Design of Embedded Control Systems under real-time Scheduling Constraints

Tesi per il conseguimento del diploma di perfezionamento
Anno Accademico 2002/2001

Luigi PALOPOLI (palopoli@sssup.it)

Collegio dei Docenti

Prof. Paolo ANCILOTTI Prof. Giorgio BUTTAZZO Dott. Andrea DOMENICI

RETIS Lab.
Scuola Superiore S. Anna
Via Carducci, 40
56100 Pisa – Italy

To the love of my life, Grazia
To my parents
To Nonna

Contents

Acknowledgments

It is obviously difficult to write down a list of the people I wish to thank for helping me during the years of my PhD. In chronological order, I think I will start from my parents who gave me life and the opportunity of finding my way through the complex and fascinating world of the Italian university. Then my heart goes to my bride-to-be, Grazia. She never stopped loving me, encouraging me and stimulating me during the tough times of the PhD, when the professional fun in undertaking novel scientific challenges is not matched by economic wealth. I never heard complaints about my choice from her and she gave me a fundamental help in correcting my worst trends to victimism. When it comes to professionals, the first person I want to thank is my supervisor, Paolo Ancilotti. He is really a great guy who trusted me and opened my mind on the secrets of the academic world. My decision to undertake PhD is largely due to Prof. Rich Gerber, who presented me the incredible opportunities of the academic career. After a while he apparently changed his mind on this topic, but he is my precious friend and I will never forget him. Professors Giorgio Buttazzo, Marco Di Natale and Andrea Domenici gave me precious hints throughout my work. Most of what I learnt in control theory was due to the stimuli offered by Professor Bicchi. He is a perfect mentor and it was a great honour and pleasure for me to work with him. PhD students and Researchers in the ReTiS lab compose a really exciting environment. It is great to joke with them, to fight with them, to work with them, to grow with them. My heartfelt thanks to Peppe, to Gerardo, Paolo, Luca, Tommaso, Davide and Marco. Alessandro did me a favour providing the latex style used for the thesis. Most of what I learnt on the realm of embedded systems was built during 7 precious months of my life spent at the University of California Berkeley. UCB is a beautiful place crowded with talented people. I will never stop being grateful to Prof. Sangiovanni-Vincentelli for inviting me in his group. He is incredibly full of energy and

ideas...sometimes it takes you months to figure what can stem from half an hour of discussion with him. I wish to thank Claudio Pinello. A good part of this thesis arose from our cooperation. Professor El-Ghaoui was very kind and gave me a lot of useful suggestions on optimization topics. Finally, I want to thank my Buddhist Master, Daisaku Ikeda; most of his teachings are now written deep in my cells and in my blood.

1 Introduction

I love deadlines. I like the whooshing sound as they fly by.
- Douglas Adams

Embedded systems (ES) are electronic components of industrial systems that typically

1. Monitor variables of the physical system such as temperature, pressure, traffic, chemical composition, position, or speed via sensor measurements;

2. Process the information making use of one or more mathematical models of the physical system;

3. Output signals, such as controls for power circuitry to open or close valves, to ignite sparks, to open or close switches. These signals control the behaviour of the physical system and allow to optimize its performance.

Embedded systems are typically hidden from the user and they have to be designed to operate in close connection with the environment. In particular, the most challenging embedded systems must produce a reaction to an external event complying with strict timing requirements. Such systems are called reactive embedded systems or real-time embedded controllers (EC). Roughly speaking, reactive systems produce responses to external events at the speed imposed by the external environment, whereas "interactive" systems produce the results of the requested computations at their own speed.

The commercial importance of Embedded Controllers (EC) and of Embedded Systems in general has steadily grown in the past few years disclosing unprecedented opportunities for improving the quality of the resulting

products. The pervasiveness of these components is well illustrated by the following facts:

- the total shipment of microprocessor units (MPU) and micro control units (MCU) in 1997 was over 4.4 billion units, and about 98% of this are related to embedded applications1; and
- between 1994 and 2004 the need for embedded software developers is expected to increase 10 fold.

One of the most important application fields is represented by the automotive industry: from a marginal technology applied to a category of niche products, embedded controllers have become a pervasive solution to a variety of different problems concerning safety, comfort, fuel consumption, emission of polluting gases and so forth. In a recent meeting of the R&D engineers of Daimler Chrysler in Stuttgart, it was argued that 90% of the innovation in future cars will rely on electronic components. In 1998, Dr. Wolfgang Reitzle, then the Chief Operating Officer of BMW, indicated that the BMW series 7 (the top of the line of BMW production) had more than 70 embedded processors accounting for more than 30% of the cost of the car. The stability problem of the Mercedes Class A vehicle was solved relatively quickly by devising new control algorithms cleverly implemented in software on the powerful platform that was dedicated to suspension control. This solution would not have been possible just a few years ago. The emerging X-by wire technology, applied for years in the avionics field, is making inroads even in economic lines of automotive products promising sharp improvements in comfort, performance, pollution and safety.

This thesis proposes a set of analytical procedures and a simulation CAD tool aimed at effective design of EC. A major emphasis will be put on embedded software (ESW). In this chapter we will first introduce the main problems related to embedded software development, clarifying the reasons for its relevance, and then summarize the main results of the thesis.

1.1 The Challenge of Embedded Software Development

The development of EC is a very complex activity for a variety of reasons. The first problem is the presence of tight safety requirements. A failure of the system can even result into the loss of human lives. Moreover, fixing a problem

in a system shipped in thousands of units can result in enormous additional costs for its manufacturer. For this reason, EC have to be design error free to the maximum possible extent in industrial. Although relevant, this is not the only problem posed by these systems. As a matter of fact, other issues such as time-to-market, costs of the deployed architecture, power consumption and memory footprint are known to play a role of increasing importance. In short, designers and engineers are confronted with a challenge of overwhelming complexity: safety-critical applications have to be developed in shorter times using low-cost components.

In this context, the interest for embedded software (ESW) has risen to previously unseen levels. In fact, hardware manufacturing cycles are long and expensive, whereas software based systems are intrinsically flexible and allow for easy porting across different architectures. The increasing cost of the masks pushes hardware manufacturer toward the sharing of hardware platforms across multiple applications in a given domain. Moreover, such innovations as reconfigurable platforms along with the constant increase in computing power and the corresponding decrease in costs are enabling the shifting of functionalities to software. Chances are that in a near future most system engineers even in semi-conductors companies will be involved in software development. Unfortunately, the current industrial practice in ESW lags far behind the attainment of its ambitious goals. In most cases the development goes through different stages: first a control law is synthesized and then its implementation is devised going through costly prototyping activities. While this empirical approach could yield acceptable results for simple control laws implemented on cheap platforms, it is no more adequate to sustain the growing complexity of modern applications. Recent accidentes, like the ones of the Mars Polar Lander and the Ariane Rocket, clearly pointed out the risks of using outdated methodologies in the development of complex systems. In less catastrophic cases, finding a design error during the late phases of the development may invalidate precious months of work.

In a utterly competitive market, where development time and cost consideration are likely to dominate decision-making processes, effective ways for streamlining the development of ESW from the specifications down to the deployment on HW/SW architectures will be a key success factor for EC manufacturers. This thesis elaborates the problem of ESW development along different directions. The proposed vision is based on the converging application of tools and ideas pertaining to different disciplines that have to be tapped to construct a “sound” methodology. The need for integration of different disciplines is not surprising; rather, it is inherent to the very concept of embedded computing: i.e. information processing tightly integrated with

physical processes. In this work, the stress will be put on the possible synergies between control and software engineering in EC design.

To this regard we will maintain a useful conceptual distinction between two different viewpoints over EC design: functional design and architectural design. Control engineering comes into play when defining the functional design, while computer engineering mainly regards architectural design.

Functional and architectural design should be as much as possible "orthogonal" activities [7]: each of them should be governed by the pursuit of its own design trade-offs. In this way it is possible to exhaustively explore alternative solutions without impacting on the design time. We use the adjective "orthogonal" as opposed to "separate" to emphasize that the outcomes of these activities should be safely and verifiably composed in a well founded methodological framework. On the contrary, in the current practice, the passage from functional specification to the selection of an architecture and to the implementation is a "blind" activity where choices belonging to different conceptual domains are intertwined. As a result, the overall design turns out to be over-constrained thus hindering the re-usability of components and generating unnecessary commitments to specific architectures (and to their producers).

One of the greatest problem in devising a sound design procedure of this kind is dealing with concurrency: i.e. parallel computations and communications sharing limited resources. A somewhat counterintuitive fact is that for functional design to be effectively orthogonal with respect to architectural design, it is necessary that an abstract modeling of concurrency be present also in functional modeling. This is one of the most important pitfalls of currently available tools. A Matlab/Simulink scheme captures very well the physical modeling of a plant or of an analog controller, but such aspects as concurrent computation/communication and the model of computation of the controller are not adequately captured. It is our opinion that a paradigm shfit is necessary and we believe that it can stem from the application of platform based approaches. An architecture, or more precisely a set of architectures, are modeled by means of a limited set of parameters and its effects on the system performance can be adequately taken into account ever since the early phases of the design. This absract view on the architecture is called "platform". If the level of abstraction for the platform is correctly chosen, the use of a platform during the functional design does not commit the architecural designer to a specific solution but provides him/her with well defined specification that can drive the exploration of different alternatives. We will show how the theory of real-time computing allows one to define platforms that are simple enoguh for being tractable in the context of analytical control

design procedures. On the other hand the most important architectural aspects affecting the system's perfromance are captured. We will show that for two important applications classical control design procedure can easily be adapted to the problem of control under real-time scheduling constraints. Fundamental components of the approach are CAD tools enabling architecture/function cosimulation and *a posteriori* performance assesment.

1.2 Contents of the thesis

In chapter 2 we will present some background information on the state of the art of disciplines that will be tapped throughout the thesis. In particular, we will shortly review the state of the art in such topics as control design with information constraints and real-time HW/SW architecture. Moreover we will present a methodological framework that, in our evaluation, could radically improve the current state of the art of EC software development. The proposed methodology is largely inspired to the principles of Platform Based Design [38].

The first part of the thesis applies these principles in two different contexts. Namely, in Chapter 3 we consider a plant comprised of several subsystems. Each of them is controlled by a dedicated control software task. Tasks share a single CPU computer board operated by a RTOS. We address two distinct issues: 1) is a platform powerful enough to sustain a specified level of performance? 2) once a platform has been selected how are design parameters to be selected to optimize performance? In Chapter 4 the considered platform consists of a shared BUS used to deliver messages to a group of actuators. In this case we assume that the BUS implements a TDMA scheduling paradigm. A numerically efficient algorithm allows to synthesize both the control law and the bus schedule. The procedure allows to take into account also physical constraints on the actuators.

An essential role of the proposed methodology is covered by the possibility of assessing the performance of the system by a Computer Aided Design (CAD) tool. The tool simulates at one time: 1) a set of plants, 2) the functional model of a controller, 3) the architectural model of a controller. To this purpose, a collection of C++ libraries called RTSIM has been realized and it is presented in the second part of the thesis in Chapter 5.

Finally, in Chapter 6, we state the conclusions of the thesis.

2 Background information

I always say that, next to a battle lost, the greatest misery is a battle gained.
–The Duke of Wellington
I prefer the most unjust peace to the most righteous war.
– Cicero

The main focus of the thesis is to establish a clear and well-founded connection between control engineering and computer engineering, with particular regard to the results of real-time scheduling theory. Therefore, we will first provide a general description of the main topics and references, belonging to either discipline, that are of interest for our purposes. By the end of the chapter, we will provide an overall description of a possible methodology that in our evaluation could be greatly beneficial with respect to software development for EC.

2.1 Control Design in EC

It is a commonplace opinion that designs of embedded systems have to be captured at the highest level of abstraction to take advantage of all available degrees of freedom.

From this perspective, the situation in control design is encouraging with respect to other fields. Control designers are used to tackle their problem using high level primitives. In particular, the use of abstract graphical notations such as Simulink is utmost popular. Each block of a Simulink diagram is part of a library and it has a well defined mathematical meaning, e.g. a gain, a transfer function, a digital filter, a finite state machine. Moreover, it exposes some free parameters (e.g the zeros and the poles of a transfer function) that are degrees of freedom left to the designer. In short, designing a

controller amounts to devising an interconnection of blocks and to appropriately selecting their parameters. The question is: how are parameters selected? Here lies the power of this approach since decades of academic research and industrial practice have produced analytical and semi-analytical procedures to synthesize the appropriate selection of parameters. The synthesis is driven by a set of metrics that can either be used as constraints or as cost functions in optimization procedures. Typical examples of such metrics are stability, robustness, convergence speed, quality of the transient evolution, disturbance rejection and so forth. A particular relevance is assumed, in our context by performance limitations introduced by the controller implementation. In particular, we want to address limitations on the amount of information that can be processed by the controller.

2.1.1 Control under information constraints

A control designer typically has a good physical intuition of what cost functions and constraints should best be used to assess the accomplishment of her/his design goals. In particular, the physical limitations of the controller, such as sensor accuracy or actuators authority, have been carefully considered in the past literature [14]. An entire branch of control theory (i.e. Model predictive control) [50] explicitly integrates limitations on achievable control signals in the design procedure.

Only recently is the increasing complexity of computations and communications inside ECs exposing the importance of other types of constraints, which, in a general sense, are related to the amount of information that can be processed and conveyed from sensors to actuators in unit time [15]. Different lines of research consider the problem of information constraints in control systems under different perspectives. Bandwidth and bit-rate constraints in communication links are dealt with in [52, 75, 76]. The application of LQG techniques to control synthesis over a limited bandwidth channel is illustrated in [69]. Another important thread of research focuses on the use of quantized sensors and actuators: the traditional view of quantization as a disturbance is replaced by a specific analysis of this phenomenon even unveiling unexpected opportunities [12, 21, 16].

In this thesis, we deal with information constraints stemming from the use of shared communication and computation components. There are many real-life examples where this situation occurs. For instance the same channel (or bus) can be used to convey multiple sensor readings to processing units or, dually, multiple command values to actuators. In another scenario,

the same processing unit can be used to carry out multiple concurrent computations (tasks).

The interest for this type of systems amongst control designer is well proven by the growing popularity of such tools as the Real-time workshop by MATHWORKS INC. or ASCET-SD by Etas Engineering. These tools shorten the distance between functional design and implementation by automatically generating the low level code from functional models. However, in absence of an adequate understanding of what exactly means to have a concurrent implementation of an EC on a limited pool of computation and communication resources during the design phase, the simple generation of code turns out to be a blunt sword.

Remarkable attempts to combine control synthesis and scheduling of communication channels are in [55, 30]. In both papers, an integrated formulation encompassing control and communication is proposed. In this thesis this problem will be dealt-with in the context of a platform based design methodology, as shown later in this chapter and in the next chapters.

2.2 Hardware-software architectures for EC

A typical architecture used in modern EC is reported in Figure 2.1. The shown architecture is comprised of different nodes connected *via* a shared communication resource (BUS). At each node, traditional solutions based on dedicated hardware are being replaced by more flexible *computer boards*. By computer boards we mean a basic microarchitecture composed of programmable cores, I/O subsystems and memory. This basic architecture can be used with multiple applications thus sharing the costs of mask sets and design (which are becoming a dominant factor in deep submicron IC) over a wider number of deployed components. In some cases, the use of FPGA enables to retain a certain degree of parametrization of the board for particular purposes.

Shared computation or communication components (or resources) must be equipped with schedulers to enforce “mutually exclusive” access: in the case of multiple access requests to a resource, only one of them can be satisfied at a given time. According to the type of components they manage, schedulers can be realized in hardware, software or both. A real-time operating system is an example of a software scheduler, while a bus arbitrator is an example of hardware scheduler. The scheduler satisfies pending requests according to a scheduling policy that, in turn, bases its decisions upon a set of scheduling parameters that are associated to each entity requiring the re-

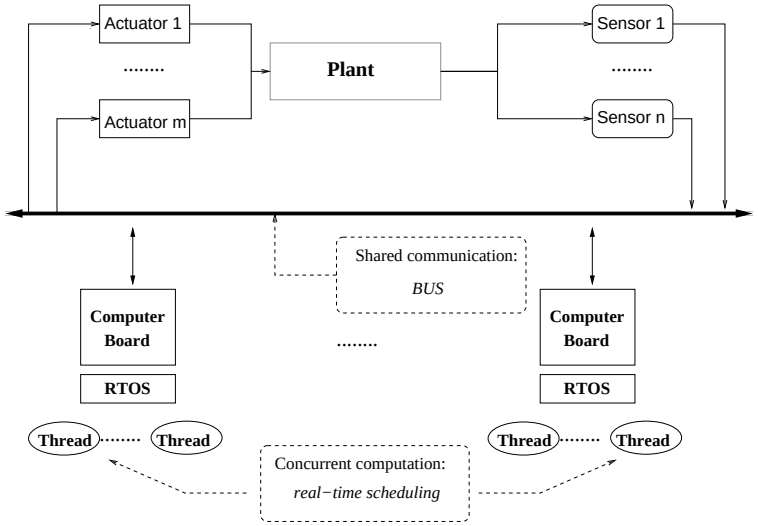


Figure 2.1 Typical architecture of a modern EC

source (which can be a packet for a communication resource or a software task for a computation resource).

Scheduling policies are chosen according to the requirements of the application. In the case of EC the most important requirements is time-determinism (or *real-time schedulability*) that can be formulated as follows: *given a set of timing constraints for the tasks and for the communication packets, find a schedule of the shared resources such that all constraints are met.*

An entire branch of academic research in computer science is devoted to real-time scheduling algorithms. In the rest of the section we give some basic reference. The interested reader is referred to the large literature in the field, encompassing specialized journals, proceedings of conferences such as the Real-Time Systems Symposium and text books.

2.2.1 Real-time scheduling of processors

This section discusses the scheduling of a computation resource (a processor) shared among multiple tasks. A *task* is a piece of software that is repeatedly executed on a processor. Each activation of a task corresponds to the execution of a *job*. A task is said *periodic* of period p, being p a natural number, if its activation instants are separated by p clock units. Each job of a real-time task is assigned an *absolute deadline*, which is the time by which the job has to terminate. The *relative deadline* of a job is defined as the difference between the absolute deadline and the instant of the job's activation.

