{y}$. The vertices of the polytopes are the origin and where the lines cross the boundary of the state space (see also Section 6.8 for an example). The generalization of the above partitioning technique to higher dimensions is trivial.

Although it is possible to partition the state space even further [SW13, AGDLB14], for the simplicity of presentation, we shall only consider this partitioning throughout this paper.

6.6.1.2 Calculation of the Maximum Time h for Each Polytope

For each polytope, we shall calculate the maximum time h that the plant could run in open-loop before instability (i.e., violating the expected control performance), considering that the initial state could be anywhere in the polytope. This is done based on a slightly modified version of the proposed approach in [DTH14].

The plant is guaranteed to be stable after running in open-loop for h time units if,

$$V(\mathbf{x}(h)) - e^{-\alpha h} V(\mathbf{x}(0)) \leq 0, \quad (6.4)$$

where $V(\cdot)$ denotes the Lyapunov function. Similar to [DTH14], here, we consider the Lyapunov function $V(\mathbf{x}) = \|\mathbf{P}\mathbf{x}\|_\infty$,

$$\|\mathbf{P}\mathbf{x}(h)\|_\infty - e^{-\alpha h} \|\mathbf{P}\mathbf{x}(0)\|_\infty \leq 0. \quad (6.5)$$

The plant state $\mathbf{x}$ at time h is as follows, assuming constant control input $\mathbf{u}$ in the interval $[0, h)$,

$$\begin{aligned} \mathbf{x}(h) &= e^{A h} \mathbf{x}(0) + \int_0^h e^{A(h-t)} dt \mathbf{B} \mathbf{u}, \\ &= \Phi(h) \mathbf{x}(0) + \Gamma(h) \mathbf{u}. \end{aligned} \quad (6.6)$$

where

$$\begin{aligned} \Phi(h) &= e^{A h}, \\ \Gamma(h) &= \int_0^h e^{A(h-t)} dt \mathbf{B}. \end{aligned}$$

To find the maximum time h where the plant could run in open-loop, for a given initial state $\mathbf{x}(0)$, h is increased iteratively until there does not exist any control input to satisfy inequality (6.5).

Let us assume for two vertices of the polytopes, namely $\underline{\mathbf{x}}(0)$ and $\bar{\mathbf{x}}(0)$, constraint (6.5) is satisfied if the system runs in open-loop for h time units,

$$\begin{aligned} \|\mathbf{P}(\Phi \underline{\mathbf{x}}(0) + \Gamma \underline{\mathbf{u}})\|_\infty - \sigma \|\mathbf{P} \underline{\mathbf{x}}(0)\|_\infty &\leq 0, \\ \|\mathbf{P}(\Phi \bar{\mathbf{x}}(0) + \Gamma \bar{\mathbf{u}})\|_\infty - \sigma \|\mathbf{P} \bar{\mathbf{x}}(0)\|_\infty &\leq 0, \end{aligned} \quad (6.7)$$

with constant $\sigma(h) = e^{-\alpha h}$.

Now we should show that for $\mathbf{x}(0) = \lambda \underline{\mathbf{x}}(0) + (1 - \lambda) \bar{\mathbf{x}}(0)$, with $0 \leq \lambda \leq 1$, constraint (6.5) is satisfied, i.e.,

$$\|\mathbf{P}(\Phi \mathbf{x}(0) + \Gamma \mathbf{u})\|_\infty - \sigma \|\mathbf{P} \mathbf{x}(0)\|_\infty \leq 0. \quad (6.8)$$

Note that, in general, the above inequality does not hold. However, within each polytope, thanks to the careful partitioning in the first step, we have,

$$\begin{aligned} \|\mathbf{P} \mathbf{x}(0)\|_\infty &= \|\mathbf{P}(\lambda \underline{\mathbf{x}}(0) + (1 - \lambda) \bar{\mathbf{x}}(0))\|_\infty \\ &= \lambda \|\mathbf{P} \underline{\mathbf{x}}(0)\|_\infty + (1 - \lambda) \|\mathbf{P} \bar{\mathbf{x}}(0)\|_\infty. \end{aligned} \quad (6.9)$$

Let us assume that the control input is given by $\mathbf{u} = \lambda \underline{\mathbf{u}} + (1 - \lambda) \overline{\mathbf{u}}$. Using the triangular property of norms and Equation (6.9), we can show that inequality (6.8) is satisfied,

$$\begin{aligned} & \|P(\Phi \mathbf{x}(0) + \Gamma \mathbf{u})\|_\infty - \sigma \|P \mathbf{x}(0)\|_\infty \leq \\ & \lambda (\|P(\Phi \underline{\mathbf{x}}(0) + \Gamma \underline{\mathbf{u}})\|_\infty - \sigma \|P \underline{\mathbf{x}}(0)\|_\infty) + \\ & (1 - \lambda) (\|P(\Phi \overline{\mathbf{x}}(0) + \Gamma \overline{\mathbf{u}})\|_\infty - \sigma \|P \overline{\mathbf{x}}(0)\|_\infty) \leq 0. \end{aligned}$$

This implies that if the system can run in open-loop for h time units considering the initial state to be any of the vertices of the convex polytope, then for any initial state within the convex polytope also the system can run in open-loop for h time units.

6.6.1.3 Construction of the Transition Graph

In this step, we shall construct the graph $\mathbf{G}$ corresponding to the transitions between the polytopes. Since the systems considered in this paper are linear, the convex polytopes after the system runs in open-loop will be mapped to new convex polytopes.

These new polytopes are easily found by considering the dynamics of the system (6.6) for the vertices of the initial convex polytopes [BV04]. Note that the transition graph $\mathbf{G}(p, q) = h$, if the p^{th} polytope after h time unit, which was found in the previous step, has overlap with the q^{th} polytope. And $\mathbf{G}(p, q) = +\infty$, if there can be no transition from the p^{th} polytope to the q^{th} polytope.

6.6.2 Extraction of the Worst-Case Request Pattern

Having the transition graph, it is now possible to find the worst-case request-bound function. To this end, we shall use dynamic programming. Note that the length of the shortest interval of time with k triggers inside, denoted by $s(k)$, is obtained as follows:

$$\begin{aligned} s(k, p) &= \min_{q=1 \dots m} \{s(k-1, q) + \mathbf{G}(q, p)\}, \\ s(k) &= \min_{p=1 \dots m} \{s(k, p)\}, \end{aligned} \tag{6.10}$$

where $s(k, p)$ is the shortest path with k nodes, which ends in the p^{th} node of the graph. Equivalently, $s(k, p)$ is the shortest interval of time with k triggers, which ends in the p^{th} polytope. From the structure of

Algorithm 2 Worst-Case Response-Time Analysis R_i^w

```

1: Initialization:  $R_i^w = c_i^w$ ;  $t = 0$ ;
2: Initialization:  $k_j = 1$ ;  $s_j(1, p) = 0, \forall p, j$ ;
3: while  $t < R_i^w$  do
4:    $t = R_i^w$ ;
5:    $R_i^w = c_i^w$ ;
6:   for all  $\tau_j \in hp(\tau_i)$  do
7:     if  $\tau_j$  is periodic then
8:        $I_j = \left\lceil \frac{t}{h_j} \right\rceil$ ;
9:     else
10:      while  $s_j(k_j) \leq t$  do
11:         $k_j = k_j + 1$ ;
12:         $s_j(k_j, p) = \min_{q=1 \dots m_j} \{s_j(k_j - 1, q) + G_j(q, p)\}$ ;
13:         $s_j(k_j) = \min_{p=1 \dots m_j} \{s(k_j, p)\}$ ;
14:      end while
15:       $I_j = k_j - 1$ ;
16:    end if
17:     $R_i^w = R_i^w + I_j \cdot c_j^w$ ;
18:  end for
19: end while
20: return  $R_i^w$ 

```

Equation (6.10), it can be observed that this problem could be solved using dynamic programming.

The request-bound function $I(t)$ has to be calculated by computing the pseudo-inverse of $s(k)$,

$$I(t) = s^{-1}(t), \quad (6.11)$$

where $I(t)$ is the maximum number of requests in any interval of length t .

6.7 Schedulability Analysis

In this section, we shall perform schedulability analysis for the self-triggered controllers. Having found the request-bound function $I_i(t)$ for each self-triggered task τ_i , we shall now introduce Algorithm 2 to compute the worst-case response-time of a task. If all tasks have

Table 6.1: Example: taskset data

i	ρ_i	c_i^w	h_i	d_i	self-triggered/hard real-time
1	3	0.3	—	0.8	self-triggered
2	2	1.0	2.0	2.0	hard real-time
3	1	1.0	6.0	6.0	hard real-time

worst-case response-times less than their deadlines, the system is schedulable.

For each hard real-time task τ_i , execution time c_i , period h_i , deadline d_i , and the set of higher priority tasks $hp(\tau_i)$ are known. However, for each self-triggered control task τ_i , only execution time c_i , deadline d_i , and the set of higher priority tasks $hp(\tau_i)$ are known. The deadline d_i for a self-triggered task τ_i can be obtained as follows,

$$d_i = \min_{\forall p, q} \{G_i(p, q)\}. \quad (6.12)$$

This is based on the fact that each self-triggered job should complete its execution before the next triggering instant, and in the worst-case scenario, we should consider the minimum among all. Observe that there is no pessimism introduced by our approach in computing this deadline.

The worst-case response-time of task τ_i is computed as follows,

$$R_i^w = c_i^w + \sum_{\tau_j \in hp(\tau_i)} I_j(R_i^w) \cdot c_j^w. \quad (6.13)$$

For a periodic hard real-time task τ_j , the worst-case interference function is $I_j(t) = \left\lceil \frac{t}{h_j} \right\rceil$. For a self-triggered control task, $I_j(t)$ is calculated based on Equation (6.10) and Equation (6.11).

The difference between the proposed algorithm and the traditional response-time analysis algorithm for periodic tasks under fixed-priority analysis is in Lines 10–15, where we compute the number of triggers of the self-triggered controller (interference in terms of number of events) in an interval of length t . Basically, we increase the number of triggers, k , iteratively, until the length of the shortest interval including k triggers is larger than t . Note that Algorithm 2 has the dynamic programming problem in Equation (6.10) embedded in Lines 12–13 of Algorithm 2.

Figure 6.1: The state space partitioning (on the left) and the corresponding transition graph (on the right).

The algorithm has pseudo-polynomial complexity, similar to response-time analysis for periodic tasks under fixed-priority policy.

6.8 Illustrative Example

Let us consider a taskset consisting of three tasks $\mathbf{T} = \{\tau_1, \tau_2, \tau_3\}$. Task τ_1 is a self-triggered control task, has the highest priority and worst-case execution-time $c_1^w = 0.3$. Task τ_2 is a hard-real time task with period $h_2 = 2.0$ and worst-case execution-time $c_2^w = 1.0$. Task τ_3 is also a hard real-time task with period $h_3 = 6.0$ and worst-case execution-time $c_3^w = 1.0$, and has the lowest priority. This information is summarized in Table 6.1. While we consider only one self-triggered task for the simplicity of presentation, the approach is by no means limited to a single self-triggered task.

In this example, we would like to find the response-time of task τ_3 , i.e., to check if it is schedulable.

Towards this, we need to find the request-bound function for the self-triggered task τ_1 . The plant associated with the self-triggered controller is identified by the following matrices (see Equation (6.1)),

$$\mathbf{A} = \begin{bmatrix} 1 & 5 \\ 0 & 2 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Note that the eigenvalues of this plant are not in the left half of the complex plane and, therefore, the plant is unstable. Finally, we assume that the state space is bounded, i.e., $\|\mathbf{x}\|_\infty \leq 1$.

Figure 6.2: The worst-case triggering (arrival) pattern of the self-triggered controller.

As discussed in Section 6.6.1, first we partition the space into convex polytopes, as shown in Figure 6.1. Secondly, for each polytope, we shall find the maximum time the plant could run in open-loop before instability. Thirdly, from this information, we can construct the transition graph. It turns out that the transition graph is as follows,

$$G = \begin{bmatrix} +\infty & 1.1 & +\infty & +\infty \\ +\infty & 1.1 & +\infty & +\infty \\ 0.8 & 0.8 & +\infty & +\infty \\ 0.9 & 0.9 & +\infty & +\infty \end{bmatrix}.$$

The graph corresponding to the above transition matrix is shown in Figure 6.1. For example, for node 3 (corresponding to polytope 3) of the graph, it can be observed that the plant, in the worst-case, could only run in open-loop for 0.8 time units and after that the trajectory will end up in node 1 (corresponding to polytope 1) or node 2 (corresponding to polytope 2).

Now, using the dynamic programming algorithm discussed in Section 6.6.2, we can find the worst-case request pattern. The worst-case request-bound function (or trigger pattern) is shown in Figure 6.2. Already at this stage, it is obvious that considering the worst-case scenario with respect to all initial states is in fact very conservative. To clarify this, note that in the worst-case scenario, the minimum inter-arrival time is 0.8 time units. However, from the transition graph, it is clear that this worst-case scenario can only occur once.

Let us now compute the worst-case response time of task τ_3 :

$$\begin{aligned}
 R_3^{w0} &= c_3^w = 1; \\
 R_3^{w1} &= c_3^w + \sum_{\tau_j \in hp(\tau_i)} I_j(R_3^{w0}) \cdot c_j^w = 1 + 2 \cdot 0.3 + 1 \cdot 1 = 2.6, \\
 R_3^{w2} &= c_3^w + \sum_{\tau_j \in hp(\tau_i)} I_j(R_3^{w1}) \cdot c_j^w = 1 + 3 \cdot 0.3 + 2 \cdot 1 = 3.9, \\
 R_3^{w3} &= c_3^w + \sum_{\tau_j \in hp(\tau_i)} I_j(R_3^{w2}) \cdot c_j^w = 1 + 4 \cdot 0.3 + 2 \cdot 1 = 4.2, \\
 R_3^{w4} &= c_3^w + \sum_{\tau_j \in hp(\tau_i)} I_j(R_3^{w3}) \cdot c_j^w = 1 + 5 \cdot 0.3 + 3 \cdot 1 = 5.5, \\
 R_3^{w5} &= c_3^w + \sum_{\tau_j \in hp(\tau_i)} I_j(R_3^{w4}) \cdot c_j^w = 1 + 6 \cdot 0.3 + 3 \cdot 1 = 5.8, \\
 R_3^{w6} &= c_3^w + \sum_{\tau_j \in hp(\tau_i)} I_j(R_3^{w5}) \cdot c_j^w = 1 + 6 \cdot 0.3 + 3 \cdot 1 = 5.8.
 \end{aligned}$$

The worst-case response-time of task τ_3 is $R_3^w = 5.8$ and, therefore, the task meets its deadline $d_3 = 6.0$. This scenario is shown in Figure 6.3. The green task is the self-triggered task τ_1 , the red task is periodic hard real-time task τ_2 , and the blue task is τ_3 for which we would like to find the worst-case response-time.

Lastly, let us also consider the state of the art approach which considers the worst-case inter-arrival time with respect to all initial states in every step, i.e., to ignore the dependency between states. In this approach, the self-triggered task is modeled as a periodic task with period $h_1 = \min_{p,q} \{G(p,q)\} = 0.8$. Based on the real-time schedulability analysis for periodic tasks, in this case, the worst-case response-time of task τ_3 is not finite, i.e., it misses its deadline $d_3 = 6.0$ and the system is deemed unschedulable. The system designer in such situations either needs to, unnecessarily, remove some of the tasks to guarantee schedulability or, again unnecessarily, to use a processor which is faster. Either way, this leads to an over-provisioned design and under-utilized resource.

This example demonstrates the importance of performing tight schedulability analysis in the presence of self-triggered controllers and the efficiency of our proposed approach. Note that even though we considered a single self-triggered control task at the highest priority level, our approach is by no means limited to this case and there is

Figure 6.3: The worst-case response time of task τ_3 .

no restricting assumption on the number of self-triggered controllers or the priorities assigned to these controllers.

In the original self-triggered schemes, it is often assumed that the computation of the control input and the next activation instant is instantaneous. However, this is different from what occurs in practice. To account for the delay experienced by each self-triggered task, at each execution, the self-triggered task computes the plant state at the next triggering instant based on the dynamics of the systems and current control input. Then, based on the plant state at the next triggering instant, the control input after the next triggering instant and the amount of time the plant could run in open-loop after the next triggering instant are calculated.

6.9 Conclusions

The lack of efficient schedulability analysis of real-time systems in the presence of self-triggered controllers has been an obstacle in implementing such applications alongside hard real-time applications on shared platforms. In this paper, we have proposed an approach for response-time analysis in the presence of self-triggered control tasks, under the fixed-priority scheduling policy. The proposed approach can be extended to other scheduling policies (e.g., earliest-deadline-first) and task models (e.g., digraph or arbitrary deadlines) [Leh90, SGY14].

CONCLUSIONS AND FUTURE WORK

IN this chapter, we shall summarize the research contributions of this thesis and discuss possible future research directions.

7.1 Conclusions

The thesis explains the intricate relation between the control theory and real-time scheduling areas and assists engineers in high quality, stable, compositional, and safe implementation of embedded control systems. In particular, the thesis proposes systematic offline design methodologies for embedded control systems as well as an on-line scheduling algorithm. Our main conclusion is that systematic control-scheduling co-design techniques will improve the design of embedded control and cyber-physical systems.

Chapter 2 illustrates the challenges of correct implementation of embedded control systems. More specifically, the guarantees provided at design time might not be preserved for the final implementation, if the implementation impacts are not considered at the design stage.

Offline design methodologies are discussed in Chapters 3, 4, and 6, while Chapter 5 proposes an online stabilizing scheduling algorithm.

In Chapter 3, we discuss that the design should be driven by the expected performance, while the worst-case stability guarantees are also considered. Ignoring the worst-case scenario results in instability of the plant, while a design mainly based on the worst-case performance leads to either a poor expected control performance or resource under-utilization.

In Chapter 4, we discuss that a design methodology based on the virtualization idea enjoys compositionality and isolation and facilitates systematic design and optimization. Note that this is orthogonal to Chapter 3, i.e., the ideas can be combined in one single design methodology.

In Chapter 6, we extend the scope of the thesis to the scenario where the platform is shared with self-triggered controllers, in addition to periodic controllers and hard real-time tasks.

In Chapter 5, we go beyond offline design methodologies and propose an online scheduling policy, to stabilize control applications, with limited scheduling overheads.

7.2 Future Research Directions

The material presented in this thesis can be the basis for several future research directions:

- The extension of the proposed approaches to multiprocessor and distributed platforms is an interesting research direction, since many automotive applications are implemented on distributed platforms.
- As opposed to the traditional embedded systems, cyber-physical systems are often tightly interacting with secondary networks and components, which makes it easier for an adversary to compromise the safety of the system. On the other hand, many such systems comprise several critical physical components, e.g., in automotive systems, adaptive cruise control or engine control allow deep intervention in the driving of a vehicle. Therefore, ignoring the security aspects in cyber-physical systems will have severe consequences. This advocates the need for new methodologies to ensure the security of such systems.

By the interconnection of the physical systems to the cyber (processing) elements, the notion of physical time is introduced in today's embedded systems. This opens the door for a new class of attacks, which we refer to as *temporal attacks*. In temporal attacks, the correct behavior of the system is targeted through manipulation of its temporal properties. In the case of control applications, such attacks may lead to a larger jitter

and instability of the physical plants. Extending results from the control–scheduling co-design area could lead to efficient resource allocation techniques to counteract temporal attacks on control applications.

Another fundamental aspect of cyber-physical systems is the existence of a mathematical model of the physical process. Using this model of the physical plant in cyber-physical systems, the intrusion detection can be done rather systematically. In short, any major discrepancy between the observed behavior and the expected behavior (according to the mathematical model) is an indication of intrusion. In addition to detection, it is also possible to propose a response mechanism, again in a systematic fashion, by allocating more resources to the task under intrusion. In the response mechanism, a careful resource allocation mechanism can be used to mitigate the effect of an attack. The intuition is that allocating more bandwidth to a control task, e.g., by using a shorter sampling period, often leads to a better control performance and stability margins.

Finally, the overhead introduced by encryption and decryption for the packets sent over the network has a direct impact on the delay experienced by control applications. Hence the trade-off between the security level and quality of control. Note that in real-time systems, the delay is not important as long as it does not exceed a certain threshold (often known as deadline), while in control applications the actual value of the delay has a significant impact on control quality and stability.

- In Chapter 6, the proposed approach in [DTH14] is considered. Further investigation is needed to find out a wider class of self-triggered controllers for which the proposed analysis can be applied.

In Chapter 6, the proposed approach in [DTH14] is minimally modified to support offline schedulability analysis. However, new self-triggered control schemes can be proposed to support the offline schedulability analysis in this thesis. For instance, in Chapter 6, the self-triggered control co-design is modeled as a feasibility problem, while in practice a convex objective function could also be considered in the optimization, without

introducing additional complexity.

Finally, towards a more efficient schedulability analysis technique, it is possible to consider the trajectory of a polytope after several steps. In Chapter 6, we only consider the trajectory after one step.