Processor scheduling algorithms can be of different types. In particular it is possible to distinguish between *on-line* and *off-line* algorithms. The former take the scheduling decisions before the activation of tasks, based on an *a priori* knowledge of the task parameters. The implementation of the algorithm consists of a simple table storing the activation times of all tasks. The latter have an ordered queue of jobs to be executed. The queue is updated, at run-time, during the tasks' execution. Another important distinction is between preemptive and non-preemptive algorithms. Preemptive algorithms allow the execution of a job to be suspended when a job with higher priority is activated and resumed later. On the contrary using non-preemptive algorithms, after a job is assigned the processor it cannot be interrupted until its completion. It is evident that, being non-preemptive algorithms a subset of preemptive algorithms, the number of task sets that are real-time schedulable by the former is a subset of those that are schedulable with the latter. Within preemptive algorithms a further classification between static and dynamic priorities is possible. For static priority algorithms the scheduling algorithms are based on fixed parameters that are assigned to a task once and for all upon its first activation. Dynamic priority algorithms allow one to change parameters driving the scheduling decisions in any moment.

Task sets composed only of periodic tasks for which the relative deadline is equal to the period are very important in control applications. The best known real-time scheduling algorithms for periodic task sets are Rate-Monotonic (RM) and Earliest-deadline-first (EDF). Both algorithms are preemptive, on-line and based on the assignment of priorities (i.e. at each clock the task having the highest priority is assigned the CPU). Using RM task priorities are assigned according to the activation period (the shorter the period, the higher the priority). The EDF scheduling algorithm assigns the highest priority to the task having the earliest absolute deadline. Clearly, EDF implements a scheduling policy based on a dynamic assignment of priorities.

Much of the recent literature in real-time processor scheduling derives from a fundamental result stated in a seminal work of Liu and Layland [47].

Theorem 1 (Liu and Layland 1973) *Consider a set of independent periodic tasks $\tau_1, \tau_2, \ldots, \tau_n$. Assume that each τ_i has worst case execution time e_i and is activated with period p_i. Then the following statements are true:*

1. *if the EDF scheduling algorithm is used then the task set is schedulable if and only if $\sum_{i=1,\ldots,n} \frac{e_i}{p_i} \leq 1$;*

2. *if the RM scheduling algorithm is used then the task set is schedulable if* $\sum_{i=1,\ldots,n} \frac{e_i}{p_i} \leq n(2^{\frac{1}{n}} - 1)$.

This result stemmed two prongs of research activity yielding a remarkable amount of results, which have been systematically organized in text books [65, 49]. In particular important problems are:

- extensions for aperiodic tasks,
- considering the case of interacting task (i.e. tasks accessing mutually exclusive resource such as shared memory buffers and devices).

As far as the first problem is concerned, a popular way for enhancing efficiency are aperiodic servers (Polling server [44], Sporadic Server [61], Constant Bandwidth Server [1], etc). When tasks interact, checking and enforcing schedulability becomes a difficult problem due to the priority inversion phenomenon. This phenomenon was discovered by Lui Sha [59] and it vividly manifested itself in a famous incident occurred to the Mars Planetary Rover Pathfinder. Strategies for avoiding priority inversion based on static priority schedulers are priority inheritance and priority ceiling [59]. The Stack Resource Policy (SRP) [5] accomplishes the same goal using dynamic priority schedulers. In the industrial practice static priorities are usually preferred to dynamic priorities because they are easier to implement.

The most important real-time operating system supporting static priority algorithms is currently VXWorks by Wind River Inc., which is shipped in millions of embedded systems ranging from network printers to military airplanes.

In spite of the remarkable amount of results found for single processor systems, the problem of real-time scheduling is much more open when it comes to consider multiprocessor systems. An important result was proposed by Sanjoy Baruah [8] for sets made of independent periodic tasks run on a symmetric multiprocessor allowing migration. The algorithm proposed realizes the abstraction called "proportionate fairness": processors can be thought of as "fluid" resources that can be shared between the different processes. A task having a worst case execution time e_i activated with period p_i, receives a share of the processor equal to $w_i = \frac{e_i}{p_i}$ and its execution proceeds as it would on a slower dedicated processor, up to a certain granularity. This work is important in that it generalizes the schedulability condition found by Liu and Layland for single processor systems: the task set is schedulable on m processors if and only if $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq m$. The concept of a processor as

a fluid resource that can be shared between processes is particular convenient for multimedia applications and inspired also the family of scheduling algorithms called proportional share [67]. A different viewpoint on the same problem is taken by the Resource Reservation algorithms such as the constant bandwidth server [2]. Resource reservation scheduling for different types of resources (CPU, disks, devices) are implemented in the commercial real-time kernel linux-RK [54], produced by Timesys inc.

2.2.2 Real-time scheduling of the communication resources

In principle, the problem of real-time scheduling of packets on communication resources (buses) is similar to real-time scheduling of software tasks on processors. However, there are some differences that must be taken into account. The first important difference is that while a task execution can, in principle, be interrupted (preempted) and resumed at any time, packets transmissions over buses cannot be interrupted. Thereby, the granularity for preemption in buses is generally coarser than the granularity for preemption in processors. Moreover, the HW/SW architecture for communication is typically layered (protocol stack). Problems like the resource contention can be tackled at different levels in the protocol stack and different combinations of strategies can be devised at the different levels.

In our abstract view, the bus is simply a shared resource that is contended between different nodes. There are two types of strategies to resolve the problem of contention. The first type of strategies relies on a shared clock and realizes different flavours of time division multiplexing (TDMA). Using TDMA, scheduling decisions are taken offline according to a periodic pattern; each period is divided into slots and each slot is assigned to one of the peers. In this way contention never occurs, as long as each node behaves properly and as the clocks of the peers are properly synchronized. Clearly, this type of protocols is very well suited for real-time communication and it is comparable to the off-line scheduling of processors. A very important technology inspired by this principle is the Time-Triggered Architecture (TTA) produced by TTTech Inc. and derived from the work of Prof. Herman Kopetz in University of Vienna [35]. We will come back later on this topic. A problem of non trivial complexity is the synchronization of the clocks. Moreover, this approach is very conservative (a slot is reserved to a node even when it does need it) and it is not well suited to handle aperiodic events. To deal with this problem, the BMW car manufacturer has recently proposed the BYTEFLIGHT technology.

An alternative concept of buses allows contention, which is solved on-

line whenever it occurs. An example is represented by Ethernet networks and, in general by aloha protocols. When two packets collide on the bus the peers re-transmit the packets after a random time thus decreasing the probability of another collision. Aloha protocols are not normally used in real-time embedded applications since they cannot provide any timing guarantee on the delivery of packets. Better results can be obtained by using the Controller Area Network (CAN) Bus. Using the CAN bus, contentions are solved assigning a priority to each packet. This feature enables the implementation of rate-monotonic [72] or EDF [53] scheduling policies. For this reason, and for the limited cost of the hardware, the CAN bus is very popular in automotive contexts.

2.2.3 The time-triggered model of computation

The time-triggered model of computation [36] is a model for the representation, design and analysis of distribute embedded systems. Its basic components are time-accurate interfaces, communication systems, host computers and transducers that connect the environment to the interfaces. This model has been developer in the University of Vienna by Prof. Kopetz and it stemmed the production of the TTA architecture.

The most important concepts are interfaces, also called temporal firewalls, that are dual-ported memories, containing timing-accurate information, accessed by communicating subsystems according an a priori known time-triggered schedule. The underlying idea is that subsystems communicate between themselves and with the external environment only at specified instants in time. The designer is let free in the choice of the scheduling provided that the specifications of the temporal firewalls are respected. This concept has been revisited in the Giotto Programming Language [68], where periodic tasks sample data from the environment and communicate only at specified instants.

2.3 The Envisioned Methodology

The methodology we envision for developing EC is depicted in Figure 2.2. In particular we can distinguish different activities:

- functional design,
- architectural design,

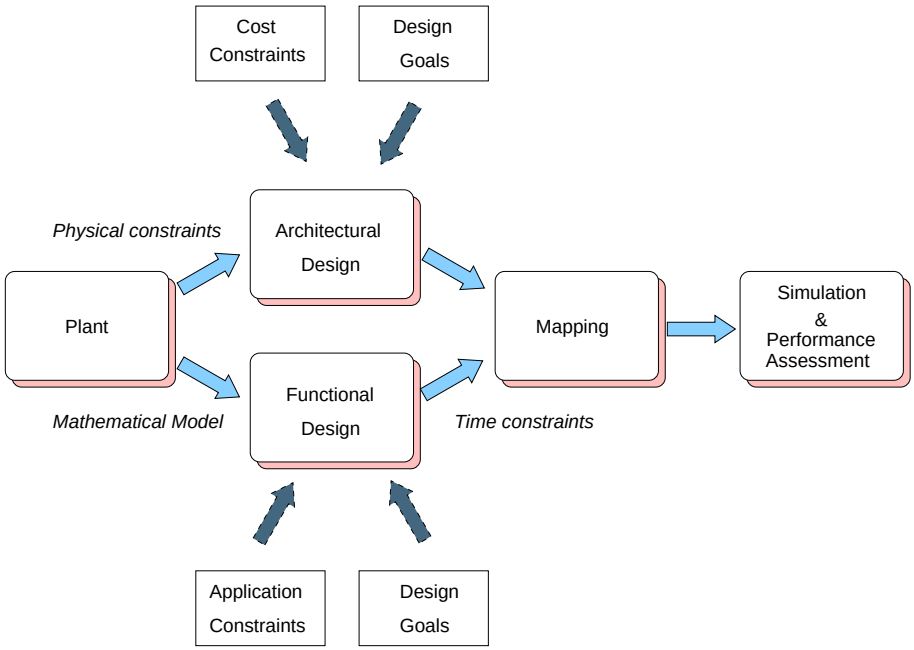


Figure 2.2 Schematic representation of the envisioned methodology

- mapping,
- performance assessment.

Functional design corresponds to the definition of the control algorithms. Inputs to this activity is the mathematical model of the plant. The design is carried out by means of analytical or semi-analytical procedures. The goals of the design and the constraints it is subject to are mostly dictated by physical considerations on the application. For instance, on a car we might require that the maximum obtainable accelerations do not jeopardise its structural integrity. As far as design goals are concerned, a possible design goal is that the quantity of fuel required in a given operation mode be minimized. In other classes of products, such as sport cars, we might require that the time required to regulate the speed on a set point be minimized.

Architectural design aims at selecting the distributed architecture (hardware, RTOS, device drivers, protocols etc.) to properly support the application processes. The selection process can be driven by different goals and constraints. The constraints include

- maximum total cost (sum of the cost of each component)

- architectural constraints (e.g. dictated by the location of sensors and actuators on a large plant)
- fault model (e.g. the fault model may exclude any resource with less than k processors or p channels).

The design goals may include

- minimize total cost
- minimize number of processors and/or channels
- minimize resource utilization factors.

During the mapping activity the different functions are assigned (mapped) to the components of the architecture. In particular a set of each functional blocks has to be mapped on a software task executed on a node. Communication between the different functional blocks are mapped on global variables, mutually exclusive memory buffers, packets transmitted over a channel and so on. Clearly when the mapping is performed, the designer has to enforce the timing constraints introduced during the control design.

The performance assessment can be developed along two different directions: simulation and prototyping. We are particularly interested in the performance assessment by simulation. The simulation tool operates on a heterogeneous aggregation of models (continuous time plants, communication links, real-time schedulers). The result of this phase are performance measures which can include: 1) control theoretic performance (overshoots, rise time, linear quadratic functions, etc.), 2) timing behaviour (CPU utilization, end-to-end delays etc.).

2.3.1 Platform based design

Figure 2.2 pictorially expresses the fundamental design principle advocated in most recent literature on ES, i.e. the orthogonalization of concerns between functional and architectural design. As shown, the two activities have their own design space, goals and constraints. Good chances are that requirements placed on either of them might even be contrasting. For instance the functional designer might push toward a powerful architecture to maximize the control performance while cost constraints induce the selection of cheap components from the architectural side. Hopefully, the design procedure should enable the definition of system-level goals and constraints

across the boundaries of the different activities. For instance, it should be possible to deal with formulations like:

- find the cheapest architecture that sustains a functional performance specification, or
- given an architecture find a design that optimizes a functional performance measure.

An effective methodology that fulfills these requirements is neither bottom-up nor top-down. Rather it has to be a provably correct "meet-in-the-middle" approach. The central question to be answered is: "how should the control designer view the implementation?". The question can be equivalently be rephrased in the architectural domain as: "how should the requirements of the control application be translated into a specification for an architecture?". Answering this question means to find a difficult balance between two contrasting needs. On the one hand, we want to hide as many details as possible of the implementation during the functional design phase. On the other we want to retain a sufficient level of awareness and control on performance limitations introduced by the implementation to enable an early and realistic assessment of the final performance of the controller: the most critical parameters of the underlying hardware and software architectures have to be exposed (e.g., the number of processors available and their speed, the communication costs among hardware components, the abstract characteristics of sensors and actuators, the real-time operating system scheduling policy). This goal can be achieved by identifying a set of abstraction levels that mark the progress from specification to implementation. Each abstraction layer has to be related to the next in the sequence by a "behaviour containment" that should guarantee that whatever has been proven correct at one layer will stay correct in the next layer down toward implementation. The step of mapping a design at one layer to the next is a refinement step where more information are added in a systematic way. To be able to prevent problems at lower levels of the design abstractions, it is imperative that we summarize the critical parameters as discussed above. At the same time, we have to be able to transmit constraints as they become explicit from the top layer to the bottom one. The set of abstraction layers (also called platforms), the parameters that characterize the implementation layers, the constraints that are mapped down from the specification layers and the tools that are used to map one layer into another constitute the so-called *platform-based design paradigms* [38].

The basic ideas of platform based design and of system platform are well captured in Figure 2.3. The vertex of the two cones represent the abstraction

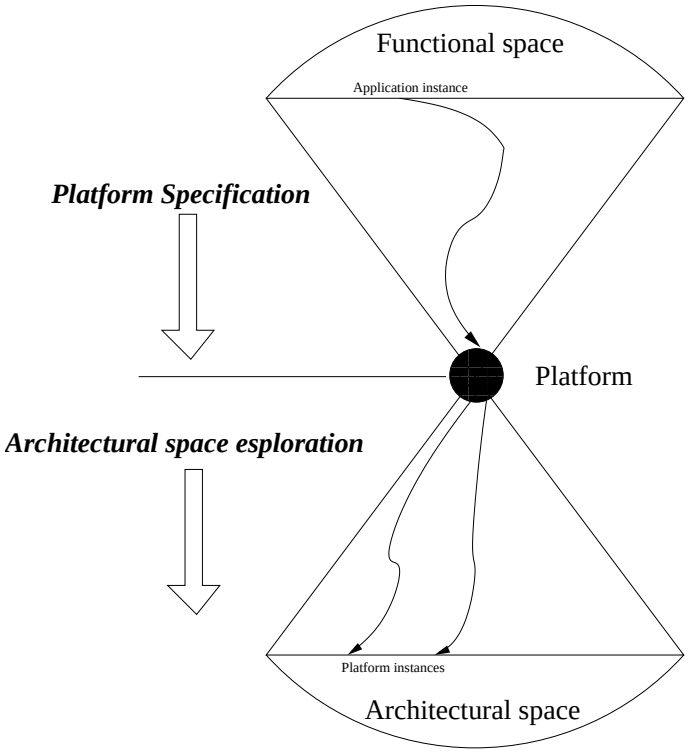


Figure 2.3 Platform abstraction and design flows

of the HW/SW platform. As a matter of fact, for the purposes of this thesis we will use a single abstraction layer capturing the entire Hardware/Software platform. As it is possible to see, the control designer uses the platform abstraction to define her/his control algorithms taking into account the limitations introduced by the platform. In doing so, she/he defines a complete specification of the platform that the architectural designer can use to explore the architectural space according pursuing her/his own trade-offs. We remark that this step can very well take advantage of existing tools such as the Real-Time workshop by Mathworks Inc., the Giotto Programming language, Cierto VCC by Cadence Inc., ASCET-SD and so forth. However, the presence of a well-defined platform specification enables a design for the overall system that is correct by construction.

2.3.2 Contributions of the thesis

In the proposed methodology two points are not adequately covered by existing tools and methods.

The first one is control design with platform constraints. To this regard this thesis offers two distinct contributions. In Chapter 3 a platform composed of a single processor board endowed with a preemptive real-time operating system is considered. In this case, the platform is required to implement a time-triggered model of computation. The platform is characterized by the following parameters: 1) the clock period, 2) the tasks worst case execution times, 3) the tasks activation periods. The latter are free design parameters that can be leveraged, along with "functional" parameters like the gains, to optimize a cost function related to the control robustness. In Chapter 4 the platform consists of a communication resource that is handled by a TDMA policy. The design space consists of the control algorithms and of the allocation of the different slots to the different actuators. We apply a variation of Model Predictive Control and we show that the use of a formulation for the problem based on the Generalized Linear Complementarity Problem (GCLP) makes for a numerically efficient solution.

The second point that in our evaluation is not adequately addressed by current design tools is the simulation of the system after mapping, which is instrumental to the performance assesment. In particular, we are interested in a fine grained modeling of the performance effects due to real-time scheduling. To this purpose we devised a tool, presented in Chapter 5. By using RTSIM, it is possible to provide an accurate modeling of the concurrent architecture of the control tasks and of the run-time support offered by the operating system for the real-time scheduling of the shared resources (CPU,

memory buffers and network links). In this way, it is possible to compare different scheduling solutions by evaluating their simulated performance directly in the domain of the control application. Moreover, the tool can be utilized to tune up such design parameters as the activation frequencies of the tasks.