BIBLIOGRAPHY

- [ÅB99] Karl Johan Åström and Bo Bernhardsson. Comparison of periodic and event-based sampling for first-order stochastic systems. In *Preprints of the 14th World Congress of IFAC*, 1999.
- [ABEP13] Amir Aminifar, Enrico Bini, Petru Eles, and Zebo Peng. Designing bandwidth-efficient stabilizing control servers. In *Proceedings of the 34th IEEE Real-Time Systems Symposium*, 2013.
- [ABEP14] Amir Aminifar, Enrico Bini, Petru Eles, and Zebo Peng. Bandwidth-efficient controller–server co-design with stability guarantees. In *Proceedings of the 17th Conference for Design, Automation and Test in Europe (DATE)*, 2014.
- [ABEP15] Amir Aminifar, Enrico Bini, Petru Eles, and Zebo Peng. Analysis and desing of real-time servers for control applications. *IEEE Transactions on Computers*, 2015.
- [ÅC05] Karl Erik Årzén and Anton Cervin. Control and embedded computing: Survey of research directions. In *Proceedings of the 16th IFAC World Congress*, 2005.
- [ÅCES00] Karl Erik Årzén, Anton Cervin, Johan Eker, and Lui Sha. An introduction to control and scheduling co-

- design. In *Proceedings of the 39th IEEE Conference on Decision and Control*, pages 4865–4870, 2000.
- [AEP15] Amir Aminifar, Petru Eles, and Zebo Peng. Jfair: A scheduling algorithm to stabilize control applications. In *Proceedings of the 21st IEEE Real-Time and Embedded Technology and Applications Symposium*, 2015.
- [AEPC13a] Amir Aminifar, Petru Eles, Zebo Peng, and Anton Cervin. Control-quality driven design of cyber-physical systems with robustness guarantees. In *Proceedings of the 16th Conference for Design, Automation and Test in Europe (DATE)*, 2013.
- [AEPC13b] Amir Aminifar, Petru Eles, Zebo Peng, and Anton Cervin. Stability-aware analysis and design of embedded control systems. In *Proceedings of the International Conference on Embedded Software (EMSOFT)*, pages 1–10, 2013.
- [AGDLB14] Ebru Aydin Gol, Xuchu Ding, Mircea Lazar, and Calin Belta. Finite bisimulations for switched linear systems. *IEEE Transactions on Automatic Control*, 59(12):3122–3134, 2014.
- [AH14] D. Antunes and W.P.M.H. Heemels. Rollout event-triggered control: Beyond periodic control performance. *IEEE Transactions on Automatic Control*, 59(12):3296–3311, Dec 2014.
- [AP04] Luis Almeida and Paulo Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proceedings of the 4th ACM international conference on Embedded software*, pages 95–103, 2004.
- [Årz99] Karl Erik Årzén. A simple event-based pid controller. In *Preprints of the 14th World Congress of IFAC*, 1999.
- [ASE⁺12] Amir Aminifar, Soheil Samii, Petru Eles, Zebo Peng, and Anton Cervin. Designing high-quality embedded

- control systems with guaranteed stability. In *Proceedings of the 33th IEEE Real-Time Systems Symposium*, pages 283–292, 2012.
- [ASEP11] Amir Aminifar, Soheil Samii, Petru Eles, and Zebo Peng. Control-quality driven task mapping for distributed embedded control systems. In *Proceedings of the 17th IEEE Embedded and Real-Time Computing Systems and Applications (RTCSA) Conference*, pages 133–142, 2011.
- [AT09] Adolfo Anta and Paulo Tabuada. On the benefits of relaxing the periodicity assumption for networked control systems over CAN. In *Proceedings of the 30th IEEE Real-Time Systems Symposium*, pages 3–12, 2009.
- [ATEP16] Amir Aminifar, Paulo Tabuada, Petru Eles, and Zebo Peng. Self-triggered controllers and hard real-time guarantees. In *Proceedings of the 19th Conference for Design, Automation and Test in Europe (DATE)*, 2016.
- [Aud91] Neil C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS 164, Department of Computer Science, University of York, December 1991.
- [ÅW97] Karl Johan Åström and Björn Wittenmark. *Computer-Controlled Systems*. Prentice Hall, 3 edition, 1997.
- [BAEP14] Unmesh D. Bordoloi, Amir Aminifar, Petru Eles, and Zebo Peng. Schedulability analysis of ethernet avb switches. In *IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–10, 2014.
- [BB05] Enrico Bini and Giorgio C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

- [BB06] Giorgio Buttazzo and Enrico Bini. Optimal dimensioning of a constant bandwidth server. In *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 169–177, 2006.
- [BBB14] Giorgio C. Buttazzo, Enrico Bini, and Darren Buttle. Rate-adaptive tasks: Model, analysis, and design issues. In *Proceedings of the 17th Conference for Design, Automation and Test in Europe (DATE)*, pages 1–6, 2014.
- [BBGL99] Sanjoy Baruah, Giorgio C. Buttazzo, Sergey Gorinsky, and Giuseppe Lipari. Scheduling periodic task systems to minimize output jitter. In *Proceedings of the 6th IEEE Embedded and Real-Time Computing Systems and Applications (RTCSA) Conference*, pages 62–69, 1999.
- [BC08] Enrico Bini and Anton Cervin. Delay-aware period assignment in control systems. In *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 291–300, 2008.
- [BCH06] M. M. Ben Gaid, A. Cela, and Y. Hamam. Optimal integrated control and scheduling of networked control systems with communication constraints: Application to a car suspension system. *IEEE Transactions on Control Systems Technology*, 14(4):776–787, 2006.
- [BCPV96] Sanjoy K. Baruah, Neil K. Cohen, C. Greg Plaxton, and Donald A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.
- [BH14] D.P. Borgers and W.P.M.H. Heemels. Event-separation properties of event-triggered control systems. *IEEE Transactions on Automatic Control*, 59(10):2644–2656, 2014.
- [BHCNRB09] Enrico Bini, Thi Huyen Châu Nguyen, Pascal Richard, and Sanjoy K. Baruah. A response-time bound

- in fixed-priority scheduling with arbitrary deadlines. *IEEE Transactions on Computer*, 58(2):279–286, 2009.
- [BI07] Moris Behnam and Damir Isovici. Real-time control and scheduling co-design for efficient jitter handling. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 516–524, 2007.
- [BLCA02] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302, 2002.
- [BLM13] Reinder J. Bril, Johan J. Lukkien, and Rudolf H. Mak. Best-case response times and jitter analysis of real-time tasks with arbitrary deadlines. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, pages 193–202, 2013.
- [BMS13] Tobias Bund, Benjamin Menhorn, and Frank Slomka. Event density analysis for event triggered control systems. In *Proceedings of the 16th Conference for Design, Automation and Test in Europe (DATE)*, pages 1111–1116, 2013.
- [BPZ02] M.S. Branicky, S.M. Phillips, and Wei Zhang. Scheduling and feedback co-design for networked control systems. In *Proceedings of the 41st IEEE Conference on Decision and Control*, volume 2, pages 1211–1217, 2002.
- [BS13] Tobias Bund and Frank Slomka. A delay density model for networked control systems. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, pages 205–212, 2013.
- [BS15] Tobias Bund and Frank Slomka. Worst-case performance validation of safety-critical control systems

- with dropped samples. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, pages 319–326, 2015.
- [BSS06] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty. *Non-linear Programming: Theory and Algorithms*. Wiley, 2006.
- [BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.
- [BVMF04] Giorgio C. Buttazzo, Manel Velasco, Pau Marti, and Gerhard Fohler. Managing quality-of-control performance under overload conditions. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 53–60, 2004.
- [CE05] Anton Cervin and Johan Eker. Control-scheduling codesign of real-time systems: The control server approach. *Journal of Embedded Computing*, 1(2):209–224, 2005.
- [CEBÅ02] Anton Cervin, Johan Eker, Bo Bernhardsson, and Karl-Erik Årzén. Feedbac-feedforward scheduling of control tasks. *Real-Time Systems*, 23(1-2):25–53, 2002.
- [Cer12] Anton Cervin. Stability and worst-case performance analysis of sampled-data control systems with input and output jitter. In *Proceedings of the 2012 American Control Conference (ACC)*, 2012.
- [CH08] Anton Cervin and Toivo Henningsson. Scheduling of event-triggered controllers on a shared network. In *Proceedings of the 47th Conference on Decision and Control*, pages 3601–3606, 2008.
- [CHL⁺03] Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, and Karl Erik Årzén. How does control timing affect performance? Analysis and simulation of

- timing using Jitterbug and TrueTime. *IEEE Control Systems Magazine*, 23(3):16–30, 2003.
- [CLE⁺04] Anton Cervin, Bo Lincoln, Johan Eker, Karl Erik Årzén, and Giorgio Buttazzo. The jitter margin and its application in the design of real-time control systems. In *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications*, 2004.
- [Con05] FlexRay Consortium. Flexray communications system, protocol specification version 2.1, 2005.
- [CVMC11] Anton Cervin, Manel Velasco, Pau Marti, and Antonio Camacho. Optimal online sampling period assignment: Theory and experiments. *IEEE Transactions on Control Systems Technology*, 19(4):902–910, 2011.
- [DB07] Robert I. Davis and Alan Burns. Robust priority assignment for fixed priority real-time systems. In *Proceedings of the 28th IEEE Real-Time Systems Symposium*, pages 3–14, 2007.
- [DKGT14] Neil Dhruva, Pratyush Kumar, Georgia Giannopoulou, and Lothar Thiele. Computing a language-based guarantee for timing properties of cyber-physical systems. In *Proceedings of the 17th Conference for Design, Automation and Test in Europe (DATE)*, 2014.
- [Doy78] John C. Doyle. Guaranteed margins for lqg regulators. *IEEE Transactions on Automatic Control*, 23(4):756–757, 1978.
- [DTH14] M.C.F. Donkers, P. Tabuada, and W.P.M.H. Heemels. Minimum attention control for linear systems. *Discrete Event Dynamic Systems*, 24(2):199–218, 2014.
- [EAL07] Arvind Easwaran, Madhukar Anand, and Insup Lee. Compositional analysis framework using edp resource models. In *Proceedings of the 28th IEEE Real-Time Systems Symposium*, pages 129–138, 2007.