I

Control synthesis

3

Holistic design of real-time controllers on single CPU platforms

It is the mark of an educated mind to be able to entertain a thought without accepting it.
– Aristotle

The platform based design approach described in Chapter 2 is demonstrated on a particular problem in this chapter. In particular we consider the problem of controlling a set of linear continuous time system using a shared computation resource. Software tasks are scheduled using a preemptive real-time scheduling policy. To cope with the problem of jitter in data acquisition and release a time-triggered model of computation is used. The combination of the time-triggered MoC and of real-time schedulability allows for an easy modeling of the Platform accounting both for the timing behaviour and for the resource constraints.

The set of control tasks must be scheduled trying to maximize some control performance metric and satisfying a the schedulability constraints arising from the platform. The performance metric proposed is a measure of the closed loop stability robustness, more specifically a generalization of the classical gain margin: the stability radius. As well as being relevant in many application, this performance metric is particularly suited to the considered application since it is directly linked to the quantity of information processed by each loop. We consider either memoryless controllers and a class of dy-

namic controllers enabling delay compensation. The stability radius is derived exactly for the special class of first order systems while upper and lower bounds are derived for the more general case of multi-variable systems. We address two different kind of problems:1) deciding whether a platform is adequate to achieve a performance specification, 2) optimizing the stability radius once a platform has been selected. The optimization problem encompasses both continuous and integer variables, since the activation periods of the tasks are integer multiples of the clock. The solution strategy is based on the continuous relaxation, while the integrality of the activation periods is recovered using a Branch and Bound scheme. Results are particularly insightful for first order systems and they will be shown on a numeric example.

Basic mathematical definitions and notation Consider a generic vector space $\mathcal{V}$ endowed with an inner product $< \cdot, \cdot >$. The notation $||\cdot||$ denotes a norm defined on $\mathcal{V}$. The dual norm associated $||\cdot||$ of $||\cdot||$ is defined as follows:

$$||\mathbf{u}||_* = \sup_{||v||=1} \{< \mathbf{u}, \mathbf{v} >\} \tag{3.1}$$

where $\mathbf{u}$, $\mathbf{v}$ belong to $\mathcal{V}$. Considering vector norms defined on $\mathbb{R}^n$ space, it can easily be seen that the dual norm of the Euclidean norm is the Euclidean norm itself, while the dual norm of the ∞ norm is the 1-norm and *vice versa*.

The $B_R(\mathbf{u})$ set denotes a ball having radius R and centered in $\mathbf{u}$:

$$B_R(\mathbf{u}) = \{\mathbf{v} \in \mathcal{V} \text{ such that } ||\mathbf{v} - \mathbf{u}|| \leq R\}. \tag{3.2}$$

The distance of a vector $\mathbf{u}$ from a set $\mathcal{U}$ in the norm $||.||$ is defined as:

$$dist(\mathbf{u}, \mathcal{U}) = \begin{cases} 0 & \text{if } \mathbf{u} \in \mathcal{U} \\ \inf_{\mathbf{v} \in \mathcal{U}} ||\mathbf{u} - \mathbf{v}|| & \text{otherwise} \end{cases}$$

The *Chebychev center* of a bounded set C in the norm $||.||$ is defined as:

$$\mathbf{u}^{(c)}(C) = argmax\ dist(\mathbf{u}, \mathcal{V} \backslash C), \tag{3.3}$$

and it is the center of the maximum radius ball inside $\mathcal{C}$.

A matrix is said Schur stable if and only if its eigenvalues are all contained in the unit circle centered in the origin.

Unless otherwise stated, results provided throughout this chapter are referred to the ∞-norm. But, in most cases, extensions to the Euclidean norm are possible.

3.1 The considered platform

Model of Computation. The considered model of computation is time triggered model and it assumes a set of periodic tasks τ_i. Tasks are executed on a computer architecture that may interact with the environment with a time granularity no less than the clock period T_c. Every p_i clock cycles task τ_i becomes active and reads its inputs; then it computes the output values. Outputs are released right at the beginning of the next activation period and their values are sustained between two subsequent writings (according to a ZoH model).

It is worth observing that the timing behavior resembles the one of a Moore sequential circuit, where the delay between inputs and outputs is equal to the activation period p_i. The fixed delay virtually nulls jitter in the release time of the outputs, greatly simplifying the control law design.

Schedulability Constraints. It is evident that for the correctness of the model to be guaranteed, each job must terminate before the next period (i.e. the periodic task set has to be schedulable). A necessary condition for achieving schedulability of the task set on a single CPU is:

$$\sum_{i=1}^{m} \frac{e_i}{p_i} \leq 1, \tag{3.4}$$

where e_i and p_i are integer numbers, counting the clock cycles required to compute process τ_i in the worst case and between successive activations of process τ_i, respectively. As we said in Chapter 2, sufficient conditions can be derived for different scheduling algorithms. Popular examples are Rate Monotonic (RM) and Earliest Deadline First (EDF). As stated in Theorem 1, a sufficient condition for schedulability is:

$$\sum_{i=1}^{m} \frac{e_i}{p_i} \leq U_l, \tag{3.5}$$

where $U_l = 1$ for EDF and $U_l = m(2^{\frac{1}{m}} - 1)(> 0.69)$ for RM. Other scheduling algorithms, called *Pfair* [8], guarantee schedulability under condition 3.5 with $U_l = 1$ on a single processor architecture and scale well also to multiprocessor architectures, by setting U_l equal to the available number of processors. It is worth pointing out that in schedulability analysis e_i are the *worst-case* execution times for processes and that conservative choices on U_l allow to achieve robustness with respect to unmodeled delays or to reserve space for other processes, as required.

Platform definition. Given the choice of the time triggered MoC and of a class of real time schedulers, the implementation platform can be parameterized by:

$$U_l, T_c, e_1, \ldots, e_m, \tag{3.6}$$

to be used in (3.5). A m-tuple of activation periods $\mathbf{p} \in \mathbb{N}^m$ satisfying (3.5) is guaranteed to yield a design adhering to the MoC. It is worth noting that the set of parameters (3.6) may represent a family of implementation platforms, all meeting or exceeding those requirements. As a matter of fact information on the HW/SW architecture condensed in the proposed platform are:

1. it must support the time-triggered MoC,
2. it must ensure worst case execution time of e_i or lower for task τ_i,
3. it has to be endowed with a preemptive RTOS,
4. the scheduling algorithm has to guarantee schedulability with total utilization U_l.

As shown next, the proposed characterization for the platform leads to a compact analytical formulation of the control design problem.

3.2 Problem Formulation

Consider a collection $\mathcal{S}$ of single input systems $\mathcal{S}^{(i)}$ described by equations:

$$\dot{\mathbf{x}}^{(i)} = A^{(i)}\mathbf{x}^{(i)} + \mathbf{b}^{(i)}u^{(i)} \tag{3.7}$$

where $A^{(i)} \in \mathbb{R}^{n_i \times n_i}$, $b^{(i)} \in \mathbb{R}^{n_i}$ and $i = 1, \ldots, m$. Assume that all systems $\mathcal{S}_i$ are completely controllable and each of them is controlled by a dedicated feedback controller implemented by a task $\tau^{(i)}$ (with a slight abuse of terminology words "controller" and "task" will henceforth be used interchangeably).

Recall that T_c is the clock period, i.e. the minimum time granularity for interactions between the control platform and the plant. Correspondingly task $\tau^{(i)}$ will execute every $p^{(i)}T_c$ time units, where $p^{(i)} \in \mathbb{Z}$. Each $\tau^{(i)}$ controller is assumed to have *complete access* to the state variables of the system every $p^{(i)}T_c$ time units. However, the time triggered paradigm introduces a delay of $p^{(i)}T_c$ between sampling the state variables and actuating the corresponding control action.

It is possible to distinguish between two types of controllers:

memoryless controllers: the controller is only able to use in the feedback the delayed variables;

dynamic controllers: the controller has its own memory, thus enabling some form of delay compensation.

In the sequel the analysis will be restricted to two simple but important control schemes belonging to either class. For notational simplicity, throughout this section we will drop the (i) superscript. Hence we shall refer to a system having equation:

$$\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{b}u. \tag{3.8}$$

Let J be the Jordan form of A and U a matrix such that $A = UJU^{-1}$. The J matrix can be written as

$$J = \begin{bmatrix} J_1(\lambda_1) & 0 & 0 & \dots & 0 \\ 0 & J_2(\lambda_2) & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \\ 0 & 0 & \dots & 0 & J_l(\lambda_l) \end{bmatrix}$$

where $\lambda_i \in \mathbb{C}, i = 1, \dots, l$ are the eigenvalues of A and:

$$J_k(\lambda_k) = \begin{bmatrix} \lambda_k & 1 & 0 & \dots & 0 & 0 \\ 0 & \lambda_k & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & & \\ 0 & \dots & \dots & \dots & \lambda_k & 1 \\ 0 & \dots & \dots & \dots & 0 & \lambda_k \end{bmatrix} \in \mathbb{C}^{r_k \times r_k}.$$

As said above, this work considers the case when the value for the control variables is held constant during a computation period (other types of piecewise open loop scheme are reserved for future work). Moreover, the time triggered model of computation imposes to release at instant $(h+1)pT_c$ the command $\tilde{u}(h)$ whose computation begins at instant hpT_c. Therefore, values applied to the actuators are given by:

$$u(t) = u(hpT_c) = \tilde{u}(h-1), \forall t \in [hpT_c, (h+1)pT_c[. \tag{3.9}$$

A convenient way for studying the dynamical properties of this systems is to consider the discrete time sequence $\tilde{\mathbf{x}}$ obtained taking samples of the system at $t = h\ p\ T_c$: $\tilde{\mathbf{x}}(h) = \mathbf{x}(h\ p\ T_c)$.

The dynamic for this sequence is described by:

$$\tilde{\mathbf{x}}(h+1) = e^{A\ p\ T_c}\tilde{\mathbf{x}}(h) + (\int_0^{p\ T_c} e^{As} ds)\mathbf{b}\tilde{\mathbf{u}}(h-1). \tag{3.10}$$

The one period delay can be accounted for in the system analysis by introducing an additional state variable $\tilde{z}$:

$$\begin{bmatrix}\tilde{\mathbf{x}}(h+1)\\ \tilde{z}(h+1)\end{bmatrix} = \tilde{A}\begin{bmatrix}\tilde{\mathbf{x}}(h)\\ \tilde{z}(h)\end{bmatrix} + \tilde{\mathbf{b}}\tilde{u}(h) \tag{3.11}$$

where:

$$\begin{aligned}\tilde{A} &= \begin{bmatrix}e^{A\,p\,T_c} & (\int_0^{pT_c} e^{As}ds)\mathbf{b}\\ 0 & 0\end{bmatrix}\\ \tilde{\mathbf{b}} &= \begin{bmatrix}\mathbf{0}\\ 1\end{bmatrix}.\end{aligned} \tag{3.12}$$

The control models produce the following command sequences:

$$\tilde{u}(h) = \begin{cases}\gamma^{(\mathbf{x})}\tilde{\mathbf{x}}(h) & \text{for memoryless controllers}\\ \gamma^{(\mathbf{x})}\tilde{\mathbf{x}}(h) + \gamma^{(u)}\tilde{z}(h) & \text{for dynamic controllers.}\end{cases} \tag{3.13}$$

Thereby the closed loop dynamics for the subsequence $\tilde{\mathbf{x}}(h)$ is given by:

$$\begin{bmatrix}\tilde{\mathbf{x}}(h+1)\\ \tilde{z}(h+1)\end{bmatrix} = (\tilde{A} + \tilde{b}\gamma)\begin{bmatrix}\tilde{\mathbf{x}}(h)\\ \tilde{z}(h)\end{bmatrix}, \tag{3.14}$$

where $\gamma = [\gamma^{(x)}\ \gamma^{(u)}]$, and $\gamma^{(u)} = 0$ for memoryless controllers, $\gamma^{(u)} \in \mathbb{R}$ for dynamic controllers.

The stability properties of linear system (3.8) under the action of the Zero Order Hold controllers in Equation(3.13) are easily related to the closed loop dynamics of the discrete time system (3.14). Assume, for instance, that the closed loop dynamical matrix of system (3.14) has a spectral radius ρ. In this case there exists a constant M such that

$$\left\|\begin{bmatrix}\tilde{\mathbf{x}}(h)\\ \tilde{z}(h)\end{bmatrix}\right\| \leq M\rho^h, \forall h \geq 0.$$

At a generic time $t \in]hpT_c, (h+1)pT_c[$, the state of the continuous time system can be written as follows:

$$\mathbf{x}(t) = e^{A(t-hpT_c)}\tilde{\mathbf{x}}(h) + (\int_0^{t-hpT_c} e^{As}ds)\mathbf{b}\tilde{z}(h).$$

Hence,

$$\begin{aligned}||\mathbf{x}(t)|| &\leq ||e^{A(t-hpT_c)}||\ ||\tilde{\mathbf{x}}(h)|| + ||(\int_0^{t-hpT_c} e^{As}ds)b||\ ||\tilde{z}(h)|| \leq\\ &\leq \left(||e^{A(t-hpT_c)}|| + (\int_0^{t-hpT_c} ||e^{As}||ds)\ ||b||\right) M\rho^{\lfloor\frac{t}{pT_c}\rfloor}.\end{aligned} \tag{3.15}$$

It is possible to derive a bound for the exponential matrix (see [48]) as follows:

$$||e^{At}|| = ||U^{-1}e^{Jt}U|| \leq \chi||e^{Jt}||,$$

where $\chi = ||U^{-1}||\ ||U||$.

Considering the ∞ norm, we can write:

$$||e^{Jt}|| \leq r\ e^{\alpha(A)t} \max_{0\leq f\leq r-1} \frac{t^f}{f!},$$

where $r = \max\{r_1,\ldots,r_l\}$ and $\alpha(\cdot)$ denotes the spectral abscissa (i.e. the maximum real part of the eigenvalues) of the argument matrix. Hence,

$$||e^{At}|| \leq \chi r\ e^{\alpha(A)t} \max_{0\leq f\leq r-1} \frac{t^f}{f!}.$$

Getting back to Equation (3.15), it is possible to write:

$$\begin{aligned} ||x(t)|| &\leq K(p)e^{\frac{\log\rho}{pT_c}t} \\ K(p) &= M\chi r \max_{0\leq f\leq r-1} \frac{(pT_c)^f}{f!}(e^{\alpha(A)pT_c} + \frac{1}{\alpha(A)}(e^{\alpha(A)pT_c} - 1)). \end{aligned} \quad (3.16)$$

It is possible to summarize this discussion in the following:

Lemma 1 *The linear systems (3.8) under the action of the zero-order-hold controllers (3.13) are exponentially stable if the discrete time systems (3.14) are. Moreover, if the state of the closed loop matrix $\tilde{A} + \tilde{\mathbf{b}}\gamma$ has spectral radius ρ, then the continuous time system converges with exponential decay rate of at least $-\frac{\log\rho}{pT_c}$.*

The problem of guaranteeing stability and convergence with a given decay rate of the continuous time system can be cast into a pole assignment problem of a discrete-time linear time invariant system. Under controllability assumption the latter problem has always feasible solutions for the class of dynamic controllers, because the entire state, inclusive of $\tilde{\mathbf{x}}(h)$ and $\tilde{z}(h)$, is assumed to be accessible. This is not necessarily the case for the class of memoryless controllers, for which only the partial information $\tilde{\mathbf{x}}(h)$ is available for feedback.

It is worth observing that both the exponential decay rate $-\frac{\log\rho}{pT_c}$ and the constant $K(p)$ capture important aspects of the system performance. The former is a measure of the convergence speed while the latter provides guaranteed bounds on the transient behaviour. For fixed ρ, both parameters are

negatively affected by longer values of p because the system evolves in open loop for longer times in the inter-sampling intervals. More precisely, the exponential decay rate is diminished slowing down convergence, while $K(p)$ increases allowing worse transient behaviours.

3.2.1 Robustness metric

The notion of robustness proposed here is a generalization of the classical gain margin and it can be defined for a generic linear system parameterized by a set of variables.

Definition 1 (Stability center and stability radius) *Consider a linear discrete time system described by:*

$$\mathbf{x}(h+1) = \hat{A}(\gamma)\mathbf{x}(h)$$

where $\gamma \in \mathbb{R}^d$ is a vector of parameters. Let Γ be the set parameters such that $\forall\gamma \in \Gamma$ the system is asymptotically stable. The stability center $\gamma^{(c)}$ is the Chebychev center of Γ. The stability radius μ is the distance of $\gamma^{(c)}$ from $\mathbb{R}^d \backslash \Gamma$.

For a given choice of activation periods each loop achieves a radius $\mu^{(i)}$; the robustness of the entire collection $\mathcal{S}$ can be characterized by $\mu = \min_i \mu^{(i)}$. Therefore, a possible design goal is to choose such activation periods $p^{(1)}, \ldots, p^{(m)}$ as to maximize μ.

The stability radius is a generalization of the gain margin for multi-variable (Multiple Input Multiple Output) systems. More precisely it accounts for multiple control parameters $\gamma_1, \ldots, \gamma_d$.

Assume two choices of the activation periods lead to closed loop systems with different stability radii. Then the one with larger stability radius is expected to remain stable under larger perturbations that may manifest in the following forms

- truncation errors due to finite precision in the implementation arithmetics of the controller task
- quantization errors on sampled data from sensors and to actuators
- additive or multiplicative noise on sensor data and actuator outputs
- system deviations from the LTI (Linear Time Invariant) model

at least in a first approximation where perturbations can be modeled as a change in the observed loop gains.

Albeit a fundamental requirement of any system's design, stability is not sufficient to capture the entire performance specification of a system. As mentioned above, there are at least two further parameters one typically needs to specify: the convergence speed and the quality of the transient evolution.

It is argued that the stability radius actually provides also a measure of how robustly a given performance specification is met. This will be illustrated focusing on the speed of convergence metric.