- [EHÅ00] Johan Eker, Per Hagander, and Karl-Erik Årzén. A feedback scheduler for real-time controller tasks. *Control Engineering Practice*, 8(12):1369–1378, 2000.
- [FD12] Nathan Fisher and Farhana Dewan. A bandwidth allocation scheme for compositional real-time systems with periodic resources. *Real-Time Systems*, 48(3):223–263, 2012.
- [FGP13] Daniele Fontanelli, Luca Greco, and Luigi Palopoli. Soft real-time scheduling for embedded control systems. *Automatica*, 49(8):2330 – 2338, 2013.
- [FM02] Xiang Feng and A.K. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the 23th IEEE Real-Time Systems Symposium*, pages 26–35, 2002.
- [FPA13] Daniel Fontanelli, Luigi Palopoli, and Luca Abeni. The continuous stream model of computation for real-time control. In *Proceedings of the 34th IEEE Real-Time Systems Symposium, Vancouver, Canada*, 2013.
- [FPG13] Daniele Fontantelli, Luigi Palopoli, and Luca Greco. Optimal cpu allocation to a set of control tasks with soft real-time execution constraints. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 233–242, 2013.
- [GLSC12] Dip Goswami, Martin Lukasiewicz, Reinhard Schneider, and Samarjit Chakraborty. Time-triggered implementations of mixed-criticality automotive software. In *Proceedings of the 15th Conference for Design, Automation and Test in Europe (DATE)*, 2012.
- [GSC11a] Dip Goswami, Reinhard Schneider, and Samarjit Chakraborty. Co-design of cyber-physical systems via controllers with flexible delay constraints. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 225–230, 2011.

- [GSC11b] Dip Goswami, Reinhard Schneider, and Samarjit Chakraborty. Re-engineering cyber-physical control applications for hybrid communication protocols. In *Proceedings of the 14th Conference for Design, Automation and Test in Europe (DATE)*, pages 1–6, 2011.
- [HC05] Dan Henriksson and Anton Cervin. Optimal on-line sampling period assignment for real-time control tasks based on plant state information. In *Proceedings of the 44th IEEE Conference on Decision and Control and the European Control Conference*, pages 4469–4474. IEEE Press, 2005.
- [HHL10] Shengyan Hong, Xiaobo Sharon Hu, and Michael Lemmon. Reducing delay jitter of real-time control tasks through adaptive deadline adjustments. In *Euromicro Conference on Real-Time Systems*, pages 229–238, 2010.
- [HJ61] Robert Hooke and T. A. Jeeves. “direct search” solution of numerical and statistical problems. *J. ACM*, 8(2):212–229, 1961.
- [HJC08] Toivo Henningsson, Erik Johannesson, and Anton Cervin. Sporadic event-based control of first-order linear stochastic systems. *Automatica*, 44(11):2890–2895, 2008.
- [HSB08] W. P. M. H. Heemels, J. H. Sandee, and P. P. J. Van Den Bosch. Analysis of event-driven controllers for linear systems. *International Journal of Control*, 81(4):571–590, 2008.
- [JP86] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.
- [KGC⁺12] Pratyush Kumar, Dip Goswami, Samarjit Chakraborty, Anuradha Annaswamy, Kai Lampka,

- and Lothar Thiele. A hybrid approach to cyber-physical systems verification. In *Proceedings of the 49th Design Automation Conference*, 2012.
- [KKH⁺08] Xenofon Koutsoukos, Nicholas Kottenstette, Joe Hall, Panos Antsaklis, and Janos Sztinapovits. Passivity-based control design of cyber-physical systems. In *International Workshop on Cyber-Physical Systems - Challenges and Applications*, 2008.
- [KL04] Chung-Yao Kao and Bo Lincoln. Simple stability criteria for systems with time-varying delays. *Automatica*, 40:1429–1434, 2004.
- [Kop97] Hermann Kopetz. *Real-Time Systems—Design Principles for Distributed Embedded Applications*. Kluwer Academic, 1997.
- [LB03] Giuseppe Lipari and Enrico Bini. Resource partitioning among real-time applications. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 151–158, 2003.
- [LC02] Bo Lincoln and Anton Cervin. Jitterbug: A tool for analysis of real-time control performance. In *Proceedings of the 41st IEEE Conference on Decision and Control*, pages 1319–1324, 2002.
- [LCHZ07] Michael Lemmon, Thidapat Chantem, Xiaobo Sharon Hu, and Matthew Zyskowski. On self-triggered full-information h-infinity controllers. In *Proceedings of the 10th International Conference on Hybrid Systems: Computation and Control*, pages 371–384, 2007.
- [Leh90] John P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pages 201–209, 1990.
- [LL73] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):47–61, 1973.

- [LW82] J. Y. T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982.
- [MAT09] Manuel Mazo, Adolfo Anta, and Paulo Tabuada. On self-triggered control for linear systems: Guarantees and complexity. In *2009 European Control Conference (ECC)*, pages 3767–3772, 2009.
- [MBP11] Giulio Mancuso, Enrico Bini, and Gabriele Pannocchia. Optimal computational resource allocation for control task under fixed priority scheduling. *Proceedings of the 18th IFAC World Congress*, 18, 2011.
- [MBP14] Giulio Mancuso, Enrico Bini, and Gabriele Pannocchia. Optimal priority assignment to control tasks. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(5s):161, 2014.
- [MF01] Aloysius K. Mok and Xiang (Alex) Feng. Towards compositionality in real-time resource partitioning based on regularity bounds. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, pages 129–138, 2001.
- [MGC⁺12] Alejandro Masrur, Dip Goswami, Samarjit Chakraborty, Jian-Jia Chen, Anuradha Annaswamy, and Ansuman Banerjee. Timing analysis of cyber-physical applications for hybrid communication protocols. In *Proceedings of the 15th Conference for Design, Automation and Test in Europe (DATE)*, pages 1233–1238, 2012.
- [MHRE05] Bren Mochocki, Xiaobo Sharon Hu, Razvan Racu, and Rolf Ernst. Dynamic voltage scaling for the schedulability of jitter-constrained real-time embedded systems. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 446–449, 2005.
- [MLB⁺04] Pau Marti, Caixue Lin, Scott A. Brandt, Manel Velasco, and Josep M. Fuertes. Optimal state feedback

- based resource allocation for resource-constrained control tasks. In *Proceedings of the 25st IEEE Real-Time Systems Symposium*, pages 161–172, 2004.
- [MSZ11] Rupak Majumdar, Indranil Saha, and Majid Zamani. Performance-aware scheduler synthesis for control systems. In *Proceedings of the 9th ACM international conference on Embedded software*, pages 299–308, 2011.
- [NH08] Payam Naghshtabrizi and João Pedro Hespanha. Distributed control systems with shared communication and computation resources. Position paper for the National Workshop on High Confidence Automotive Cyber-Physical Systems, 2008.
- [NH09] Payam Naghshtabrizi and João Pedro Hespanha. Analysis of distributed control systems with shared communication and computation resources. In *Proceedings of the 2009 American Control Conference (ACC)*, 2009.
- [NPAG06] Truong Nghiem, George J. Pappas, Rajeev Alur, and Antoine Girard. Time-triggered implementations of dynamic controllers. In *Proceedings of the 6th ACM & IEEE International conference on Embedded software*, pages 2–11, 2006.
- [NS00] Marco Di Natale and John A. Stankovic. Scheduling distributed real-time tasks with minimum jitter. *IEEE Transactions on Computers*, 49(4):303–316, 2000.
- [NW99] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2 edition, 1999.
- [PGGGGH98] J.C. Palencia Gutierrez, J.J. Gutierrez Garcia, and M. Gonzalez Harbour. Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems. In *Proceedings of the 10th Euro-micro Workshop on Real-Time Systems*, pages 35–44, 1998.

- [PL13] Linh T. X. Phan and Insup Lee. Improving schedulability of fixed-priority real-time systems using shapers. In *Proceedings of the 19th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 217–226, 2013.
- [PPBSV05] Luigi Palopoli, Claudio Pinello, Antonio Bicchi, and Alberto Sangiovanni-Vincentelli. Maximizing the stability radius of a set of systems under real-time scheduling constraints. *Automatic Control, IEEE Transactions on*, 50(11):1790–1795, 2005.
- [RS00] Henrik Rehbinder and Martin Sanfridson. Integration of off-line scheduling and optimal control. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, pages 137–143, 2000.
- [RS02] Ola Redell and Martin Sanfridson. Exact best-case response time analysis of fixed priority scheduled tasks. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, pages 165–172, 2002.
- [SBE⁺12] Soheil Samii, Unmesh D. Bordoloi, Petru Eles, Zebo Peng, and Anton Cervin. Control-quality optimization for distributed embedded systems with adaptive fault tolerance. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems*, pages 68–77, 2012.
- [SBM15] Indranil Saha, Sanjoy Baruah, and Rupak Majumdar. Dynamic scheduling for networked control systems. In *Proceedings of the 28th International Conference on Hybrid Systems: Computation and Control*, pages 98–107, 2015.
- [SCEP09] Soheil Samii, Anton Cervin, Petru Eles, and Zebo Peng. Integrated scheduling and synthesis of control applications on distributed embedded systems. In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 57–62, 2009.

- [SEP⁺10] Soheil Samii, Petru Eles, Zebo Peng, Paulo Tabuada, and Anton Cervin. Dynamic scheduling and control-quality optimization of self-triggered control applications. In *Proceedings of the 31st IEEE Real-Time Systems Symposium*, pages 95–104, 2010.
- [SEPC09] Soheil Samii, Petru Eles, Zebo Peng, and Anton Cervin. Quality-driven synthesis of embedded multi-mode control systems. In *Proceedings of the 46th Design Automation Conference*, pages 864–869, 2009.
- [Ser72] Omri Serlin. Scheduling of time critical processes. In *Proceedings of AFIPS Spring Computing Conference*, pages 925–932, 1972.
- [SGY14] Martin Stigge, Nan Guan, and Wang Yi. Refinement-based exact response-time analysis. In *the 26th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 143–152, 2014.
- [SL03] Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*, pages 2–13, 2003.
- [SLSS96] Danbing Seto, John P. Lehoczky, Lui Sha, and Kang G. Shin. On task schedulability in real-time control systems. In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pages 13–21, 1996.
- [SRLK02] Saowanee Saewong, Ragunathan (Raj) Rajkumar, John P. Lehoczky, and Mark H. Klein. Analysis of hierarchical fixed-priority scheduling. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, pages 152–160, 2002.
- [SW13] Christoffer Sloth and Rafael Wisniewski. Complete abstractions of dynamical systems by timed automata. *Nonlinear Analysis: Hybrid Systems*, 7(1):80–100, 2013.