Focusing the attention on the former, Lemma 1 relates the exponential decay rate of the continuous time system to the spectral radius ρ of the closed loop matrix $\tilde{A}+\tilde{b}\gamma$ of the discrete time system (3.14). In particular for system trajectories to decay with rate $\beta > 0$ (i.e. $||x(t)|| \leq Me^{-\beta t}$), it is sufficient that $\rho \leq e^{-\beta pT_c}$. This is equivalent to requiring that matrix $e^{-\beta pT_c}(\tilde{A}+\tilde{b}\gamma)$ be Schur stable. This is equivalent to requiring that matrix $e^{-\beta pT_c}(\tilde{A}+\tilde{b}\gamma)$ be Schur stable. The stability radius of the latter system quantifies how robustly the required performance metric (convergence rate $\geq \beta$) is met by the closed loop system.

3.3 Computing the stability radius

Basically, for each system $\mathcal{S}^{(i)}$ composing the collection, the stability radius can be derived by studying the values of the gains γ such that the $\tilde{A}+\tilde{\mathbf{b}}\gamma$ matrix is Schur stable (also in this context the (i) superscript denoting the different systems of the collection has been removed to simplify notation). Pre-multiplying $\tilde{A}+\tilde{\mathbf{b}}\gamma$ by $\left[\begin{smallmatrix} U^{-1} & 0 \\ 0 & 1 \end{smallmatrix}\right]$ and post-multiplying by $\left[\begin{smallmatrix} U & 0 \\ 0 & 1 \end{smallmatrix}\right]$, where $J = U^{-1}AU$, the study of the stability radius can be performed on:

$$\begin{bmatrix} e^{JpT_c} & U^{-1}(\int_0^{pT_c} e^{As}ds)\mathbf{b} \\ \gamma^{(\mathbf{x})}U & \gamma^{(u)} \end{bmatrix}$$

for dynamic controllers. The case of memory-less controllers is easily obtained selecting projecting on $\gamma^{(u)} = 0$.

The coefficients of the characteristic polynomial are affine functions of the gains and the region Γ of stabilizing gains can be computed *via* the Jury criterion. In the general case Γ is the intersection of nonlinear inequalities.

For the case of first order systems the region Γ is a polyhedral, hence the computation of the stability radius is much easier. Moreover the solution

to the optimization problem can be closely approximated in closed form, thus providing useful insights on the dependencies of the solution on design parameters. For these reasons, first order systems will be analyzed in depth. Later in the section, an upper and a lower bound for the stability radius of multi-variable systems will be computed.

3.3.1 The case of first order systems

For first order systems the Jury criterion provides a set of linear inequalities. It is useful to treat separately the case of memoryless and dynamic controller.

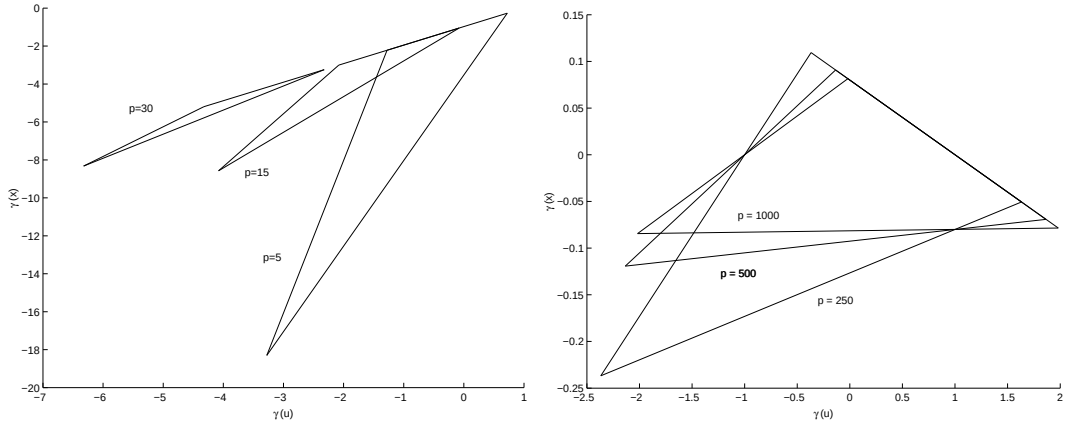


Figure 3.1 Stability region Γ for two systems. Left: $\lambda_1 = 48.8$, $b = 50$, $T_c = 10^{-3}s$. Right: $\lambda_1 = -4$, $b = 50$, $T_c = 10^{-3}s$.

Dynamic controller. The characteristic polynomial of the $\tilde{A} + \tilde{b}\gamma$ matrix is given by:

$$z^2 - (e^{\lambda_1 pT_c} + \gamma^u)z + e^{\lambda_1 pT_c}\gamma^u - (\int_0^{pT_c} e^{\lambda_1 s} ds)b\gamma^{(x)}.$$

The application of the Jury criterion yields:

$$\begin{aligned} e^{\lambda_1 pT_c}\gamma^{(u)} - (\int_0^{pT_c} e^{\lambda_1 s}ds)b\gamma^{(x)} &< 1 \\ (e^{\lambda_1 pT_c}-1)\gamma^{(u)} - (\int_0^{pT_c} e^{\lambda_1 s}ds)b\gamma^{(x)} &> -1+e^{\lambda_1 pT_c} \qquad (3.17) \\ (e^{\lambda_1 pT_c}+1)\gamma^{(u)} - (\int_0^{pT_c} e^{\lambda_1 s}ds)b\gamma^{(x)} &> -1-e^{\lambda_1 pT_c} \end{aligned}$$

(3.18)

In figure 3.1 examples of stability regions Γ for different values of λ_1 and of p are reported. As shown in the picture Γ is in this case a triangle (in the case of static controller it was an interval). The triangle shrinks for increasing values of p.

For open loop unstable systems, as p tends to infinity the three vertices collapse onto the point having coordinates $\gamma^{(u)} = -e^{\lambda_1 pT_c}$, $\gamma^{(x)} = -\frac{e^{\lambda_1 pT_c}\lambda_1}{b}$; therefore the triangle tends to vanish. On the contrary, for open loop stable systems the triangle shrinks to an "asymptotical" stability region (the vertexes tend to distinct values). These qualitative considerations are confirmed by the following:

Proposition 1 *The stability radius of the system is given by:*

$$\mu = \begin{cases} \frac{\lambda_1}{e^{\lambda_1 pT_c}(\lambda_1+|b|)-|b|} & \text{if } \lambda_1 > 0 \\ \frac{2\lambda_1}{e^{\lambda_1 pT_c}(\lambda_1+2|b|)+\lambda_1-2|b|} & \text{if } \lambda_1 < 0 \\ \frac{1}{1+pT_c|b|} & \text{if } \lambda_1 = 0 \end{cases} \qquad (3.19)$$

To prove the above, it is useful to recall the following simple result:

Lemma 2 *Let $\mathcal{P}$ be a polyhedral set defined by a set of linear inequalities:*

$$h_i\mathbf{u} \leq f_i, i = 1,\ldots,c.$$

The Chebychev center of $\mathcal{P}$ in the norm $||.||$ is the solution of the following linear program:

$$\begin{array}{ll} \max & \mu \\ \textit{subject to} & h_i\mathbf{u} + \mu\ ||h_i||_* \leq f_i, i = 1,\ldots,m \\ & \mu \geq 0 \end{array} \qquad (3.20)$$

Proof of Proposition 1:
The proof is given for the case $\lambda > 0$ (the other cases follow similar arguments). With regard to the ∞ norm (whose dual is the 1-norm), the application of Lemma 2 to the set defined in Equation 3.17 yields the following linear program:

$$\begin{array}{ll} \max & \mu \\ \text{subject to} & H \begin{bmatrix} \gamma^{(u)} \\ \gamma^{(x)} \\ \mu \end{bmatrix} \leq \mathbf{q} \\ & \mu \geq 0 \end{array}$$

where

$$H = \begin{bmatrix} e^{\lambda_1 pT_c} & -\frac{e^{\lambda_1 pT_c}-1}{\lambda_1} b & e^{\lambda_1 pT_c} + \frac{e^{\lambda_1 pT_c}-1}{\lambda_1}|b| \\ 1 - e^{\lambda_1 pT_c} & \frac{e^{\lambda_1 pT_c}-1}{\lambda_1} b & e^{\lambda_1 pT_c} - 1 + \frac{e^{\lambda_1 pT_c}-1}{\lambda_1}|b| \\ -1 - e^{\lambda_1 pT_c} & \frac{e^{\lambda_1 pT_c}-1}{\lambda_1} b & e^{\lambda_1 pT_c} + 1 + \frac{e^{\lambda_1 pT_c}-1}{\lambda_1}|b| \end{bmatrix}$$

$$\mathbf{q} = \begin{bmatrix} 1 \\ 1 - e^{\lambda_1 pT_c} \\ 1 + e^{\lambda_1 pT_c} \end{bmatrix}.$$

The dual problem is given by:

$$\begin{array}{ll} \min & [\mathbf{q}^T \ 0]\mathbf{y} \\ \text{subject to} & \begin{bmatrix} & & 0 \\ H^T & & 0 \\ & & -1 \end{bmatrix} \mathbf{y} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ & \mathbf{y} \geq 0 \end{array}$$

The solution $H^{-1}\mathbf{q}$ is primal feasible. It is easily seen that the complementary slackness solution given by

$$\begin{bmatrix} H^{-T} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ 0 \end{bmatrix}$$

is dual feasible. Hence the two solutions are an optimal pair. The claim of the proposition easily follows. •

The above confirms the qualitative intuition that the stability radius is, a decreasing function of p that tends to 0 for open loop unstable systems and to a minimum guaranteed value for open loop stable systems.

Memoryless controller. The case of static controllers is derived considering the projection of the region Γ on $\gamma^{(u)} = 0$. Considering the regions in Fig. 3.1 this intersection is a segment, possibly degenerate to a point or the empty set. More formally the result can be summarized in the following:

Proposition 2 *The following statements hold:*

1. *if $\lambda_1 = 0$ the closed loop system is asymptotically stable for all $\gamma^{(x)}$ such that $-\frac{1}{pT_c} < b\gamma^{(x)} < 0$;*
2. *if $\lambda_1 \neq 0$ the set of stabilizing solutions for $\gamma^{(x)}$ is given by: $-\frac{\lambda_1}{e^{\lambda_1 pT_c}-1} < b\gamma^{(x)} < -\lambda_1$. If the open loop system is asymptotically stable ($\lambda_1 < 0$) then this set is non empty (a trivial stabilizing solution is $\gamma^{(x)} = 0$). If the system is open loop unstable($\lambda_1 > 0$) then the set of stabilizing solutions for $\gamma^{(x)}$ is non empty if and only if $p < \frac{\log 2}{\lambda_1 T_c}$.*

A simple corollary of the above is:

Corollary 1 *The stability radius of the system is*

$$\mu = \begin{cases} \frac{\lambda_1}{2|b|(e^{\lambda_1 pT_c}-1)}(2 - e^{\lambda_1 pT_c}) & \lambda_1 \neq 0 \\ \frac{1}{2|b|pT_c} & \lambda_1 = 0 \end{cases} \tag{3.21}$$

The stability center is given by:

$$\gamma^{(c)} = \begin{cases} -\frac{\lambda_1}{2}\frac{e^{\lambda_1 pT_c}}{b(e^{\lambda_1 pT_c}-1))} & \lambda_1 \neq 0 \\ -\frac{1}{2bpT_c} & \lambda_1 = 0 \end{cases} \tag{3.22}$$

The stability radius μ thus obtained is a decreasing function of the activation period p of the process. If the system is open loop unstable ($\lambda_1 > 0$) μ vanishes when p overcomes a threshold value. On the contrary, if $\lambda_1 < 0$ there is a minimum value for $\mu = -\frac{\lambda_1}{|b|}$ that is inherently guaranteed by the system.

As one would expect, dealing with a static output (state $\tilde{z}$ is not accessible) feedback stabilization, the existence of the solution is not guaranteed in case of open loop unstable system. This marks an important difference from the case of dynamic controllers for which there exists a stabilizing solution for any choice of p. This point is well illustrated by Figure 3.2, where the stability radii obtained with memoryless and dynamic controllers are plotted as functions of the activation period of the task.

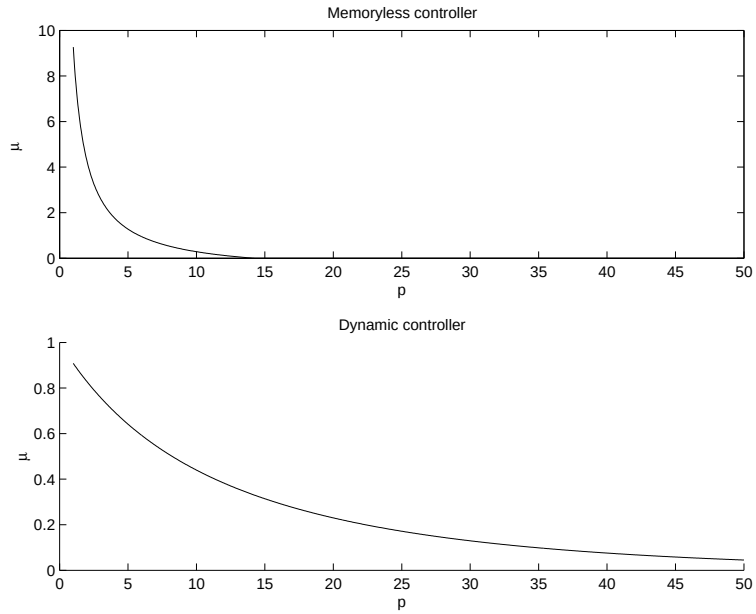


Figure 3.2 Stability radius as a function of the activation period p for a first order system: $\lambda_1 = 48.8$, $T_c = 1ms$, $b = 50$. Top: memoryless controller, Bottom: dynamic controller

Also notice that, due to projection, the stability radius is computed into a lower dimensional subspace (specifically $\mathbb{R}$). For this reason it may happen that static controllers exhibit a larger stability radius than the corresponding dynamic controller, a somewhat unintuitive result.

3.3.2 Results for multi-variable systems

Although the application of the Jury criterion is possible in principle for systems of any order, in practice it is a viable solution only for low order systems. As a matter of fact, its application produces in the general case, a region in the space of the gains given by the intersection of nonlinear inequalities, which is generally hard to study. However, it is possible to find "easy" polyhedral bounds that can be used in the design problem. One external and one internal polyhedral approximation will be described next, providing respectively an upper and lower bound for the stability radius.

3.3.2.1 An upper bound

An upper bound for the stability radius can be found by exploiting the following:

Lemma 3 *A necessary condition for a matrix X to be Schur stable is that $det(X) < 1$.*

In the case of dynamic controllers (the case of memoryless controller is a special case), one easily gets that the determinant of $\tilde{A} + \tilde{b}\gamma$ is equal to

$$det\left(\begin{bmatrix} e^{J_1 pT_c} & 0 & \dots & 0 & (\int_0^{pT_c} e^{J_1 s} ds) V_1^T \mathbf{b} \\ 0 & e^{J_2 pT_c} & \dots & 0 & (\int_0^{pT_c} e^{J_2 s} ds) V_2^T \mathbf{b} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & e^{J_l pT_c} & (\int_0^{pT_c} e^{J_l s} ds) V_l^T \mathbf{b} \\ \gamma^{(x)} U_1 & \gamma^{(x)} U_2 & \dots & \gamma^{(x)} U_l & \gamma^{(u)} \end{bmatrix} \right)$$

where

$$U = [U_1 U_2 \dots U_l] = [\mathbf{u}_1 \mathbf{u}_2 \dots \mathbf{u}_n]$$
$$V = \begin{bmatrix} V_1^T \\ V_2^T \\ \dots \\ V_l^T \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \dots \\ \mathbf{v}_n^T \end{bmatrix}$$

are such that $U = V^{-1}$ and $A = UJV$. The computation of the determinant above yields an affine expression in the gains:

$$det(\tilde{A} + \tilde{b}\gamma) = \gamma^{(x)} \mathbf{q}(p) + \gamma^{(u)} \prod_{k=1}^{l} (e^{\lambda_k pT_c})^{r_k}, \tag{3.23}$$

where the expression for vector $\mathbf{q}(p)$ is easy to find, as a function of p, but not very clear to write in closed form in the general case. Now it is possible to state the following:

Proposition 3 *An upper bound for the stability radius is given by:*

$$\overline{\mu} = \begin{cases} \frac{1}{||\mathbf{q}(p)||_1} & \textit{for memoryless controllers} \\ \frac{1}{||\mathbf{q}(p)||_1 + |\prod_{k=1}^{l} (e^{\lambda_k pT_c})^{r_k}|} & \textit{for dynamic controllers} \end{cases} \tag{3.24}$$

Proof:
As mentioned above, it is possible to obtain an upper bound by applying the necessary condition for asymptotical stability:

$$|det(A + \tilde{\mathbf{b}}\gamma)| < 1.$$

Using expression 3.23, the stabilizing gains, for the case of dynamic controller, belong to the slab:

$$-1 < \gamma^{(x)}\mathbf{q}(p) + \gamma^{(u)} \prod_{k=1}^{l} (e^{\lambda_k p T_c})^{r_k} < 1.$$

An upper bound for the stability radius is given by the radius of the maximum ball contained in the slab. By Lemma 2 such radius is given by the optimal solution of the following linear program:

$$\begin{array}{ll} \max & \mu \\ \text{subject to} & \gamma^{(x)}\mathbf{q}(p) + \gamma^{(u)} \prod_{k=1}^{l} (e^{\lambda_k p T_c})^{r_k} + \mu(||\mathbf{q}(p)||_1 + |\prod_{k=1}^{l} (e^{\lambda_k p T_c})^{r_k}|) < 1 \\ & -\gamma^{(x)}\mathbf{q}(p) - \gamma^{(u)} \prod_{k=1}^{l} (e^{\lambda_k p T_c})^{r_k} + \mu(||\mathbf{q}(p)||_1 + |\prod_{k=1}^{l} (e^{\lambda_k p T_c})^{r_k}|) < 1 \\ & \mu \geq 0. \end{array}$$

whose dual is:

$$\begin{array}{ll} \min & [1\ 1\ 0] \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \\ \text{subject to} & \begin{bmatrix} \mathbf{q}(p)^T & -\mathbf{q}(p)^T & \mathbf{0} \\ \prod_{k=1}^{l} (e^{\lambda_k p T_c})^{r_k} & -\prod_{k=1}^{l} (e^{\lambda_k p T_c})^{r_k} & 0 \\ ||\mathbf{q}(p)||_1 + |\prod_{k=1}^{l} (e^{\lambda_k p T_c})^{r_k}| & ||\mathbf{q}(p)||_1 + |\prod_{k=1}^{l} (e^{\lambda_k p T_c})^{r_k}| & -1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \\ & y_i \geq 0. \end{array}$$

Now, it is easy to see that the solution given by $\gamma^{(x)} = 0$, $\gamma^{(u)} = 0$ and $\mu = \frac{1}{||\mathbf{q}(p)||_1 + |\prod_{k=1}^{l} (e^{\lambda_k p T_c})^{r_k}|}$ is primal feasible. The proof of the optimality is ended observing that the complementary slackness solution:

$$y_1 = y_2 = \frac{1}{2(||\mathbf{q}(p)||_1 + |\prod_{k=1}^{l} (e^{\lambda_k p T_c})^{r_k}|)} \text{ and } y_3 = 0,$$

is dual feasible. •

Example 1. In the special case of all Jordan blocks having dimension 1, it is easy to compute the $\mathbf{q}(p)$ vector, which is given by:

$$\mathbf{q}(p) = -\sum_{i=1}^{n} \frac{e^{\lambda_i p T_c} - 1}{\lambda_i} \prod_{j \neq i} e^{\lambda_j p T_c} \hat{\mathbf{b}}_i. \tag{3.25}$$

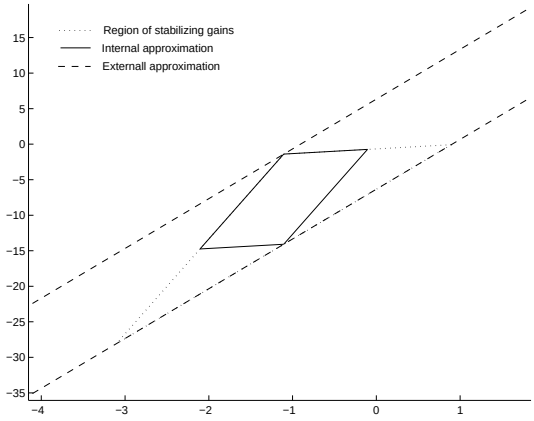


Figure 3.3 Internal and external approximations used for the computation of lower and upper bounds for a first order system.

where $\hat{\mathbf{b}}_i = \mathbf{u}_i \mathbf{v}_i^T \mathbf{b}$. Therefore, in this special case, the stability radius is upper-bounded by:

$$\overline{\mu} = \frac{1}{||\sum_{i=1}^{n}(\frac{e^{\lambda_i pT_c}-1}{\lambda_i}\prod_{j\neq i} e^{\lambda_j pT_c}\hat{b}_i)||_1 + \prod_{i=1}^{n} e^{\lambda_i pT_c}}.$$

A looser upper bound for the stability radius for dynamic controllers is given by: $\frac{1}{|\prod_{k=1}^{l}(e^{\lambda_k pT_c})^{r_k}|}$. It is worth observing that if $|\prod_{k=1}^{l}(e^{\lambda_k r_k T_c})| > 1$ then the stability radius is upper-bounded by a function decreasing with exponential rate with respect to p.

3.3.2.2 A lower bound

A lower bound for the stability radius can be derived by using the following:

Lemma 4 *Consider a matrix X and let $z^n + q_n z^{n-1} + \ldots + q_1 z + q_0$ be its characteristic polynomial. The matrix is Schur stable if*

$$\sum_{i=0}^{n-1} |q_i| < 1.$$

Proof:
As a first consideration observe that any matrix norm is an upper bound for

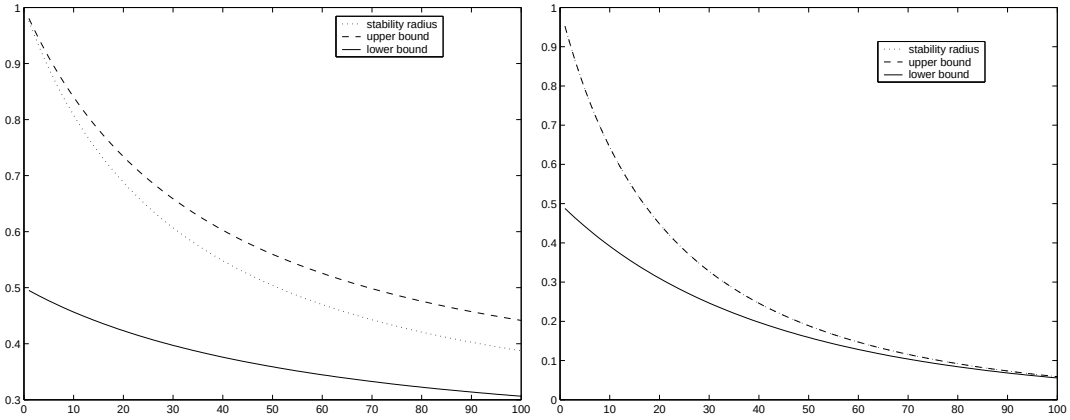


Figure 3.4 Upper and lower bounds for the stability radius of a first order system. Left: open loop stable system ($\lambda_1 = -10$). Right: open loop unstable system ($\lambda_1 = 20$)

the spectral radius. In particular $\rho(X) \leq ||X||_\infty$. If the X matrix is written in standard companion form the result easily follows (we recall that the ∞ norm of a matrix X is given by $\max_i \sum_j |x_{i,j}|$). • As pointed out earlier, the

coefficients of the $\tilde{A} + \tilde{b}\gamma$ are affine functions of the gains:

$$q_i = q_{i,0} + \gamma \mathbf{q}_i,$$

where vectors $\mathbf{q}_i$ and scalars $q_{i,0}$ can be found as functions of the p activation period. Applying Lemma 4, it is possible to define a polyhedron entirely contained in the region of stabilizing gains. Let $\mathcal{E} = \{1, 2, \ldots, n\}$ be an index set. Define a complete enumeration $\mathcal{Q}_0$, $\mathcal{Q}_1$, …, $\mathcal{Q}_{2^n}$ of all possible subsets of $\mathcal{E}$.

Define $\Theta_{i,0}$ and Θ_i as follows:

$$\Theta_{j,0} = \sum_{i \in \mathcal{Q}_j} q_{i,0} - \sum_{i \in \mathcal{P} \setminus \mathcal{Q}_j} q_{i,0}$$
$$\Theta_j = \sum_{i \in \mathcal{Q}_j} \mathbf{q}_i - \sum_{i \in \mathcal{P} \setminus \mathcal{Q}_j} \mathbf{q}_i$$

Now it is possible to compute numerically a lower bound $\underline{\mu}$ of the stability radius by solving the following linear program:

$$\begin{array}{ll} \max_{\mu,\gamma} & \mu \\ \text{subject to} & \gamma\Theta_j + ||\Theta_j||_1 \leq 1 - \Theta_{j,0} \\ & \mu \geq 0 \end{array} \qquad (P.1)$$

Example 2. As an example application of the above result, consider again a first order system with dynamic controller whose parameters are as follows:

$\lambda_1 = 20, T_c = 1ms, p = 5, b = 30$. The region of stabilizing gains is depicted in dotted lines in Figure 3.3. In the same figure, the external approximation for the region used to compute the upper bound is the slab depicted in dashed lines. Finally, the continuous line is used to depict the internal approximation proposed above for the computation of a lower bound of the stability radius. In Figure 3.4 the exact stability radius is compared with its upper and lower bound. For open loop unstable system the exact value and the lower bounds are coincident.

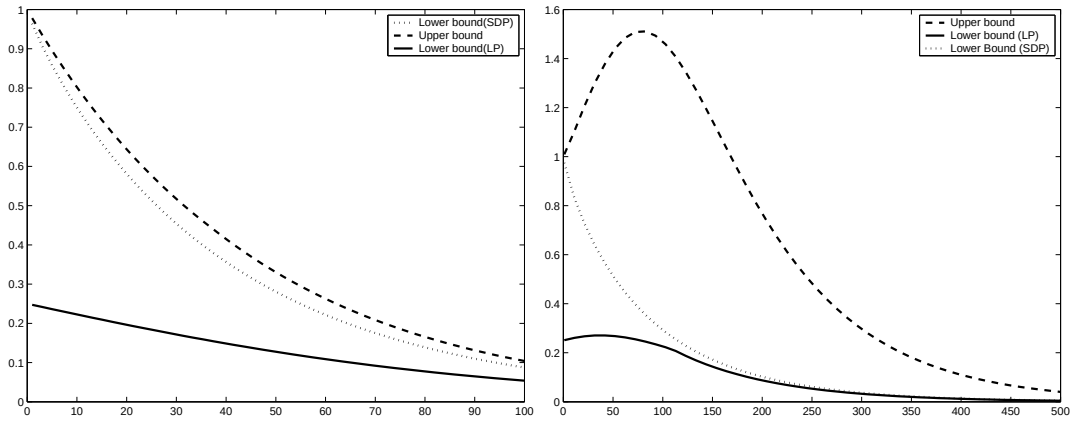


Figure 3.5 Upper and lower bounds for the stability radius of a second order system. Left: system has eigenvalues $\lambda_1 = 1$, $\lambda_2 = 20$. Right: system has eigenvalues $\lambda_1 = 10$, $\lambda_2 = -20$.

Example 3. As an example of multi-variable systems, we consider a second order diagonalizable system controlled by a dynamic controller. Let U and V be the matrices of right and left eigenvectors:

$$\begin{aligned} U &= [\mathbf{u}_1 \mathbf{u}_2] \\ V &= \left[\mathbf{v}_1^T \ \mathbf{v}_2^T\right] \\ A &= U diag([\lambda_1, \lambda_2])V. \end{aligned}$$

The upper bound can be computed in closed form and it is given by:

$$\overline{\mu} = \frac{1}{e^{\lambda_1 p T_c} e^{\lambda_1 p T_c} + ||\frac{e^{\lambda_1 p T_c}-1}{\lambda_1} e^{\lambda_2 p T_c} \hat{b}_1 + \frac{e^{\lambda_2 p T_c}-1}{\lambda_2} e^{\lambda_1 p T_c} \hat{b}_2)||_1},$$

where $\hat{\mathbf{b}}_i = \mathbf{u}_i \mathbf{v}_i^T \mathbf{b}$. As far as the lower bound is concerned, vectors $\mathbf{q}_i$ and scalars $q_{i,0}$ are given by:

$$\begin{aligned}
q_{0,0} &= 0 \\
q_{1,0} &= e^{\lambda_1 p T_c} e^{\lambda_2 p T_c} \\
q_{2,0} &= -e^{\lambda_1 p T_c} - e^{\lambda_2 p T_c} \\
\mathbf{q}_0 &= \begin{bmatrix} \frac{e^{\lambda_1 p T_c}-1}{\lambda_1} e^{\lambda_2 p T_c} \hat{b}_1 + \frac{e^{\lambda_2 p T_c}-1}{\lambda_2} e^{\lambda_1 p T_c} \hat{b}_2 \\ -e^{\lambda_1 p T_c} e^{\lambda_2 p T_c} \end{bmatrix} \\
\mathbf{q}_1 &= \begin{bmatrix} \frac{e^{\lambda_1 p T_c}-1}{\lambda_1} \hat{b}_1 + \frac{e^{\lambda_2 p T_c}-1}{\lambda_2} \hat{b}_2 \\ e^{\lambda_1 p T_c} + e^{\lambda_2 p T_c} \end{bmatrix} \\
\mathbf{q}_2 &= \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}
\end{aligned} \tag{3.26}$$

Using these expressions, it is possible to set up problem *P.*1 and solve it numerically for different values of p. For both examples we chose $U = \left[\begin{smallmatrix} 0.5028 & 0.4289 \\ 0.7095 & 0.3046 \end{smallmatrix}\right]$, $\mathbf{b} = \left[\begin{smallmatrix} 0.5467 \\ -0.591 \end{smallmatrix}\right]$.

Figure 3.5 shows some example of upper and lower bounds on second order systems.

As it is possible to see from the examples above, the upper and lower bounds are particularly close to each other (for p sufficiently large) if the system is strongly unstable. In these cases, it appears a reasonable choice to use $\overline{\mu}$ as a good heuristic in optimization processes where the stability radius is used as cost function.

3.4 Tackling the design problem

Two fundamental questions are raised in the context of the platform based methodology proposed in this thesis:

- is a chosen platform able to sustain a performance specification?,
- how are design parameters to be chosen to maximize performance on a given platform?

The first question is useful during the first phase of the design, when the specification of the system's performance has to be translated into the specification of a platform. The second question essentially corresponds to a refinement step and it is performed during the last phases of the development,

when a finer characterization for the architecture performance is possible and the system performance can be optimized. The performance metric we advocate in this chapter is the minimum stability radius of the collection: $\mu = min_{i=1,\ldots,n}\mu_i$.

In this framework, deciding whether a platform is powerful enough to sustain a given performance specification amounts to solving the following feasibility problem:

$$\begin{cases} \text{find an m-uple} & p^{(1)}, \ , p^{(2)}, \ \ldots, \ p^{(m)} \\ \text{subject to} & \sum_i \frac{e^{(i)}}{p^{(i)}} \leq U_l \\ & \mu \geq \mu_0 \geq 0. \\ & p^{(i)} \in \mathbb{N} \end{cases} \qquad (P.2)$$

where μ is the stability radius of the collection. Similarly, optimizing the system performance of an assigned platform amounts to solving the following optimization problem:

$$\begin{cases} \max_{p^{(1)},p^{(2)},\ldots,p^{(m)}} & \mu \\ \text{subject to} & \sum_i \frac{e^{(i)}}{p^{(i)}} \leq U_l \\ & \mu \geq \mu_0 \geq 0 \\ & \\ & p^{(i)} \in \mathbb{N} \end{cases} \qquad (P.3)$$

It is worth to remark that requiring a minimum guaranteed value $\mu_0 > 0$ for the stability radius implies that all systems of the collection have to be asymptotically stable. Moreover, it is useful that μ_0 be greater than the stability radii inherently guaranteed by open loop stable systems. If this assumption were not made, it would be possible to choose any value for the activation period of the tasks controlling open loop stable systems and the resulting design space would become unbounded. On the other hand, there is no apparent convenience in designing a controller for a subsystem whose open-loop performance are considered as acceptable.

Decision variables are partly continuous (the feedback gains) and partly integers (the tasks activation periods). However, the chosen performance metric naturally induces the selection of the gains on the stability center. Moreover, for the sake of simplicity, we will assume that T_c and U_l are fixed. Therefore, the only considered decision variable (i.e. free parameters) will be the integer periods $p^{(i)}$. In the following results, the design problem is tackled by considering the relaxation of the problem to real values for the periods. By the end of the section, a simple branch and bound scheme is illustrated that permit us to recover the integrality of the activation periods.

3.4.1 The continuous relaxation

Recalling the definition of the stability radius of the collection ($\mu = \min_i \mu^{(i)}$) we can re-write the feasibility problem *P.* 2 as follows:

$$\begin{cases} \text{find an m-uple} & p^{(1)}, \ , p^{(2)}, \ \ldots, \ p^{(m)} \\ \text{subject to} & \sum_i \frac{e^{(i)}}{p^{(i)}} \leq U_l \\ & \mu^{(i)} \geq \mu \geq \mu_0 \geq 0 \\ & p^{(i)} \in real^+ \end{cases}$$

Notice that periods have been relaxed to positive reals.

A similar operation is possible for the Problem *P.* 3. In this formulation $\mu^{(i)}$ are functions of $p^{(i)}$ parametrized by the dynamical parameters of system $\mathcal{S}^{(i)}$. Introduce functions $\sigma^{(i)}$ defined as follows:

$$\sigma^{(i)}(\mu) = \max\{p \in \mathbb{R}^+, \text{ such that } \mu^{(i)}(p) \geq \mu\}. \tag{3.27}$$

Function $\sigma^{(i)}(\mu)$ is the maximum activation period that can be used on the task τ_i controlling $\mathcal{S}^{(i)}$ to achieve ate least μ for the stability radius. Clearly, it need not be defined for all values of μ. Introduce the function $H(\mu)$ defined as follows:

$$H(\mu) = \sum_{i=1} \frac{e^{(i)}}{\sigma^{(i)}(\mu)} - U_l. \tag{3.28}$$

By construction $\sigma^{(i)}(\mu)$ is a decreasing function of μ and H is an increasing function of μ. We are in condition to state the following:

Fact 1 *The following statements are true:*

- *The stability radius μ_0 can be attained on a platform parametrized by $e^{(i)}, T_c, U_l$ (i.e. Problem* P. *2 is feasible) if and only if*

$$H(\mu_0) \leq 0; \tag{3.29}$$

- *If condition 3.29 holds then the $H(\mu)$ has only one zero μ_* in the set $\mu \geq \mu_0$ that is optimal solution of Problem (*P. *3). The optimal periods are given by $p_*^{(i)} = \sigma^{(i)}(\mu_*)$.*

The above is simply a different formulation the feasibility and the optimization problems. The complexity of the procedure has been moved into the computation of the $\sigma^{(i)}$ functions. To this regard, observe that if $\mu^{(i)}$ is a decreasing function of the period, then $\sigma^{(i)}$ is its inverse function. This condition occurs in many practical systems. An example is offered once again by first order systems and it will be shown next.

3.4.1.1 The case of first order systems

In this section we will show the results of the optimization for first order systems. For the sake of brevity, we will assume that all the systems composing the collection are open-loop unstable: $\lambda_1^{(i)} > 1, \forall i = 1, \ldots, m$. We will deal separately with the case of memoryless and dynamic controllers.