- [TBW94] Ken Tindell, Alan Burns, and Andy J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.
- [VFM03] Manel Velasco, Josep M. Fuertes, and Pau Marti. The self-triggered task model for real-time control systems. In *the 24th IEEE Real-Time Systems Symposium*, pages 67–70, 2003.
- [VMB08] Manel Velasco, Pau Marti, and Enrico Bini. Control-driven tasks: Modeling and analysis. In *Real-Time Systems Symposium, 2008*, pages 280–290, 2008.
- [VMB15] Manel Velasco, Pau Marti, and Enrico Bini. Optimal-sampling-inspired self-triggered control. In *the IEEE International Conference on Event-based Control, Communications & Signal Processing*, 2015.
- [VZF91] Dinesh C. Verma, Hui Zhang, and Domenico Ferrari. Delay jitter control for real-time communication in a packet switching network. In *Proceedings of the IEEE Conference on Communications Software: Communications for Distributed Applications and Systems*, pages 35–43, 1991.
- [WA07] Gera Weiss and Rajeev Alur. Automata based interfaces for control and scheduling. In *Hybrid Systems: Computation and Control*, volume 4416 of *Lecture Notes in Computer Science*, pages 601–613. Springer Berlin Heidelberg, 2007.
- [WBSS11a] M. Westmijze, M.J.G. Bekooij, G.J.M. Smit, and M. Schrijver. Evaluation of scheduling heuristics for jitter reduction of real-time streaming applications on multi-core general purpose hardware. In *Proceedings of the 9th IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, pages 140–146, 2011.
- [WBSS11b] M. Westmijze, M.J.G. Bekooij, G.J.M. Smit, and M. Schrijver. Evaluation of scheduling heuristics for

- jitter reduction of real-time streaming applications on multi-core general purpose hardware. In *Proceedings of the 9th IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, pages 140–146, Oct 2011.
- [WL09] Xiaofeng Wang and Michael Lemmon. Self-triggered feedback control systems with finite-gain stability. *IEEE Transactions on Automatic Control*, 54(3):452–467, 2009.
- [WNT95] Björn Wittenmark, Johan Nilsson, and Martin Törn-gren. Timing problems in real-time control systems. In *Proceedings of the American Control Conference*, pages 2000–2004, 1995.
- [XÅBC14] Yang Xu, Karl Erik Årzén, Enrico Bini, and Anton Cervin. Response time driven design of control systems. In *19th IFAC World Congress*, Cape Town, South Africa, August 2014.
- [XÅC⁺15] Yang Xu, Karl Erik Årzén, Anton Cervin, Enrico Bini, and Bogdan Tanasa. Exploiting job response-time information in the co-design of real-time control systems. In *IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 247–256, 2015.
- [ZSWM08] Fumin Zhang, Klementyna Szwaykowska, Wayne Wolf, and Vincent Mooney. Task scheduling for control oriented requirements for cyber-physical systems. In *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 47–56, 2008.

Department of Computer and Information Science
Linköpings universitet

Dissertations

Linköping Studies in Science and Technology
Linköping Studies in Arts and Science

Linköping Studies in Statistics

Linköpings Studies in Information Science

Linköping Studies in Science and Technology

- No 14 **Anders Haraldsson:** A Program Manipulation System Based on Partial Evaluation, 1977, ISBN 91-7372-144-1.
- No 17 **Bengt Magnhagen:** Probability Based Verification of Time Margins in Digital Designs, 1977, ISBN 91-7372-157-3.
- No 18 **Mats Cedwall:** Semantisk analys av process-beskrivningar i naturligt språk, 1977, ISBN 91-7372-168-9.
- No 22 **Jaak Urmi:** A Machine Independent LISP Compiler and its Implications for Ideal Hardware, 1978, ISBN 91-7372-188-3.
- No 33 **Tore Risch:** Compilation of Multiple File Queries in a Meta-Database System 1978, ISBN 91-7372-232-4.
- No 51 **Erland Jungert:** Synthesizing Database Structures from a User Oriented Data Model, 1980, ISBN 91-7372-387-8.
- No 54 **Sture Hägglund:** Contributions to the Development of Methods and Tools for Interactive Design of Applications Software, 1980, ISBN 91-7372-404-1.
- No 55 **Pär Emanuelson:** Performance Enhancement in a Well-Structured Pattern Matcher through Partial Evaluation, 1980, ISBN 91-7372-403-3.
- No 58 **Bengt Johnsson, Bertil Andersson:** The Human-Computer Interface in Commercial Systems, 1981, ISBN 91-7372-414-9.
- No 69 **H. Jan Komorowski:** A Specification of an Abstract Prolog Machine and its Application to Partial Evaluation, 1981, ISBN 91-7372-479-3.
- No 71 **René Reboh:** Knowledge Engineering Techniques and Tools for Expert Systems, 1981, ISBN 91-7372-489-0.
- No 77 **Östen Oskarsson:** Mechanisms of Modifiability in large Software Systems, 1982, ISBN 91-7372-527-7.
- No 94 **Hans Lunell:** Code Generator Writing Systems, 1983, ISBN 91-7372-652-4.
- No 97 **Andrzej Lingas:** Advances in Minimum Weight Triangulation, 1983, ISBN 91-7372-660-5.
- No 109 **Peter Fritzson:** Towards a Distributed Programming Environment based on Incremental Compilation, 1984, ISBN 91-7372-801-2.
- No 111 **Erik Tengvald:** The Design of Expert Planning Systems. An Experimental Operations Planning System for Turning, 1984, ISBN 91-7372-805-5.
- No 155 **Christos Levcopoulos:** Heuristics for Minimum Decompositions of Polygons, 1987, ISBN 91-7870-133-3.
- No 165 **James W. Goodwin:** A Theory and System for Non-Monotonic Reasoning, 1987, ISBN 91-7870-183-X.
- No 170 **Zebo Peng:** A Formal Methodology for Automated Synthesis of VLSI Systems, 1987, ISBN 91-7870-225-9.
- No 174 **Johan Fagerström:** A Paradigm and System for Design of Distributed Systems, 1988, ISBN 91-7870-301-8.

- No 192 **Dimitar Driankov:** Towards a Many Valued Logic of Quantified Belief, 1988, ISBN 91-7870-374-3.
- No 213 **Lin Padgham:** Non-Monotonic Inheritance for an Object Oriented Knowledge Base, 1989, ISBN 91-7870-485-5.
- No 214 **Tony Larsson:** A Formal Hardware Description and Verification Method, 1989, ISBN 91-7870-517-7.
- No 221 **Michael Reinfrank:** Fundamentals and Logical Foundations of Truth Maintenance, 1989, ISBN 91-7870-546-0.
- No 239 **Jonas Löwgren:** Knowledge-Based Design Support and Discourse Management in User Interface Management Systems, 1991, ISBN 91-7870-720-X.
- No 244 **Henrik Eriksson:** Meta-Tool Support for Knowledge Acquisition, 1991, ISBN 91-7870-746-3.
- No 252 **Peter Eklund:** An Epistemic Approach to Interactive Design in Multiple Inheritance Hierarchies, 1991, ISBN 91-7870-784-6.
- No 258 **Patrick Doherty:** NML3 - A Non-Monotonic Formalism with Explicit Defaults, 1991, ISBN 91-7870-816-8.
- No 260 **Nahid Shahmehri:** Generalized Algorithmic Debugging, 1991, ISBN 91-7870-828-1.
- No 264 **Nils Dahlbäck:** Representation of Discourse-Cognitive and Computational Aspects, 1992, ISBN 91-7870-850-8.
- No 265 **Ulf Nilsson:** Abstract Interpretations and Abstract Machines: Contributions to a Methodology for the Implementation of Logic Programs, 1992, ISBN 91-7870-858-3.
- No 270 **Ralph Rönquist:** Theory and Practice of Tense-bound Object References, 1992, ISBN 91-7870-873-7.
- No 273 **Björn Fjellborg:** Pipeline Extraction for VLSI Data Path Synthesis, 1992, ISBN 91-7870-880-X.
- No 276 **Staffan Bonnier:** A Formal Basis for Horn Clause Logic with External Polymorphic Functions, 1992, ISBN 91-7870-896-6.
- No 277 **Kristian Sandahl:** Developing Knowledge Management Systems with an Active Expert Methodology, 1992, ISBN 91-7870-897-4.
- No 281 **Christer Bäckström:** Computational Complexity of Reasoning about Plans, 1992, ISBN 91-7870-979-2.
- No 292 **Mats Wirén:** Studies in Incremental Natural Language Analysis, 1992, ISBN 91-7871-027-8.
- No 297 **Mariam Kamkar:** Interprocedural Dynamic Slicing with Applications to Debugging and Testing, 1993, ISBN 91-7871-065-0.
- No 302 **Tingting Zhang:** A Study in Diagnosis Using Classification and Defaults, 1993, ISBN 91-7871-078-2.
- No 312 **Arne Jönsson:** Dialogue Management for Natural Language Interfaces - An Empirical Approach, 1993, ISBN 91-7871-110-X.
- No 338 **Simin Nadjm-Tehrani:** Reactive Systems in Physical Environments: Compositional Modelling and Framework for Verification, 1994, ISBN 91-7871-237-8.

- No 371 **Bengt Savén:** Business Models for Decision Support and Learning. A Study of Discrete-Event Manufacturing Simulation at Asea/ABB 1968-1993, 1995, ISBN 91-7871-494-X.
- No 375 **Ulf Söderman:** Conceptual Modelling of Mode Switching Physical Systems, 1995, ISBN 91-7871-516-4.
- No 383 **Andreas Kågedal:** Exploiting Groundness in Logic Programs, 1995, ISBN 91-7871-538-5.
- No 396 **George Fodor:** Ontological Control, Description, Identification and Recovery from Problematic Control Situations, 1995, ISBN 91-7871-603-9.
- No 413 **Mikael Pettersson:** Compiling Natural Semantics, 1995, ISBN 91-7871-641-1.
- No 414 **Xinli Gu:** RT Level Testability Improvement by Testability Analysis and Transformations, 1996, ISBN 91-7871-654-3.
- No 416 **Hua Shu:** Distributed Default Reasoning, 1996, ISBN 91-7871-665-9.
- No 429 **Jaime Villegas:** Simulation Supported Industrial Training from an Organisational Learning Perspective - Development and Evaluation of the SSIT Method, 1996, ISBN 91-7871-700-0.
- No 431 **Peter Jonsson:** Studies in Action Planning: Algorithms and Complexity, 1996, ISBN 91-7871-704-3.
- No 437 **Johan Boye:** Directional Types in Logic Programming, 1996, ISBN 91-7871-725-6.
- No 439 **Cecilia Sjöberg:** Activities, Voices and Arenas: Participatory Design in Practice, 1996, ISBN 91-7871-728-0.
- No 448 **Patrick Lambrix:** Part-Whole Reasoning in Description Logics, 1996, ISBN 91-7871-820-1.
- No 452 **Kjell Orsborn:** On Extensible and Object-Relational Database Technology for Finite Element Analysis Applications, 1996, ISBN 91-7871-827-9.
- No 459 **Olof Johansson:** Development Environments for Complex Product Models, 1996, ISBN 91-7871-855-4.
- No 461 **Lena Strömbäck:** User-Defined Constructions in Unification-Based Formalisms, 1997, ISBN 91-7871-857-0.
- No 462 **Lars Degerstedt:** Tabulation-based Logic Programming: A Multi-Level View of Query Answering, 1996, ISBN 91-7871-858-9.
- No 475 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av hur ekonomiska styrsystem utformas och används efter företagsförvärv, 1997, ISBN 91-7871-914-3.
- No 480 **Mikael Lindvall:** An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Software Evolution, 1997, ISBN 91-7871-927-5.
- No 485 **Göran Forslund:** Opinion-Based Systems: The Cooperative Perspective on Knowledge-Based Decision Support, 1997, ISBN 91-7871-938-0.
- No 494 **Martin Sköld:** Active Database Management Systems for Monitoring and Control, 1997, ISBN 91-7219-002-7.
- No 495 **Hans Olsén:** Automatic Verification of Petri Nets in a CLP framework, 1997, ISBN 91-7219-011-6.
- No 498 **Thomas Drakengren:** Algorithms and Complexity for Temporal and Spatial Formalisms, 1997, ISBN 91-7219-019-1.
- No 502 **Jakob Axelsson:** Analysis and Synthesis of Heterogeneous Real-Time Systems, 1997, ISBN 91-7219-035-3.
- No 503 **Johan Ringström:** Compiler Generation for Data-Parallel Programming Languages from Two-Level Semantics Specifications, 1997, ISBN 91-7219-045-0.
- No 512 **Anna Moberg:** Närhet och distans - Studier av kommunikationsmönster i satellitkontor och flexibla kontor, 1997, ISBN 91-7219-119-8.
- No 520 **Mikael Ronström:** Design and Modelling of a Parallel Data Server for Telecom Applications, 1998, ISBN 91-7219-169-4.
- No 522 **Niclas Ohlsson:** Towards Effective Fault Prevention - An Empirical Study in Software Engineering, 1998, ISBN 91-7219-176-7.
- No 526 **Joachim Karlsson:** A Systematic Approach for Prioritizing Software Requirements, 1998, ISBN 91-7219-184-8.
- No 530 **Henrik Nilsson:** Declarative Debugging for Lazy Functional Languages, 1998, ISBN 91-7219-197-x.
- No 555 **Jonas Hallberg:** Timing Issues in High-Level Synthesis, 1998, ISBN 91-7219-369-7.
- No 561 **Ling Lin:** Management of 1-D Sequence Data - From Discrete to Continuous, 1999, ISBN 91-7219-402-2.
- No 563 **Eva I. Ragnemalm:** Student Modelling based on Collaborative Dialogue with a Learning Companion, 1999, ISBN 91-7219-412-X.
- No 567 **Jörgen Lindström:** Does Distance matter? On geographical dispersion in organisations, 1999, ISBN 91-7219-439-1.
- No 582 **Vanja Josifovski:** Design, Implementation and Evaluation of a Distributed Mediator System for Data Integration, 1999, ISBN 91-7219-482-0.
- No 589 **Rita Kovordányi:** Modeling and Simulating Inhibitory Mechanisms in Mental Image Reinterpretation - Towards Cooperative Human-Computer Creativity, 1999, ISBN 91-7219-506-1.
- No 592 **Mikael Ericsson:** Supporting the Use of Design Knowledge - An Assessment of Commenting Agents, 1999, ISBN 91-7219-532-0.
- No 593 **Lars Karlsson:** Actions, Interactions and Narratives, 1999, ISBN 91-7219-534-7.
- No 594 **C. G. Mikael Johansson:** Social and Organizational Aspects of Requirements Engineering Methods - A practice-oriented approach, 1999, ISBN 91-7219-541-X.
- No 595 **Jörgen Hansson:** Value-Driven Multi-Class Overload Management in Real-Time Database Systems, 1999, ISBN 91-7219-542-8.
- No 596 **Niklas Hallberg:** Incorporating User Values in the Design of Information Systems and Services in the Public Sector: A Methods Approach, 1999, ISBN 91-7219-543-6.
- No 597 **Vivian Vimarlund:** An Economic Perspective on the Analysis of Impacts of Information Technology: From Case Studies in Health-Care towards General Models and Theories, 1999, ISBN 91-7219-544-4.
- No 598 **Johan Jenvald:** Methods and Tools in Computer-Supported Taskforce Training, 1999, ISBN 91-7219-547-9.
- No 607 **Magnus Merkel:** Understanding and enhancing translation by parallel text processing, 1999, ISBN 91-7219-614-9.
- No 611 **Silvia Coradeschi:** Anchoring symbols to sensory data, 1999, ISBN 91-7219-623-8.
- No 613 **Man Lin:** Analysis and Synthesis of Reactive Systems: A Generic Layered Architecture Perspective, 1999, ISBN 91-7219-630-0.

- No 618 **Jimmy Tjäder:** Systemimplementering i praktiken - En studie av logiker i fyra projekt, 1999, ISBN 91-7219-657-2.
- No 627 **Vadim Engelson:** Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing, 2000, ISBN 91-7219-709-9.
- No 637 **Esa Falkenroth:** Database Technology for Control and Simulation, 2000, ISBN 91-7219-766-8.
- No 639 **Per-Arne Persson:** Bringing Power and Knowledge Together: Information Systems Design for Autonomy and Control in Command Work, 2000, ISBN 91-7219-796-X.
- No 660 **Erik Larsson:** An Integrated System-Level Design for Testability Methodology, 2000, ISBN 91-7219-890-7.
- No 688 **Marcus Bjärelund:** Model-based Execution Monitoring, 2001, ISBN 91-7373-016-5.
- No 689 **Joakim Gustafsson:** Extending Temporal Action Logic, 2001, ISBN 91-7373-017-3.
- No 720 **Carl-Johan Petri:** Organizational Information Provision - Managing Mandatory and Discretionary Use of Information Technology, 2001, ISBN-91-7373-126-9.
- No 724 **Paul Scerri:** Designing Agents for Systems with Adjustable Autonomy, 2001, ISBN 91 7373 207 9.
- No 725 **Tim Heyer:** Semantic Inspection of Software Artifacts: From Theory to Practice, 2001, ISBN 91 7373 208 7.
- No 726 **Pär Carlshamre:** A Usability Perspective on Requirements Engineering - From Methodology to Product Development, 2001, ISBN 91 7373 212 5.
- No 732 **Juha Takkinen:** From Information Management to Task Management in Electronic Mail, 2002, ISBN 91 7373 258 3.
- No 745 **Johan Åberg:** Live Help Systems: An Approach to Intelligent Help for Web Information Systems, 2002, ISBN 91-7373-311-3.
- No 746 **Rego Granlund:** Monitoring Distributed Teamwork Training, 2002, ISBN 91-7373-312-1.
- No 757 **Henrik André-Jönsson:** Indexing Strategies for Time Series Data, 2002, ISBN 917373-346-6.
- No 747 **Anneli Hagdahl:** Development of IT-supported Interorganisational Collaboration - A Case Study in the Swedish Public Sector, 2002, ISBN 91-7373-314-8.
- No 749 **Sofie Pilemalm:** Information Technology for Non-Profit Organisations - Extended Participatory Design of an Information System for Trade Union Shop Stewards, 2002, ISBN 91-7373-318-0.
- No 765 **Stefan Holmlid:** Adapting users: Towards a theory of use quality, 2002, ISBN 91-7373-397-0.
- No 771 **Magnus Morin:** Multimedia Representations of Distributed Tactical Operations, 2002, ISBN 91-7373-421-7.
- No 772 **Pawel Pietrzak:** A Type-Based Framework for Locating Errors in Constraint Logic Programs, 2002, ISBN 91-7373-422-5.
- No 758 **Erik Berglund:** Library Communication Among Programmers Worldwide, 2002, ISBN 91-7373-349-0.
- No 774 **Choong-ho Yi:** Modelling Object-Oriented Dynamic Systems Using a Logic-Based Framework, 2002, ISBN 91-7373-424-1.
- No 779 **Mathias Broxvall:** A Study in the Computational Complexity of Temporal Reasoning, 2002, ISBN 91-7373-440-3.
- No 793 **Asmus Pandikow:** A Generic Principle for Enabling Interoperability of Structured and Object-Oriented Analysis and Design Tools, 2002, ISBN 91-7373-479-9.
- No 785 **Lars Hult:** Publika Informationstjänster. En studie av den Internetbaserade encyklopedins bruksegenskaper, 2003, ISBN 91-7373-461-6.
- No 800 **Lars Taxén:** A Framework for the Coordination of Complex Systems' Development, 2003, ISBN 91-7373-604-X.
- No 808 **Klas Gäre:** Tre perspektiv på förväntningar och förändringar i samband med införande av informationssystem, 2003, ISBN 91-7373-618-X.
- No 821 **Mikael Kindborg:** Concurrent Comics - programming of social agents by children, 2003, ISBN 91-7373-651-1.
- No 823 **Christina Ölvingson:** On Development of Information Systems with GIS Functionality in Public Health Informatics: A Requirements Engineering Approach, 2003, ISBN 91-7373-656-2.
- No 828 **Tobias Ritzau:** Memory Efficient Hard Real-Time Garbage Collection, 2003, ISBN 91-7373-666-X.
- No 833 **Paul Pop:** Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems, 2003, ISBN 91-7373-683-X.
- No 852 **Johan Moe:** Observing the Dynamic Behaviour of Large Distributed Systems to Improve Development and Testing - An Empirical Study in Software Engineering, 2003, ISBN 91-7373-779-8.
- No 867 **Erik Herzog:** An Approach to Systems Engineering Tool Data Representation and Exchange, 2004, ISBN 91-7373-929-4.
- No 872 **Aseel Berglund:** Augmenting the Remote Control: Studies in Complex Information Navigation for Digital TV, 2004, ISBN 91-7373-940-5.
- No 869 **Jo Skåmedal:** Telecommuting's Implications on Travel and Travel Patterns, 2004, ISBN 91-7373-935-9.
- No 870 **Linda Askenäs:** The Roles of IT - Studies of Organising when Implementing and Using Enterprise Systems, 2004, ISBN 91-7373-936-7.
- No 874 **Annika Flycht-Eriksson:** Design and Use of Ontologies in Information-Providing Dialogue Systems, 2004, ISBN 91-7373-947-2.
- No 873 **Peter Bunus:** Debugging Techniques for Equation-Based Languages, 2004, ISBN 91-7373-941-3.
- No 876 **Jonas Mellin:** Resource-Predictable and Efficient Monitoring of Events, 2004, ISBN 91-7373-956-1.
- No 883 **Magnus Bång:** Computing at the Speed of Paper: Ubiquitous Computing Environments for Healthcare Professionals, 2004, ISBN 91-7373-971-5.
- No 882 **Robert Eklund:** Disfluency in Swedish human-human and human-machine travel booking dialogues, 2004, ISBN 91-7373-966-9.
- No 887 **Anders Lindström:** English and other Foreign Linguistic Elements in Spoken Swedish. Studies of Productive Processes and their Modelling using Finite-State Tools, 2004, ISBN 91-7373-981-2.
- No 889 **Zhiping Wang:** Capacity-Constrained Production-inventory systems - Modelling and Analysis in both a traditional and an e-business context, 2004, ISBN 91-85295-08-6.
- No 893 **Pernilla Qvarfordt:** Eyes on Multimodal Interaction, 2004, ISBN 91-85295-30-2.
- No 910 **Magnus Kald:** In the Borderland between Strategy and Management Control - Theoretical Framework and Empirical Evidence, 2004, ISBN 91-85295-82-5.