Memoryless controller The expression for the stability radius is

$$\mu^{(i)} = \frac{\lambda_1^{(i)}}{2|b^{(i)}|(e^{\lambda_1^{(i)} pT_c} - 1)}(2 - e^{\lambda_1^{(i)} pT_c})$$

and it is monotone decreasing in p. The resulting expression for $\sigma^{(i)}$ is:

$$\sigma^{(i)}(\mu) = \frac{1}{\lambda_1^{(i)} T_c} \log \frac{2\lambda_1^{(i)} + 2\mu|b^{(i)}|}{\lambda_1^{(i)} + 2\mu|b^{(i)}|}. \tag{3.30}$$

These expressions can be plugged into 3.28 and obtain the optimum value by finding the zero of $H(\mu)$. The search for the zero can be restricted to a small interval by finding "cheap" upper and lower bounds. To this purpose, it is convenient to consider the function $\omega^{(i)}(\mu) = \frac{1}{\sigma^{(i)}(\mu)}$ that represents the minimum frequency necessary to attain μ on system $\mathcal{S}^{(i)}$. Observing that $\omega^{(i)}$ is concave and strictly increasing, it is easy to show the following inequality:

$$\begin{aligned} \overline{\omega}^{(i)} &\geq \omega^{(i)} \geq \underline{\omega}^{(i)} \\ \overline{\omega}^{(i)} &= (\tfrac{3}{2} + 2|b^{(i)}|\tfrac{\mu}{\lambda_1^{(i)}})\lambda_1^{(i)} T_c \\ \underline{\omega}^{(i)} &= (\tfrac{1}{\log 2} + 2|b^{(i)}|\tfrac{\mu}{\lambda_1^{(i)}})\lambda_1^{(i)} T_c \end{aligned} \tag{3.31}$$

The use of linear approximations allows for a very fast computation of a lower and an upper bound of the optimal value μ_*. Moreover, as it is possible to see in Figure 3.6, the bounds are actually very tight. Therefore they can be used to find approximated solutions to the optimization problem.

It is interesting to take a glance at the expression of one of these approximate solutions. In particular the use of $\underline{\omega}^{(i)}$ produces an upper bound. Introduce

$$\hat{U}_l = U_l - \sum_j e^j \frac{\lambda^{(j)} T_c}{\log 2}.$$

It is easy to find:

$$\mu_* \leq \frac{\hat{U}_l}{2\sum_j e^{(j)}|b^{(j)}|}. \tag{3.32}$$

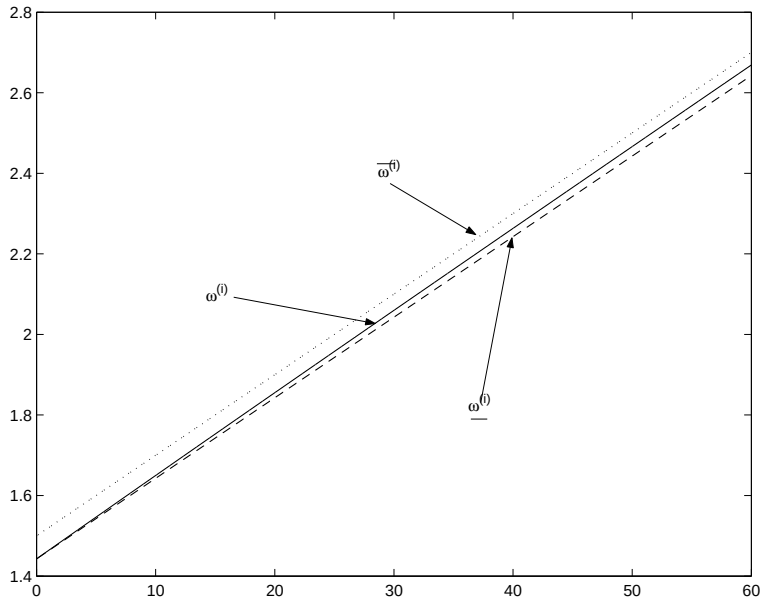


Figure 3.6 Linear bounds for $\omega^{(i)}$ for a first order system controlled by a memoryless feedback: $\lambda_1^{(i)} T_c = 1$, $b^{(i)} = 10$.

The approximated optimal values for the periods are given by:

$$\frac{1}{p_*^{(i)}} \approx \frac{\lambda^{(i)} T_c}{\log 2} + |b^{(i)}| \frac{\hat{U}_l}{2 \sum_j e^{(j)} |b^{(j)}|}. \tag{3.33}$$

For the optimal frequencies we can distinguish a term $\frac{\lambda^{(i)} T_c}{\log 2}$ necessary to achieve stability, which is summed to a weighted fraction of the reduced utilization $\hat{U}_l$.

Dynamic Controller Under the assumption that all systems $\mathcal{S}_i$ are open loop stable, we have $\mu^{(i)} = \frac{\lambda_1^{(i)}}{e^{\lambda_1^{(i)} p T_c} (\lambda_1^{(i)} + |b^{(i)}|) - |b^{(i)}|}$. The function is decreasing in p and upper bounded by 1. Hence, the $\sigma^{(i)}$ function is defined only for $\mu \leq 1$ and it is given by:

$$\sigma^{(i)}(\mu) = \frac{1}{\lambda_1^{(i)} T_c} \log \frac{\lambda_1^{(i)} + \mu |b^{(i)}|}{\mu (\lambda_1^{(i)} + |b^{(i)}|)}. \tag{3.34}$$

In this case there is not an easy linear bound for $\frac{1}{\sigma^{(i)}}$ so the search for the 0 of H has to be performed in the admissible range of μ.

3.4.1.2 The general case

Dealing with multi-variable systems the main problem is represented by the difficulty in finding the the analytical expression for $\mu^{(i)}$.

In the feasibility problem it is possible to use the lower bound $\underline{\mu}^{(i)}$ computed in Section 3.3.2.2. The price to be paid is losing necessity of Condition 3.29. Hence $H(\mu_0) \leq 0$ becomes a conservative test.

As far as the optimization problem is concerned, if the search is restricted to periods for which the $\overline{\mu}^{(i)}$ and $\underline{\mu}^{(i)}$ bounds are sufficiently close, it is possible to use one of them as a good heuristic to find suboptimal solutions.

3.4.2 The integrality constraint

As far as the feasibility of a specification on a platform is concerned, if the $\mu^{(i)}$ functions are decreasing with $p^{(i)}$ it is possible to set up a conservative procedure in two steps:

1. apply the feasibility test for the continuous relaxation $H(\mu_0) \leq 0$; if the test fails conclude unfeasibility, otherwise set periods $p^{(i)}$ to $\sigma^{(i)}(\mu_0)$;

2. truncate the periods to their floor; if

$$\sum_i \frac{e^{(i)}}{\lfloor p^{(i)} \rfloor} < U_l$$

   conclude feasibility.

For the optimization problem, it is possible to use a standard branch and bound scheme. To illustrate the algorithm it is useful to introduce some notation. $\mathcal{P} \subseteq \mathbb{N}^m$ will denote the set of period m-tuples $\mathbf{p}$ respecting the constraint inequalities; $\mu_l(\bar{\mathcal{P}})$, $\mu_u(\bar{\mathcal{P}})$ will denote respectively a lower bound and an upper bound of the objective function $\mu(\mathbf{p})$ on the region $\bar{\mathcal{P}}$. $\mathcal{L}$ will denote a list of the disjoint subregions of the solution space which are currently candidate for the solution. The μ_l function is assumed to be derived from an admissible solution. The μ_u function is required to have the property that, when applied to a set containing a single element, it returns the value of the cost function computed for that element. The branch and bound algorithm is codified in Algorithm 1. For the problem under analysis the $\mathcal{H}_i$ subregions can be obtained by fixing some of the elements of the $\mathbf{p} = [p^{(1)}, p^{(2)}, \ldots, p^{(m)}]$ vector. Let $\mathcal{I}(\mathcal{H}) \subseteq \{1, \ldots, m\}$ be the index set of periods that are fixed to

Algorithm 1
Insert $\mathcal{P}$ *into* $\mathcal{L}$
set variable μ^* *to* $\mu_l(\mathcal{P})$
choose $\mathbf{p} \in \mathcal{P}$ *s.t.* $\mu_l(\mathcal{P}) = \mu_{\mathcal{S}}(\mathbf{p})$
set variable $\mathbf{p}^* = \mathbf{p}$
repeat
  choose a region $\mathcal{H}$ *from* $\mathcal{L}$ *and remove it from* $\mathcal{L}$
  partition $\mathcal{H}$ *into non-empty disjoint subregions* $\mathcal{H}_1, \ldots, \mathcal{H}_h$
  for each $\mathcal{H}_i$
    if $\mu_u(\mathcal{H}_i) \leq \mu^*$
      discard $\mathcal{H}_i$
    else
      insert $\mathcal{H}_i$ *into* $\mathcal{L}$
      if $\mu_l(\mathcal{H}_i) > \mu^*$
        set $\mu^* = \mu_l(\mathcal{H}_i)$
        set $\mathbf{p}^* = \mathbf{p} \in \mathcal{H}_i$ *s.t.* $\mu_{\mathcal{S}}(\mathbf{p}) = \mu_l(\mathcal{H}_i)$
      endif
    endif
  end for each
until $\mathcal{L}$ *is empty*

yield $\mathcal{H}$, then a finer partition $\mathcal{H}_1, \ldots, \mathcal{H}_h$ can be obtained selecting one index z not in $\mathcal{I}(\mathcal{H})$ and fixing $p^{(1)}$ to its h different admissible values[1]. The selection of the μ_u, μ_l functions is a particularly important issue. In fact, if the bounds are not tight, the algorithm does not "discard" a good number of regions resulting into a nearly exhaustive search. On the other hand accurate bounds may not be viable if overly expensive computation is required to derive them. In order to ensure correctness of the algorithm, the lower bound $\mu_l(\mathcal{H}_i)$ is assumed to be always constructed using some feasible solution $\mathbf{p}^* \in \mathcal{H}_i$ such that $\mu(\mathbf{p}^*) = \mu_l(\mathcal{H}_i)$.

One possible upper bound $\mu_u(\mathcal{H}_i)$ can be derived using $\min_{j \in \mathcal{I}(\mathcal{H}_i)} \mu^{(j)}(p^{(j)})$, i.e. $\mu_u(\mathcal{H}_i) = \min\{\mu_u(\mathcal{H}), \min_{j \in \mathcal{I}(\mathcal{H}_i) \setminus \mathcal{I}(\mathcal{H})} \mu_{\mathcal{S}_j}(p_j)\}$. This upper bound may be quite optimistic and becomes tighter as the number of fixed periods increases. When all periods are fixed, i.e. $\mathcal{H}_i = \{\mathbf{p}\}$, it becomes $\mu_u(\mathcal{H}_i) = \mu_{\mathcal{S}}(\mathbf{p})$. Its computation cost is very low especially when periods are fixed incrementally. Different derivations for μ_u can be obtained as shown in the section devoted to the continuous relaxation.

3.5 A numerical example

In this section we present some numerical examples on the application of the proposed procedure to a set of first order systems. We consider a set of three scalar systems, whose dynamics are described by the $a^{(i)}, b^{(i)}$ parameters in Table 3.1. We will separately deal with the case of memoryless and dynamic controllers. In both cases the platform exploration will be performed on two hypothetical platforms whose parameters are reported in Table 3.2. As a performance specification we will require the stability radius to be no less than 0.1

	System 1	System 2	System 3
$a^{(i)}i$	0.3	10	500
$b^{(i)}$	-3	12	1

Table 3.1 Dynamical parameters of the controlled systems

Memoryless controller Considering the continuous relaxation, the maximum activation periods for the three tasks to attain the stability radius $\mu =$

[1] This set is finite because it is a projection of some subset of $\mathcal{P}$ which is finite. The number h varies with z.

	Platform 1	Platform 2
$e^{(1)}$	50	50
$e^{(2)}$	15	15
$e^{(3)}$	6	4
T_c	1ms	1ms

Table 3.2 Parameters of the considered platforms

0.1 as are $p^{(1)} = 958.9402$, $p^{(2)} = 59.1364$ and $p^{(3)} = 6.9215$. Considering Platform 1 the total utilization is $\sum_i \frac{e^(i)}{p^{(i)}} = 1.17 > U_l$. Hence, the problem is unfeasible on Platform 1. On Platform 2, the total utilization with the continuous periods is 0.883. The periods can be truncated to their floor preserving or improving performance. Even in this case, we obtain a total utilization equal to 0.9731. Hence, it is possible to infer that Platform 2 is powerful enough to sustain the specification.

After the selection of Platform 2, it is possible to start the optimization procedure to maximize the stability radius. The continuous relaxation yields an optimum equal to $\mu = 0.2716$ with periods $p^{(1)} = 481.73$, $p^{(2)} = 47.338$ and $p^{(3)} = 6.9044$. Using the Branch and Bound algorithm, the optimum value is $\mu_* = 0.1384$ and it is attained at $p_*^{(1)} = 764$, $p_*^{(2)} = 56$ and $p_*^{(3)} = 6$.

Dynamic controller In this case the maximum activation period to comply with the specification on the stability radius are $p^{(1)} = 1992.8$, $p^{(2)} = 162.7$ and $p^{(3)} = 22.9$. Considering the truncation of the periods to the floor integers, it is easily seen that both Platform 1 and Platform 2 can sustain the specification.

For the sake of brevity, the optimization step was performed only on Platform 1. The continuous relaxation yields $\mu = 0.4545$ and it is attained at $p^{(1)} = 345.25$, $p^{(2)} = 43.54$ and $p^{(3)} = 7.8$. In this case the Branch and Bound algorithm yields an optimum value close to the one of the continuous relaxation: $\mu_* = 0.4469$, $p^{(1)} = 346$, $p^{(2)} = 44$ and $p^{(3)} = 8$.

3.6 Future extensions

In this chapter the platform based design approach has been shown on a concrete example. The combination of the time-triggered model of computation and utilization based real-time schedulability test has proven successful in modeling the platform. We believe there is room for continuing

this work in different directions. In particular, another possible performance metric is represented by sensitivity of the state to norm-bounded noise. As a matter of fact also this metric is quite directly linked to the amount of information processed by the controller. Another line of investigation might regard the conservativeness of classical hard real-time scheduling. Is it convenient to tolerate occasional failures in updating the actuators taking advantage of higher activation rates? Even retaining an hard real-time assumption (i.e. each deadline strictly respected), it appears convenient to consider different time-triggered models for which data are released before the next activation (i.e. tasks might have a relative deadline lower than the period).

4

Numerically efficient control through a shared bus

Talkers are no good doers.
Henry VI - Shakespeare

In this chapter, we deal with control design under information constraints stemming from the use of shared communication components. There are many real-life examples where this situation occurs. For instance the same channel (or bus) can be used to convey multiple sensor readings to processing units or, dually, multiple command values to actuators. As shown in the previous chapter designing a controller with shared resources in the loop, amounts to synthesizing an appropriate control law *and* deciding the allocation of the resource by choosing a set of scheduling parameters. The way this problem can be solved depends on the approach used for scheduling.

A very popular scheduling mechanism for communication resources is based on time division multiplexing allocation (TDMA): the time during which a resource can be accessed is divided in a fixed number of equal slots that are statically assigned to each potential user. The schedule thus defined is repeated cyclically. Remarkable attempts to combine control synthesis and scheduling of communication channels *via* TDMA are in [55, 30]. For each allocation of slots, the authors find an optimal design for the closed loop system using the theory of linear periodic systems. However, different scheduling choices can be only compared with exhaustive search, which is not necessarily a viable solution in many applications.

In this chapter, we start from a lower level of abstraction, i.e. initially we formulate an optimization problem to decide on-line the allocation of each

slot of the shared resource. In the second step, taking advantage of the parametric structure of the optimization problem, the state space is partitioned in polhyedral regions: in each region the solution of the problem has a fixed schedule for the bus. Thereby, in each region it is possible to use a TDMA schedule.

The particular problem considered here is the stabilization of a multi-actuator system for which commands are issued through a shared bus. This problem is common to a number of applications; in our case, the motivation came from automotive electronics systems where a standard bus, the CAN bus, is used to link subsystems. In fault tolerant automotive applications it is very frequent to have bus configurations with a throughput of 128Kbit/s. If the plant is sampled at 2 ms, at each sample we can transmit 4 floating numbers (formatted according to the IEEE 754 standard). If the architecture has only 10 groups of actuators, each requiring 4 floating point numbers, the need for a design procedure that can solve at the same time the control problem and the scheduling of the shared resource is clear. The alternatives to the procedure we propose could be either to lower significantly the sampling frequency, which is not a good choice if the plant has highly unstable modes, or using a much more expensive technology to have (ten times) faster transmissions.

To achieve stabilization, we use Model Predictive Control (MPC). This choice allows to formulate physical constraints on the actuators and the specification of safe sets on the state's evolution. Thereby, it is possible to study in a unified framework the interplay between physical and information constraints.

In this chapter, we introduce two different ways for modeling the exclusivity condition (i.e. the fact that the bus can be allocated to one actuator at each time): the first uses a linear complementarity formulation, leading to a parametric variation of the Generalized Linear Complementarity Problem (GLCP) [78]; the second uses the theory of mixed logic dynamical systems (MLD) [9]. This is not surprising, since in the related field of hybrid systems linear complementarity formulations have been proposed [74, 57] and proven equivalent to mixed logic dynamical systems [29]. This work builds upon the PGLCP formulation, which, in this application, is particularly compact and convenient. We show an algorithm for the PGLCP solution, first proposed in [51], and demonstrate its efficiency on a numerical example.

Finally, we briefly deal with the issue of parametric dependence of the PGLCP solutions from the state from which the optimization problem is initiated. In particular, we introduce the definition of basis set, characterizing a

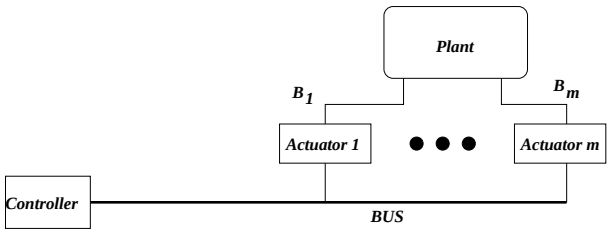


Figure 4.1 Scheme of the addressed system

region of the state space for which solutions of the PGLCP with the same set of active constraints exist. Some preliminary results on the basis sets of the PGLCP are then offered. In our context, basis sets are particularly interesting since they represent regions of the state space for which the stabilizing control law has a fixed schedule of the bus over the control horizon of the MPC.