- No 918 **Jonas Lundberg:** Shaping Electronic News: Genre Perspectives on Interaction Design, 2004, ISBN 91-85297-14-3.
- No 900 **Mattias Arvola:** Shades of use: The dynamics of interaction design for sociable use, 2004, ISBN 91-85295-42-6.
- No 920 **Luis Alejandro Cortés:** Verification and Scheduling Techniques for Real-Time Embedded Systems, 2004, ISBN 91-85297-21-6.
- No 929 **Diana Szentivanyi:** Performance Studies of Fault-Tolerant Middleware, 2005, ISBN 91-85297-58-5.
- No 933 **Mikael Cäker:** Management Accounting as Constructing and Opposing Customer Focus: Three Case Studies on Management Accounting and Customer Relations, 2005, ISBN 91-85297-64-X.
- No 937 **Jonas Kvarnström:** TALplanner and Other Extensions to Temporal Action Logic, 2005, ISBN 91-85297-75-5.
- No 938 **Bourhane Kadmiry:** Fuzzy Gain-Scheduled Visual Servoing for Unmanned Helicopter, 2005, ISBN 91-85297-76-3.
- No 945 **Gert Jervan:** Hybrid Built-In Self-Test and Test Generation Techniques for Digital Systems, 2005, ISBN: 91-85297-97-6.
- No 946 **Anders Arpteg:** Intelligent Semi-Structured Information Extraction, 2005, ISBN 91-85297-98-4.
- No 947 **Ola Angelsmark:** Constructing Algorithms for Constraint Satisfaction and Related Problems - Methods and Applications, 2005, ISBN 91-85297-99-2.
- No 963 **Calin Curescu:** Utility-based Optimisation of Resource Allocation for Wireless Networks, 2005, ISBN 91-85457-07-8.
- No 972 **Björn Johansson:** Joint Control in Dynamic Situations, 2005, ISBN 91-85457-31-0.
- No 974 **Dan Lawesson:** An Approach to Diagnosability Analysis for Interacting Finite State Systems, 2005, ISBN 91-85457-39-6.
- No 979 **Claudiu Duma:** Security and Trust Mechanisms for Groups in Distributed Services, 2005, ISBN 91-85457-54-X.
- No 983 **Sorin Manolache:** Analysis and Optimisation of Real-Time Systems with Stochastic Behaviour, 2005, ISBN 91-85457-60-4.
- No 986 **Yuxiao Zhao:** Standards-Based Application Integration for Business-to-Business Communications, 2005, ISBN 91-85457-66-3.
- No 1004 **Patrik Haslum:** Admissible Heuristics for Automated Planning, 2006, ISBN 91-85497-28-2.
- No 1005 **Aleksandra Tešanovic:** Developing Reusable and Reconfigurable Real-Time Software using Aspects and Components, 2006, ISBN 91-85497-29-0.
- No 1008 **David Dinka:** Role, Identity and Work: Extending the design and development agenda, 2006, ISBN 91-85497-42-8.
- No 1009 **Iakov Nakhimovski:** Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis, 2006, ISBN 91-85497-43-X.
- No 1013 **Wilhelm Dahllöf:** Exact Algorithms for Exact Satisfiability Problems, 2006, ISBN 91-85523-97-6.
- No 1016 **Levon Saldamli:** PDEModelica - A High-Level Language for Modeling with Partial Differential Equations, 2006, ISBN 91-85523-84-4.
- No 1017 **Daniel Karlsson:** Verification of Component-based Embedded System Designs, 2006, ISBN 91-85523-79-8.
- No 1018 **Ioan Chisalita:** Communication and Networking Techniques for Traffic Safety Systems, 2006, ISBN 91-85523-77-1.
- No 1019 **Tarja Susi:** The Puzzle of Social Activity - The Significance of Tools in Cognition and Cooperation, 2006, ISBN 91-85523-71-2.
- No 1021 **Andrzej Bednarski:** Integrated Optimal Code Generation for Digital Signal Processors, 2006, ISBN 91-85523-69-0.
- No 1022 **Peter Aronsson:** Automatic Parallelization of Equation-Based Simulation Programs, 2006, ISBN 91-85523-68-2.
- No 1030 **Robert Nilsson:** A Mutation-based Framework for Automated Testing of Timeliness, 2006, ISBN 91-85523-35-6.
- No 1034 **Jon Edvardsson:** Techniques for Automatic Generation of Tests from Programs and Specifications, 2006, ISBN 91-85523-31-3.
- No 1035 **Vaida Jakoniene:** Integration of Biological Data, 2006, ISBN 91-85523-28-3.
- No 1045 **Genevieve Gorrell:** Generalized Hebbian Algorithms for Dimensionality Reduction in Natural Language Processing, 2006, ISBN 91-85643-88-2.
- No 1051 **Yu-Hsing Huang:** Having a New Pair of Glasses - Applying Systemic Accident Models on Road Safety, 2006, ISBN 91-85643-64-5.
- No 1054 **Åsa Hedenskog:** Perceive those things which cannot be seen - A Cognitive Systems Engineering perspective on requirements management, 2006, ISBN 91-85643-57-2.
- No 1061 **Cécile Åberg:** An Evaluation Platform for Semantic Web Technology, 2007, ISBN 91-85643-31-9.
- No 1073 **Mats Grindal:** Handling Combinatorial Explosion in Software Testing, 2007, ISBN 978-91-85715-74-9.
- No 1075 **Almut Herzog:** Usable Security Policies for Runtime Environments, 2007, ISBN 978-91-85715-65-7.
- No 1079 **Magnus Wahlström:** Algorithms, measures, and upper bounds for Satisfiability and related problems, 2007, ISBN 978-91-85715-55-8.
- No 1083 **Jesper Andersson:** Dynamic Software Architectures, 2007, ISBN 978-91-85715-46-6.
- No 1086 **Ulf Johansson:** Obtaining Accurate and Comprehensive Data Mining Models - An Evolutionary Approach, 2007, ISBN 978-91-85715-34-3.
- No 1089 **Traian Pop:** Analysis and Optimisation of Distributed Embedded Systems with Heterogeneous Scheduling Policies, 2007, ISBN 978-91-85715-27-5.
- No 1091 **Gustav Nordh:** Complexity Dichotomies for CSP-related Problems, 2007, ISBN 978-91-85715-20-6.
- No 1106 **Per Ola Kristensson:** Discrete and Continuous Shape Writing for Text Entry and Control, 2007, ISBN 978-91-85831-77-7.
- No 1110 **He Tan:** Aligning Biomedical Ontologies, 2007, ISBN 978-91-85831-56-2.
- No 1112 **Jessica Lindblom:** Minding the body - Interacting socially through embodied action, 2007, ISBN 978-91-85831-48-7.
- No 1113 **Pontus Wärnestål:** Dialogue Behavior Management in Conversational Recommender Systems, 2007, ISBN 978-91-85831-47-0.
- No 1120 **Thomas Gustafsson:** Management of Real-Time Data Consistency and Transient Overloads in Embedded Systems, 2007, ISBN 978-91-85831-33-3.