4.1 Model Predictive Control of Control Systems with Communication Constraints

We consider the problem of stabilizing to the trivial equilibrium a discrete time linear system described by the following state equations:

$$\mathbf{x}(k+1) = A\mathbf{x}(k) + B_1\mathbf{u}_1(k) + \ldots + B_m\mathbf{u}_m(k) \tag{4.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ and $u_i \in \mathbb{R}^{m_i}$. A scheme of the addressed problem is in Figure 4.1. We will make the following assumptions:

1. pair $(A, [B_1 B_2 \ldots B_m])$ is time invariant and completely controllable;
2. a measure for the system's state $\mathbf{x}(k)$ is available at each step k;
3. control commands $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m$ are issued through a communication resource (generically denoted as "bus") which is shared amongst the different groups of actuators; hence, at a given time only one command vector $\mathbf{u}_i$ is allowed to be different from zero. In the sequel this condition will be referred to as "exclusivity condition".

State and command variables are required to evolve within a polyhedral "safe" set:

$$H\mathbf{x}(k) + G_1\mathbf{u}_1(k) + G_2\mathbf{u}_2(k) + \ldots + G_m\mathbf{u}_m(k) \leq \mathbf{g}, \tag{4.2}$$

where inequalities are imposed componentwise. For the stabilization problem to be well posed it is necessary that the point:

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{u}_1 \\ \mathbf{u}_2 \\ \dots \\ \mathbf{u}_m \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \dots \\ \mathbf{0} \end{bmatrix},$$

belongs to the safe set (i.e. $\mathbf{g} \geq \mathbf{0}$).

Remark 1 *This model is rich enough to include both physical constraints on the actuators (modeled by the definition of the safe set) and information constraints (i.e. the exclusivity condition). However, an evident limitation is the required instantaneous availability of the state. Releasing this assumption amounts to formulating the problem of finding an asymptotical estimation of the system's state when the measurements of the different sensors are conveyed through a shared bus. This problem can be formulated using techniques similar to the ones shown in this chapter. However, since measurements on sensors can be very complex data (e.g. a grabbed image), other factors like information quantization and abstraction take a prominent role. These issues will be subject to future investigations.*

The stabilization problem will be attacked using a Model Predictive Control (MPC) scheme. The reader interested to MPC in general is addressed to existing text books such as [50], or to complete review papers such as [11]. In this context, we just describe our specific application recalling some basic results which will be used throughout the chapter. MPC is a closed loop algorithm that can shortly be described as follows:

for each instant k –
acquire the state vector $\mathbf{x}(k)$;
compute sequences $\mathbf{v}_i(0), \mathbf{v}_i(1), \dots, \mathbf{v}_i(N_u - 1)$ for all i
apply only the first elements of the computed sequences: $\mathbf{u}_i(k) = \mathbf{v}_i(0)$ for all i
″.

In the chosen notation $\mathbf{u}_i$ are commands actually applied to the system, whereas $\mathbf{v}_i$ are sequences of "virtual" commands, of which only the first element is really applied to the system at each iteration. In particular, the $\mathbf{v}_i$ sequences are

computed solving a constrained optimal control problem with a finite horizon. To formulate such a problem, introduce the following cost function:

$$J(x(.), \mathbf{u}_1(.), \ldots, \mathbf{u}_m(.), h_0, h) = \sum_{j=h_0}^{h-1} (||\Psi \mathbf{x}(j)|| + \sum_{i=1}^{m} ||\Phi_i \mathbf{u}_i(j)||), \qquad (4.3)$$

where $||.||$ will henceforth denote the *infinity* norm. Ψ and Φ_i are assumed to be nonsingular matrices of size $n \times n$ and $m_i \times m_i$ respectively. The computation of the command sequence is performed, at step k, by solving the following optimization problem:

$$\begin{array}{ll}
\min_{\mathbf{v}_1(.),\mathbf{v}_2(.),\ldots,\mathbf{v}_m(.)} J(\mathbf{y}(.), \mathbf{v}_1(.), \ldots, \mathbf{v}_m(.)) & \\
\mathbf{y}(h+1) = A\mathbf{y}(h) + \sum_{i=1}^{m} B_i \mathbf{v}_i(h), & h \in [0, N_u[\\
\mathbf{y}(h+1) = A\mathbf{y}(h), & h \in [N_u + 1, N_y[\\
H\mathbf{y}(h) + \sum_{i=1}^{m} G_i \mathbf{v}_i(h) \leq \mathbf{g}, & i \in [1,m], h \in [0, \ldots, N_u[\qquad (P.4) \\
H\mathbf{y}(h) \leq \mathbf{g}, & i \in [1,m], h \in [N_u, \ldots, N_y[\\
\mathbf{y}(0) = \mathbf{x}(k), & \\
\nexists i,\ j,\ h \text{ s. t.} \mathbf{v}_i(h) \neq 0 \text{ and } \mathbf{v}_j(h) \neq 0, & i, j \in [1,m], h \in [0, \ldots, N_u[
\end{array}$$

In the above formulation the $\mathbf{y}$ vector has been introduced to describe the "virtual" evolution of the state during the optimization problem, whereas $\mathbf{x}$ denotes the actual evolution of the state variables. Clearly at each iteration $\mathbf{y}(0)$ has to be initialized with $\mathbf{x}(k)$. N_y and N_u are two constants such that $N_y \geq N_u$. A more compact formulation is obtained observing that the first and the equality constraints on the system's dynamical evolution can be eliminated, since $\mathbf{y}$ can be expressed as a function of $\mathbf{v}_i$ and $\mathbf{y}(0)$

$$\mathbf{y}(h) = A^h \mathbf{y}(0) + \sum_{j=0}^{h-1} A^{h-1-j} (\sum_{i=1}^{m} B_i \mathbf{v}_i(j)). \qquad (4.4)$$

Hence the cost function J depends only on the initial state and on the sequences:

$$J(\mathbf{y}(.), \mathbf{v}_1(.), \ldots, \mathbf{v}_m(.), 0, N_y) = J(\mathbf{y}(0), \mathbf{v}_1(.), \ldots, \mathbf{v}_m(.)).$$

In the above, with a slight abuse of notation, we also dropped the dependence from the length of the control horizon. Hence, an equivalent formulation for Problem *P.* 4 is:

$$\begin{aligned}
&\min_{\mathbf{v}_1(.),\mathbf{v}_2(.),\ldots,\mathbf{v}_m(.)} J(\mathbf{y}(.),\mathbf{v}_1(.),\ldots,\mathbf{v}_m(.)) \\
&H_0\mathbf{y}(0)+\sum_{i=1}^{m}\sum_{j=0}^{N_u-1} G_{i,\,j}\mathbf{v}_i(h)) \leq \mathbf{f} \\
&\mathbf{y}(0)=\mathbf{x}(k), \\
&\nexists i,\ j,\ h \text{s. t.} \mathbf{v}_i(h)\neq 0 \text{ and } \mathbf{v}_j(h)\neq 0, \qquad\qquad i,j\in[1,m], h\in[0,\ldots,N_u[
\end{aligned} \tag{P. 5}$$

where matrices H_0, $G_{i,\,j}$ and vector $\mathbf{f}$ can easily be computed plugging Eq. (4.4) into the constraint of the safe set.

We observe that Problem *P.* 5 significantly differs from other MPC optimization problems, such as the one presented in [10], because of the presence of the constraint in the last row that enforces the exclusivity condition. In the rest of this section we will first recall some basic result on MPC and then we will show how the exclusivity constraint can be dealt with.

4.1.1 MPC based stabilization

When dealing with MPC control, a preliminary issue regards whether the optimization problem has feasible solutions at each step or not. Feasibility is required, since, otherwise, the control law would be ill-defined and we shall assume that it is guaranteed. Such an assumption is trivially satisfied if there are not constraints on the $\mathbf{x}$ variable ($H = 0$), for the null sequences are always feasible. In the general case, we have to restrict to initial states for which a feasible solution exists. If such set is too small, it is possible to apply a "softening" technique as shown, for example, in [79].

Concerning stability of these systems, a very general result, revisited in different flavors by several authors, was proven in [32].

Theorem 2 *Consider system 4.1, assume that commands are computed applying the model predictive scheme, where the additional constraint* $\mathbf{y}(N_y) = 0$ *is included in Problem* P. *4, and assume that the resulting problem is feasible at each step. Then the resulting closed loop system is asymptotically stable.*

The following descends immediately from the proof in the cited paper:

Algorithm 2
Start from a conservative evaluation for the cost function: $J^{(0)}$*;*
for each iteration –
solve the sub-optimal problem up to a value for the cost function lower or equal than $J^{(k)}$
apply the first elements of the sub-optimal sequences $\mathbf{v}_i^*$
evaluate the cost function $J^{(k)}$ *for the shifted sequences* $\mathbf{v}_i'$
"

Corollary 2 *Consider system 4.1, where commands are computed applying the model predictive scheme, with the additional constraint* $\mathbf{y}(N_y) = 0$ *and assume that the resulting problem is feasible at each step. Assume that at step* k *the optimization problem is interrupted at a suboptimal solution* $\mathbf{v}_i^*$ *such that* $J(\mathbf{x}(k+1), \mathbf{v}_1^*(.), \ldots, \mathbf{v}_m^*(.)) \leq J(\mathbf{x}(k+1), \mathbf{v}_1'(.), \ldots, \mathbf{v}_m'(.))$*, where* $\mathbf{v}\prime_i$ *are obtained from the optimal sequences* $\mathbf{v}_i$ *computed at step* $k-1$ *by discarding the first element and appending* $\mathbf{0}$*s to the end of the sequences. Then the resulting closed loop system is asymptotically stable.*

$\mathbf{v}\prime_i$ are called *shifted* sequences. Corollary 2 can be used to construct a stabilizing control algorithm as shown in Algorithm 2 The suboptimal problem, with the additional terminal constraint $\mathbf{y}(N_y) = 0$, is the following:

$$
\begin{array}{ll}
J(\mathbf{x}(k), \mathbf{v}_1(.), \ldots, \mathbf{v}_m(.)) \leq J^{(k)} & \\
H_0\mathbf{x}(k) + \sum_{i=1}^{m} \sum_{j=0}^{N_u-1} G_{i,\,j}\mathbf{v}_i(h)) \leq \mathbf{f} & \\
A^{N_y}\mathbf{x}(k) + A^{N_y-N_u} \sum_{j=0}^{N_u-1} A^{N_u-j}(\sum_{i=1}^{m} B_i\mathbf{v}_i(j)) = \mathbf{0} & \\
\nexists i,\ j,\ h \text{s. t.} \mathbf{v}_i(h) \neq 0 \text{ and } \mathbf{v}_j(h) \neq 0, & i, j \in [1, m], h \in [0, \ldots, N_u[
\end{array}
\qquad (P.6)
$$

We remark that Problem *P.* 6 is a feasibility problem: i.e. for each $\mathbf{x}(k)$, find a feasible set of sequences $\mathbf{v}_i$ or prove that they do not exist. Our assumption is that the latter situation never occurs.

4.1.2 Dealing with the exclusivity constraint

As a preliminary remark, observe that the exclusivity constraint can be expressed as:

$$||\mathbf{v}_i(h)||\,||\mathbf{v}_j(h)|| = 0, \forall i \neq j, h \in [0, N_u[.$$

Furthermore, an equivalent of Problem *P.* 6 can be found recalling the definition of the cost function J given in Equation 4.3, and introducing nonnegative slack variables $\epsilon^y \in \mathbb{R}^{N_y}$, $\epsilon^u_1 \in \mathbb{R}^{N_u}, \ldots, \epsilon^u_m \in \mathbb{R}^{N_u}$:

$$
\begin{aligned}
&\mathbf{1}^T_{N_y}\epsilon^y + \textstyle\sum_{i=1}^m \mathbf{1}^T_{N_u}\epsilon^u_i \leq J^{(k)} \\
&-\mathbf{1}_n\epsilon^y(h) \leq \Psi(A^h\mathbf{x}(k) + \textstyle\sum_{i=1}^m \sum_{j=0}^{h-1} A^{h-1-j}B_i\mathbf{v}_i(j)) \leq \mathbf{1}_n\epsilon^y(h), && h \in [0, N_u[\\
&-\mathbf{1}_n\epsilon^y(h) \leq \Psi(A^h\mathbf{x}(k) + A^{h-N_u}\textstyle\sum_{i=1}^m \sum_{j=0}^{N_u-1} A^{N_u-1-j}B_i\mathbf{v}_i(j)) \leq \mathbf{1}_n\epsilon^y(h), && h \in [N_u, N_y[\\
&-\mathbf{1}_{m_i}\epsilon^u_i(h) \leq \Phi_i\mathbf{v}_i(h) \leq \mathbf{1}_{m_i}\epsilon^u_i(h), && h \in [0, N_u[\\
&H_0\mathbf{x}(k) + \textstyle\sum_{i=1}^m \sum_{j=0}^{N_u} G_{i,\,j}\mathbf{v}_i(h)) \leq \mathbf{f} && \text{(}P\text{. 7)} \\
&A^{N_y}\mathbf{x}(k) + A^{N_y-N_u}\textstyle\sum_{j=0}^{N_u-1} A^{N_u-j-1}(\sum_{i=1}^m B_i\mathbf{v}_i(j)) = \mathbf{0} \\
&||\mathbf{v}_i(h)||\,||\mathbf{v}_j(h)|| = 0, && \forall i \neq j, h \in [0, N_u[\\
&\epsilon^y \geq 0, \epsilon^u_i \geq 0.
\end{aligned}
$$

By the notation $\epsilon^y(h)$, $\epsilon^u_i(h)$ we mean respectively the h-th component of vectors ϵ^y, ϵ^u respectively. Considering, now, the following problem:

$$
\begin{aligned}
&\min \textstyle\sum_{i\neq j, h=0,\ldots,N_u-1} \epsilon^u_i(h)\epsilon^u_j(h) \\
&\mathbf{1}^T_{N_y}\epsilon^y + \textstyle\sum_{i=1}^m \mathbf{1}^T_{N_u}\epsilon^u_i \leq J^{(k)} \\
&-\mathbf{1}_n\epsilon^y(h) \leq \Psi(A^h\mathbf{x}(k) + \textstyle\sum_{i=1}^m \sum_{j=0}^{h-1} A^{h-1-j}B_i\mathbf{v}_i(j)) \leq \mathbf{1}_n\epsilon^y(h), && h \in [0, N_u[\\
&-\mathbf{1}_n\epsilon^y(h) \leq \Psi(A^h\mathbf{x}(k) + A^{h-N_u}\textstyle\sum_{i=1}^m \sum_{j=0}^{N_u-1} A^{N_u-1-j}B_i\mathbf{v}_i(j)) \leq \mathbf{1}_n\epsilon^y(h), && h \in [N_u, N_y[\\
&-\mathbf{1}_{m_i}\epsilon^u_i(h) \leq \Phi_i v_i(h) \leq \mathbf{1}_{m_i}\epsilon^u_i(h), && h \in [0, N_u[\\
&H_0\mathbf{x}(k) + \textstyle\sum_{i=1}^m \sum_{j=0}^{N_u} G_{i,\,j}\mathbf{v}_i(h)) \leq \mathbf{f} && \text{(}P\text{. 8)} \\
&A^{N_y}\mathbf{x}(k) + A^{N_y-N_u}\textstyle\sum_{j=0}^{N_u-1} A^{N_u-j-1}(\sum_{i=1}^m B_i\mathbf{v}_i(j)) = \mathbf{0} \\
&\epsilon^y \geq 0, \epsilon^u_i \geq 0,
\end{aligned}
$$

it is easy to prove the following:

Fact 2 *Problem* P. *7 has a feasible solution if and only if the minimum of Problem* P. *8 is 0.*

Proof:

$\leftarrow$

First, note that the two problems share all constraints except for the exclusivity constraint on the norms. Thereby, it is sufficient to prove that the optimal solution of Problem *P.* 8 is feasible for Problem *P.* 7. If the optimum is of Problem *P.* 8 is 0, being ϵ_i^u non negative, the $\forall i \neq j, \forall h \in [0, N_u[$ it must be $\epsilon_i(h) = 0$ or $\epsilon_j(h) = 0$. Recalling the constraint $-\mathbf{1}_{m_i}\epsilon_i^u(h) \leq \Phi_i v_i(h) \leq \mathbf{1}_{m_i}\epsilon_i^u(h)$ and considering the nonsingularity of Φ_i this leads to: $||\mathbf{v}_i(h)|| = 0$ or $||\mathbf{v}_j(h)|| = 0$ that is exactly the exclusivity constraint of Problem *P.* 7.