- No 1127 **Alexandru Andrei**: Energy Efficient and Predictable Design of Real-time Embedded Systems, 2007, ISBN 978-91-85831-06-7.
- No 1139 **Per Wikberg**: Eliciting Knowledge from Experts in Modeling of Complex Systems: Managing Variation and Interactions, 2007, ISBN 978-91-85895-66-3.
- No 1143 **Mehdi Amirjoo**: QoS Control of Real-Time Data Services under Uncertain Workload, 2007, ISBN 978-91-85895-49-6.
- No 1150 **Sanny Syberfeldt**: Optimistic Replication with Forward Conflict Resolution in Distributed Real-Time Databases, 2007, ISBN 978-91-85895-27-4.
- No 1155 **Beatrice Alenljung**: Envisioning a Future Decision Support System for Requirements Engineering - A Holistic and Human-centred Perspective, 2008, ISBN 978-91-85895-11-3.
- No 1156 **Artur Wilk**: Types for XML with Application to Xcerpt, 2008, ISBN 978-91-85895-08-3.
- No 1183 **Adrian Pop**: Integrated Model-Driven Development Environments for Equation-Based Object-Oriented Languages, 2008, ISBN 978-91-7393-895-2.
- No 1185 **Jörgen Skågeby**: Gifting Technologies - Ethnographic Studies of End-users and Social Media Sharing, 2008, ISBN 978-91-7393-892-1.
- No 1187 **Imad-Eldin Ali Abugessaisa**: Analytical tools and information-sharing methods supporting road safety organizations, 2008, ISBN 978-91-7393-887-7.
- No 1204 **H. Joe Steinhauer**: A Representation Scheme for Description and Reconstruction of Object Configurations Based on Qualitative Relations, 2008, ISBN 978-91-7393-823-5.
- No 1222 **Anders Larsson**: Test Optimization for Core-based System-on-Chip, 2008, ISBN 978-91-7393-768-9.
- No 1238 **Andreas Borg**: Processes and Models for Capacity Requirements in Telecommunication Systems, 2009, ISBN 978-91-7393-700-9.
- No 1240 **Fredrik Heintz**: DyKnow: A Stream-Based Knowledge Processing Middleware Framework, 2009, ISBN 978-91-7393-696-5.
- No 1241 **Birgitta Lindström**: Testability of Dynamic Real-Time Systems, 2009, ISBN 978-91-7393-695-8.
- No 1244 **Eva Blomqvist**: Semi-automatic Ontology Construction based on Patterns, 2009, ISBN 978-91-7393-683-5.
- No 1249 **Rogier Woltjer**: Functional Modeling of Constraint Management in Aviation Safety and Command and Control, 2009, ISBN 978-91-7393-659-0.
- No 1260 **Gianpaolo Conte**: Vision-Based Localization and Guidance for Unmanned Aerial Vehicles, 2009, ISBN 978-91-7393-603-3.
- No 1262 **AnnMarie Ericsson**: Enabling Tool Support for Formal Analysis of ECA Rules, 2009, ISBN 978-91-7393-598-2.
- No 1266 **Jiri Trnka**: Exploring Tactical Command and Control: A Role-Playing Simulation Approach, 2009, ISBN 978-91-7393-571-5.
- No 1268 **Bahlol Rahimi**: Supporting Collaborative Work through ICT - How End-users Think of and Adopt Integrated Health Information Systems, 2009, ISBN 978-91-7393-550-0.
- No 1274 **Fredrik Kuivinen**: Algorithms and Hardness Results for Some Valued CSPs, 2009, ISBN 978-91-7393-525-8.
- No 1281 **Gunnar Mathiason**: Virtual Full Replication for Scalable Distributed Real-Time Databases, 2009, ISBN 978-91-7393-503-6.
- No 1290 **Viacheslav Izosimov**: Scheduling and Optimization of Fault-Tolerant Distributed Embedded Systems, 2009, ISBN 978-91-7393-482-4.
- No 1294 **Johan Thapper**: Aspects of a Constraint Optimisation Problem, 2010, ISBN 978-91-7393-464-0.
- No 1306 **Susanna Nilsson**: Augmentation in the Wild: User Centered Development and Evaluation of Augmented Reality Applications, 2010, ISBN 978-91-7393-416-9.
- No 1313 **Christer Thörn**: On the Quality of Feature Models, 2010, ISBN 978-91-7393-394-0.
- No 1321 **Zhiyuan He**: Temperature Aware and Defect-Probability Driven Test Scheduling for System-on-Chip, 2010, ISBN 978-91-7393-378-0.
- No 1333 **David Broman**: Meta-Languages and Semantics for Equation-Based Modeling and Simulation, 2010, ISBN 978-91-7393-335-3.
- No 1337 **Alexander Siemers**: Contributions to Modelling and Visualisation of Multibody Systems Simulations with Detailed Contact Analysis, 2010, ISBN 978-91-7393-317-9.
- No 1354 **Mikael Asplund**: Disconnected Discoveries: Availability Studies in Partitioned Networks, 2010, ISBN 978-91-7393-278-3.
- No 1359 **Jana Rambusch**: Mind Games Extended: Understanding Gameplay as Situated Activity, 2010, ISBN 978-91-7393-252-3.
- No 1373 **Sonia Sangari**: Head Movement Correlates to Focus Assignment in Swedish, 2011, ISBN 978-91-7393-154-0.
- No 1374 **Jan-Erik Källhammer**: Using False Alarms when Developing Automotive Active Safety Systems, 2011, ISBN 978-91-7393-153-3.
- No 1375 **Mattias Eriksson**: Integrated Code Generation, 2011, ISBN 978-91-7393-147-2.
- No 1381 **Ola Leifler**: Affordances and Constraints of Intelligent Decision Support for Military Command and Control - Three Case Studies of Support Systems, 2011, ISBN 978-91-7393-133-5.
- No 1386 **Soheil Samii**: Quality-Driven Synthesis and Optimization of Embedded Control Systems, 2011, ISBN 978-91-7393-102-1.
- No 1419 **Erik Kuiper**: Geographic Routing in Intermittently-connected Mobile Ad Hoc Networks: Algorithms and Performance Models, 2012, ISBN 978-91-7519-981-8.
- No 1451 **Sara Stymne**: Text Harmonization Strategies for Phrase-Based Statistical Machine Translation, 2012, ISBN 978-91-7519-887-3.
- No 1455 **Alberto Montebelli**: Modeling the Role of Energy Management in Embodied Cognition, 2012, ISBN 978-91-7519-882-8.
- No 1465 **Mohammad Saifullah**: Biologically-Based Interactive Neural Network Models for Visual Attention and Object Recognition, 2012, ISBN 978-91-7519-838-5.
- No 1490 **Tomas Bengtsson**: Testing and Logic Optimization Techniques for Systems on Chip, 2012, ISBN 978-91-7519-742-5.
- No 1481 **David Byers**: Improving Software Security by Preventing Known Vulnerabilities, 2012, ISBN 978-91-7519-784-5.
- No 1496 **Tommy Färnqvist**: Exploiting Structure in CSP-related Problems, 2013, ISBN 978-91-7519-711-1.
- No 1503 **John Wilander**: Contributions to Specification, Implementation, and Execution of Secure Software, 2013, ISBN 978-91-7519-681-7.

- No 1506 **Magnus Ingmarsson**: Creating and Enabling the Useful Service Discovery Experience, 2013, ISBN 978-91-7519-662-6.
- No 1547 **Wladimir Schamai**: Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool, 2013, ISBN 978-91-7519-505-6.
- No 1551 **Henrik Svensson**: Simulations, 2013, ISBN 978-91-7519-491-2.
- No 1559 **Sergiu Rafiliu**: Stability of Adaptive Distributed Real-Time Systems with Dynamic Resource Management, 2013, ISBN 978-91-7519-471-4.
- No 1581 **Usman Dastgeer**: Performance-aware Component Composition for GPU-based Systems, 2014, ISBN 978-91-7519-383-0.
- No 1602 **Cai Li**: Reinforcement Learning of Locomotion based on Central Pattern Generators, 2014, ISBN 978-91-7519-313-7.
- No 1652 **Roland Samlaus**: An Integrated Development Environment with Enhanced Domain-Specific Interactive Model Validation, 2015, ISBN 978-91-7519-090-7.
- No 1663 **Hannes Uppman**: On Some Combinatorial Optimization Problems: Algorithms and Complexity, 2015, ISBN 978-91-7519-072-3.
- No 1664 **Martin Sjölund**: Tools and Methods for Analysis, Debugging, and Performance Improvement of Equation-Based Models, 2015, ISBN 978-91-7519-071-6.
- No 1666 **Kristian Stavåker**: Contributions to Simulation of Modelica Models on Data-Parallel Multi-Core Architectures, 2015, ISBN 978-91-7519-068-6.
- No 1680 **Adrian Lifa**: Hardware/Software Codesign of Embedded Systems with Reconfigurable and Heterogeneous Platforms, 2015, ISBN 978-91-7519-040-2.
- No 1685 **Bogdan Tanasa**: Timing Analysis of Distributed Embedded Systems with Stochastic Workload and Reliability Constraints, 2015, ISBN 978-91-7519-022-8.
- No 1691 **Håkan Warnquist**: Troubleshooting Trucks – Automated Planning and Diagnosis, 2015, ISBN 978-91-7685-993-3.
- No 1702 **Nima Aghaee**: Thermal Issues in Testing of Advanced Systems on Chip, 2015, ISBN 978-91-7685-949-0.
- No 1715 **Maria Vasilevskaya**: Security in Embedded Systems: A Model-Based Approach with Risk Metrics, 2015, ISBN 978-91-7685-917-9.
- No 1729 **Ke Jiang**: Security-Driven Design of Real-Time Embedded System, 2016, ISBN 978-91-7685-884-4.
- No 1733 **Victor Lagerkvist**: Strong Partial Clones and the Complexity of Constraint Satisfaction Problems: Limitations and Applications, 2016, ISBN 978-91-7685-856-1.
- No 1734 **Chandan Roy**: An Informed System Development Approach to Tropical Cyclone Track and Intensity Forecasting, 2016, ISBN 978-91-7685-854-7.
- No 1746 **Amir Aminifar**: Analysis, Design, and Optimization of Embedded Control Systems, 2016, ISBN 978-91-7685-826-4.

Linköping Studies in Arts and Science

- No 504 **Ing-Marie Jonsson**: Social and Emotional Characteristics of Speech-based In-Vehicle Information Systems: Impact on Attitude and Driving Behaviour, 2009, ISBN 978-91-7393-478-7.
- No 586 **Fabian Segelström**: Stakeholder Engagement for Service Design: How service designers identify and communicate insights, 2013, ISBN 978-91-7519-554-4.
- No 618 **Johan Blomkvist**: Representing Future Situations of Service: Prototyping in Service Design, 2014, ISBN 978-91-7519-343-4.
- No 620 **Marcus Mast**: Human-Robot Interaction for Semi-Autonomous Assistive Robots, 2014, ISBN 978-91-7519-319-9.

Linköping Studies in Statistics

- No 9 **Davood Shahsavani**: Computer Experiments Designed to Explore and Approximate Complex Deterministic Models, 2008, ISBN 978-91-7393-976-8.
- No 10 **Karl Wahlin**: Roadmap for Trend Detection and Assessment of Data Quality, 2008, ISBN 978-91-7393-792-4.
- No 11 **Oleg Sysoev**: Monotonic regression for large multivariate datasets, 2010, ISBN 978-91-7393-412-1.
- No 13 **Agné Burauskaite-Harju**: Characterizing Temporal Change and Inter-Site Correlations in Daily and Sub-daily Precipitation Extremes, 2011, ISBN 978-91-7393-110-6.

Linköping Studies in Information Science

- No 1 **Karin Axelsson**: Metodisk systemstrukturerings- att skapa samstämmighet mellan informationssystem- arkitektur och verksamhet, 1998. ISBN-9172-19-296-8.
- No 2 **Stefan Cronholm**: Metodverktyg och användbarhet - en studie av datorstödd metodbaserad systemutveckling, 1998, ISBN-9172-19-299-2.
- No 3 **Anders Avdic**: Användare och utvecklare - om anveckling med kalkylprogram, 1999. ISBN-91-7219-606-8.
- No 4 **Owen Eriksson**: Kommunikationskvalitet hos informationssystem och affärsprocesser, 2000, ISBN 91-7219-811-7.
- No 5 **Mikael Lind**: Från system till process - kriterier för processbestämning vid verksamhetsanalys, 2001, ISBN 91-7373-067-X.
- No 6 **Ulf Melin**: Koordination och informationssystem i företag och nätverk, 2002, ISBN 91-7373-278-8.
- No 7 **Pär J. Ågerfalk**: Information Systems Actability - Understanding Information Technology as a Tool for Business Action and Communication, 2003, ISBN 91-7373-628-7.
- No 8 **Ulf Seigerroth**: Att förstå och förändra systemutvecklingsverksamheter - en taxonomi för metautveckling, 2003, ISBN91-7373-736-4.
- No 9 **Karin Hedström**: Spår av datoriseringens värden - Effekter av IT i äldreomsorg, 2004, ISBN 91-7373-963-4.
- No 10 **Ewa Braf**: Knowledge Demanded for Action - Studies on Knowledge Mediation in Organisations, 2004, ISBN 91-85295-47-7.
- No 11 **Fredrik Karlsson**: Method Configuration method and computerized tool support, 2005, ISBN 91-85297-48-8.
- No 12 **Malin Nordström**: Styrbar systemförvaltning - Att organisera systemförvaltningsverksamhet med hjälp

av effektiva förvaltningsobjekt, 2005, ISBN 91-85297-60-7.

No 13 **Stefan Holgersson:** Yrke: POLIS - Yrkeskunskap, motivation, IT-system och andra förutsättningar för polisarbete, 2005, ISBN 91-85299-43-X.

No 14 **Benneth Christiansson, Marie-Therese Christiansson:** Mötet mellan process och komponent - mot ett ramverk för en verksamhetsnära kravspecifikation vid anskaffning av komponent-baserade informationssystem, 2006, ISBN 91-85643-22-X.