$\rightarrow$

Consider a feasible solution of Problem *P.* 7: $\mathbf{v}_1^*(.),\ \mathbf{v}_2^*(.), \ldots,\ \mathbf{v}_m^*(.), \epsilon_1^{u*}, \ldots, \epsilon_m^{u*}$, ϵ^{y*}. Considering all elements h for which $||\mathbf{v}_i^*(h)|| = 0$, we can replace $\epsilon_i^{u*}(h)$ with 0 and still obtain a feasible solution. Observe, now, that the solution thus constructed is feasible for Problem *P.* 8 and that its cost function yields 0 due to the exclusivity constraint of Problem *P.* 7. But 0 is lower bound for Problem *P.* 8 due to the non-negativity of ϵ_i^u and, being attained, it is its minimum. •

It is now convenient to introduce a more compact notation. Decision variables $\epsilon_i^u(h)$ can be grouped into a vector $\mathbf{e}$:

$$\mathbf{e} = [\epsilon_1^u(0), \ldots, \epsilon_m^u(0), \epsilon_1^u(1), \ldots, \epsilon_1^u(N_u - 1), \ldots, \epsilon_m^u(N_y - 1)]^T. \tag{4.5}$$

Introduce vector $\mathbf{w} = \frac{1}{2}U\mathbf{e}$; the U matrix is given by:

$$U = \begin{bmatrix} Z_{N_u,\,1} & \mathbf{0}_{N_u} & \mathbf{0}_{N_u} & \ldots & \mathbf{0}_{N_u} \\ Z_{N_u,\,2} & \mathbf{0}_{N_u} & \mathbf{0}_{N_u} & \ldots & \mathbf{0}_{N_u} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ Z_{N_u,\,N_u} & \mathbf{0}_{N_u} & \mathbf{0}_{N_u} & \ldots & \mathbf{0}_{N_u} \\ \mathbf{0}_{N_u} & Z_{N_u,\,1} & \mathbf{0}_{N_u} & \ldots & \mathbf{0}_{N_u} \\ \mathbf{0}_{N_u} & Z_{N_u,\,2} & \mathbf{0}_{N_u} & \ldots & \mathbf{0}_{N_u} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \mathbf{0}_{N_u} & \mathbf{0}_{N_u} & \mathbf{0}_{N_u} & \ldots & Z_{N_u,\,1} \\ \mathbf{0}_{N_u} & \mathbf{0}_{N_u} & \mathbf{0}_{N_u} & \ldots & Z_{N_u,\,2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \mathbf{0}_{N_u} & \mathbf{0}_{N_u} & \mathbf{0}_{N_u} & \ldots & Z_{N_u,\,N_u} \end{bmatrix}, \tag{4.6}$$

where $Z_{i,\,n}$ denotes a row vector of n elements composed of all ones except for a zero in the i-th position. By construction if vector $\mathbf{e}$ is made of non-negative elements, so is vector $\mathbf{w}$. Furthermore, $\mathbf{v}_i(.)$ can be set equal to the

difference $\mathbf{z}_+ - \mathbf{z}_-$, where vectors $\mathbf{z}_+$ and $\mathbf{z}_-$ are component-wise nonnegative. Moreover, it is possible to introduce a positive vector of slack variables s to transform inequality constraints into equality constraints. Stacking vectors $\mathbf{z}_+$, $\mathbf{z}_-$, s, ϵ^y into a vector $\mathbf{z}$, Problem *P.* 8 is equivalent to the following:

$$\begin{array}{l} \min \mathbf{e}^T \mathbf{w} \\ \Gamma \mathbf{e} + \Delta \mathbf{w} + \Omega \mathbf{z} = \rho + \Theta \mathbf{x}(k), \\ \mathbf{e} \geq \mathbf{0}, \mathbf{w} \geq \mathbf{0}, \mathbf{z} \geq \mathbf{0} \end{array} \qquad (P.\,9)$$

where matrices Γ, Δ, Ω, Θ and vector ρ can be computed from the discussion. We search for a minimizer $(\mathbf{e}^*, \mathbf{w}^*)$ with cost function 0. Such a situation occurs if and only if $(\mathbf{e}^*, \mathbf{w}^*)$ is a feasible solution of the following problem:

$$\begin{array}{l} \mathbf{e}^T \mathbf{w} = 0 \\ \Gamma \mathbf{e} + \Delta \mathbf{w} + \Omega \mathbf{z} = \rho + \Theta \mathbf{x}(k), \\ \mathbf{e} \geq \mathbf{0}, \mathbf{w} \geq \mathbf{0}, \mathbf{z} \geq \mathbf{0}. \end{array} \qquad (P.\,10)$$

Problem *P.* 10 is called in the literature Generalized Linear Complementarity Problem (GLCP). Problem *P.* 9 is called bilinear equivalent (BEP) of the GLCP. The most important feature of the presented formulation is the affine dependence of the right hand side of the equality constraint from $\mathbf{x}(k)$ which makes the problem parametric. For this reason we will call Problem *P.* 10 Parametric Generalized Linear Complementarity Problem (PGLCP).

4.1.3 An alternative approach

In this section, we briefly outline, for comparison purposes, an alternative method, based on the theory of mixed logical dynamical systems (MLD) [9], to model the exclusivity condition. Each actuator i can be associated to a logical variable $\delta_i \in \{0,\ 1\}$. $\delta_i(k)$ takes i if actuator i is selected at time k and 0 otherwise. The system dynamical equations can be written as follows:

$$\mathbf{x}(k+1) = A\mathbf{x}(k) + B_1\delta_i(k)\mathbf{u}_1(k) + \ldots + B_m\delta_m\mathbf{u}_m(k). \qquad (4.7)$$

Problem *P.* 4 can be reformulated as:

$$
\begin{aligned}
&\min_{(\mathbf{v}_1(.),\gamma_1(.)),\ldots,(\mathbf{v}_m(.),\gamma_m(.))} J \\
&\mathbf{y}(h+1) = A\mathbf{y}(h) + \textstyle\sum_{i=1}^{m} B_i\gamma_i(h)\mathbf{v}_i(h), && h \in [0, N_u[\\
&\mathbf{y}(h+1) = A\mathbf{y}(h), && h \in [N_u+1, N_y[\\
&H\mathbf{y}(h) + \textstyle\sum_{i=1}^{m} G_i\mathbf{v}_i(h) \leq \mathbf{g}, && i \in [1,m], h \in [0,\ldots,N_u[\qquad (P.11) \\
&H\mathbf{y}(h) \leq \mathbf{g}, && i \in [1,m], h \in [N_u,\ldots,N_y[\\
&\mathbf{y}(0) = \mathbf{x}(k), \\
&\gamma_i(h) \in \{0,\ 1\},\ \textstyle\sum_{i=1}^{m} \gamma_i(h) = 1, && h \in [0, N_u[
\end{aligned}
$$

Note that we used $\gamma_i(.)$ to denote the virtual sequence of logic commands $\delta_i(.)$ and that the exclusivity condition is simply $\sum_i \gamma_i(h) = 1$. This problem is mixed integer and it is nonlinear. Introducing two constants $M > 0$, and $m < 0$ (both having an enormous absolute value) we can state the following correspondence:

$$
\mathbf{y}(h+1) = A\mathbf{y}(h) + \textstyle\sum_{i=1}^{m} B_i\gamma_i(h)\mathbf{v}_i(h) \leftrightarrow \begin{cases}
\mathbf{y}(h+1) \leq A\mathbf{y}(h) + M\sum_{i=1}^{m}\delta_i(h) \\
\mathbf{y}(h+1) \geq A\mathbf{y}(h) + m\sum_{i=1}^{m}\delta_i(h) \\
\mathbf{y}(h+1) \leq A\mathbf{y}(h) + B_1\mathbf{v}_1(h) + M(1-\delta_1(h)) \\
\mathbf{y}(h+1) \geq A\mathbf{y}(h) + B_1\mathbf{v}_1(h) + m(1-\delta_1(h)) \\
\ldots \\
\mathbf{y}(h+1) \leq A\mathbf{y}(h) + B_m\mathbf{v}_1(h) + M(1-\delta_1(h)) \\
\mathbf{y}(h+1) \geq A\mathbf{y}(h) + B_m\mathbf{v}_1(h) + m(1-\delta_1(h))
\end{cases}
$$

By using this correspondence in *P.*11, we come up with a multi-parametric mixed integer linear program (mp-MILP). Advantages of this approach is the large availability of tools for solving this class of problems. Moreover in [18] a method is shown for performing off-line much of the computation. The method allows to find a polyhedral partition of the state space. Inside each region the combinatorial variables (in our case the schedule of the bus) are fixed while the continuous variable vary with an affine law. The method has recently been applied to control of hybrid systems [9]. An evident drawback with respect to the PGLCP is the spatial complexity: it is necessary to introduce m new variables and each equality constraint on the plant evolution is transformed into a set of $2m+1$ inequalities. On the contrary, with the approach that we are proposing, such constraints can be eliminated altogether by using Equation 4.4. Most importantly, we will show next that very effi-

cient solution methods exist for the PGLCP, though it is generally a NP-Hard problem.

4.2 The Generalized Linear Complementarity Problem (GLCP)

. The GLCP was introduced in the literature of the operations research by Ye [78] and it is equivalent to the following extended linear complementarity problem (XLCP), first introduced by Mangasarian [51]:

$$\begin{aligned} &\mathbf{e}^T\mathbf{w} = 0 \\ &\Gamma\mathbf{e} + \Delta\mathbf{w} \in \mathcal{P}, \\ &\mathbf{e} \geq \mathbf{0}, \mathbf{w} \geq \mathbf{0}, \end{aligned} \tag{P. 12}$$

where $\mathcal{P}$ denotes a generic polyhedral set. Clearly $(\mathbf{e}^*, \mathbf{w}^*)$ is a solution of Problem *P.* 12 if and only if it is a global optimizer of the following bilinear equivalent with cost function 0:

$$\begin{aligned} &\min \mathbf{e}^T\mathbf{w} \\ &\Gamma\mathbf{e} + \Delta\mathbf{w} \in \mathcal{P}, \\ &\mathbf{e} \geq \mathbf{0}, \mathbf{w} \geq \mathbf{0}, \end{aligned} \tag{P. 13}$$

Due to the equivalence of the two problems, results and methods developed for the GLCP can be ported to the XLCP and vice versa. Such problems have been intensely studied producing specific algorithms and a plethora of conditions under which the solutions exhibit interesting mathematical properties (e.g. convexity).

An interior-point solution method is proposed for the GLCP in [78]. Notably, the method produces a solution of the problem in polynomial time if $\Delta^T\Gamma$ is negative semi-definite. In the experimental results presented below we applied the algorithm proposed in [51], which is an adaptation to the XLCP of the method proposed for bilinear programming by Frank and Wolfe in 1956 [22]. We can summarize the algorithm as follows: The idea underlying the above algorithm is very simple: at each iteration the cost function is linearized around the solution found at the previous iteration and the linear cost function thus obtained is used to solve an LP at the next iteration. It is immediate to observe:

1. the sequence $\mathbf{w}^{(k)}$, $\mathbf{e}^{(k)}$ lives in a bounded space (given by the convex hull of the veritces of the feasibility polhyedron);

Start the algorithm finding a feasible solution $\mathbf{w}^{(0)}$, $\mathbf{e}^{(0)}$ for the BP and set $k = 0$; if there are no feasible solutions exit.
1. Compute an optimal solution $\overline{\mathbf{w}}^{(k)}$, $\overline{\mathbf{e}}^{(k)}$ for the following problem:
$$\min \mathbf{w}^T\mathbf{e}^{(k)} + \mathbf{e}^T\mathbf{w}^{(k)} \text{ subject to the same constraints as the BP}$$
2. stop if $(\overline{\mathbf{w}}^{(k)})^T\mathbf{e}^{(k)} + (\overline{\mathbf{e}}^{(k)})^T\mathbf{w}^{(k)} = 2(\mathbf{w}^{(k)})^T(\mathbf{e}^{(k)})$
3. Set $k = k + 1$ and update $\mathbf{w}^{(k)}$ and $\mathbf{e}^{(k)}$ as follows:
$$\begin{bmatrix} \mathbf{w}^{(k+1)} \\ \mathbf{e}^{(k+1)} \end{bmatrix} = (1 - \tau_k) \begin{bmatrix} \mathbf{w}^{(k)} \\ \mathbf{e}^{(k)} \end{bmatrix} + \tau_k \begin{bmatrix} \overline{\mathbf{w}^{(k)}} \\ \overline{\mathbf{e}^{(k)}} \end{bmatrix}$$
where
$$\tau_k = argmin_{\tau\in[0,1]}(\mathbf{w}^k + \tau(\overline{\mathbf{w}}^{(k)} - \mathbf{w}^{(k)})^T(\mathbf{e}^k + \tau(\overline{\mathbf{e}}^{(k)} - \mathbf{e}^{(k)})$$
and go to step 1.

---

2. the optimum values computed at each iteration are a monotone decreasing sequence, which is bounded below by 0; hence, the algorithm converges to a limit.

In [51] it is also proven that if the BP has the property that every one of its KKT points solves the XLCP and if the set of feasible solutions of the XLCP is nonempty, then the algorithm produces a feasible vertex of the XLCP in a finite number of iterations.

4.2.1 A numerical example

In order to show the effectiveness of the approach, we present the result of the application of the algorithm to a simple example. The dynamical and the

input matrices of the system are:

$$A = \begin{bmatrix} 0.5 & 0 & & \\ 0 & 1.1 & 0 & \\ & & 1.1739 & -0.1306 \\ & 0 & 0.7863 & -1.4445 \end{bmatrix} \tag{4.8}$$

$$B_1 = \begin{bmatrix} 1.6873 \\ 0.9528 \\ 0 \\ 0 \end{bmatrix} \tag{4.9}$$

$$B_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \end{bmatrix} \tag{4.10}$$

The safe polyhedral set for the state is described by the following inequalities:

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \leq \begin{bmatrix} 0.6 \\ 0.6 \\ 0.6 \\ 0.32 \\ -0.32 \\ 0.32 \\ -0.32 \end{bmatrix} \tag{4.11}$$

Commands are constrained by:

$$\begin{array}{ccccc} 0.12 & \geq & u_1 & \geq & -0.12 \\ 0.3 & \geq & u_2 & \geq & -0.3 \end{array} \tag{4.12}$$

As far as the weighting matrices for the performance index are concerned, we chose:

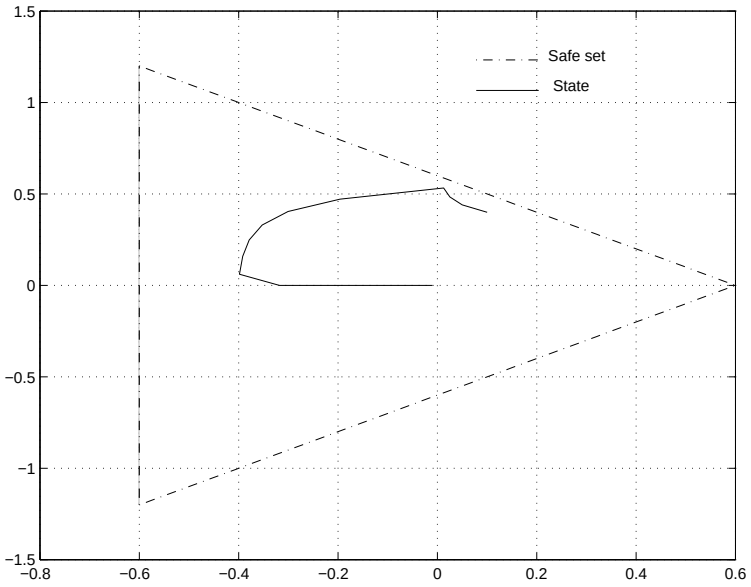
$$\Psi = \begin{bmatrix} 1 * I_{2\times 2} & 0 \\ 0 & 100 * I_{2\times 2} \end{bmatrix} \tag{4.13}$$

$$\Phi_1 = 10 \tag{4.14}$$

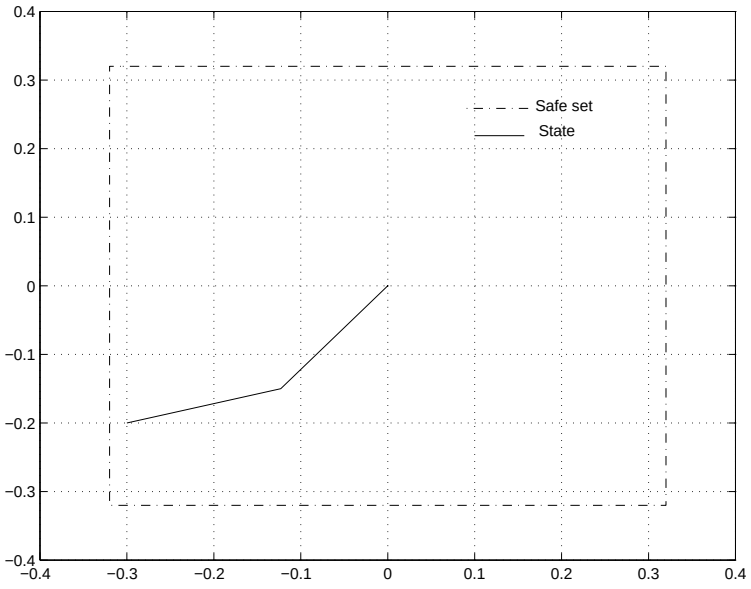
$$\Phi_2 = 1 \tag{4.15}$$

$$\tag{4.16}$$

The chosen command horizon was $N_u = 20$ and the control horizon was $N_y = 25$. As it is possible to see, our example system has a block diagonal



(a)



(b)

Figure 4.2 Evolution of the state variables of the two subsystems: (a) variables x_1, x_2 , (b) variables x_3, x_4

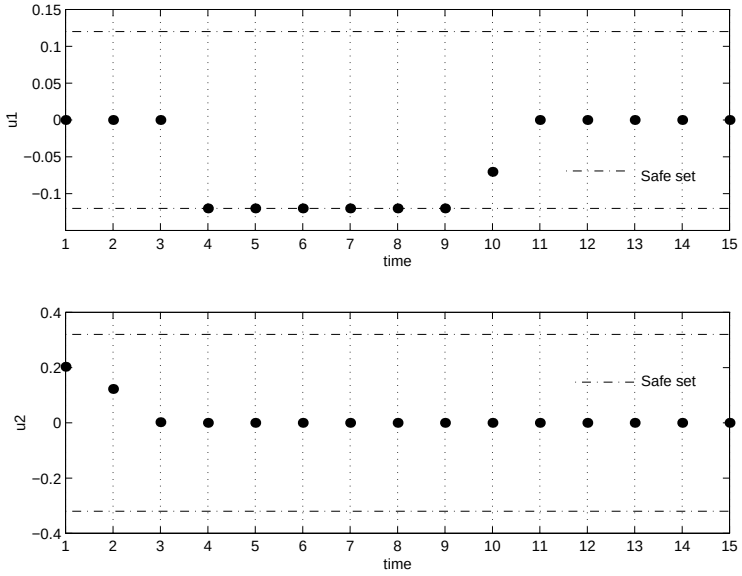


Figure 4.3 Evolution of the command variables of the two subsystems.

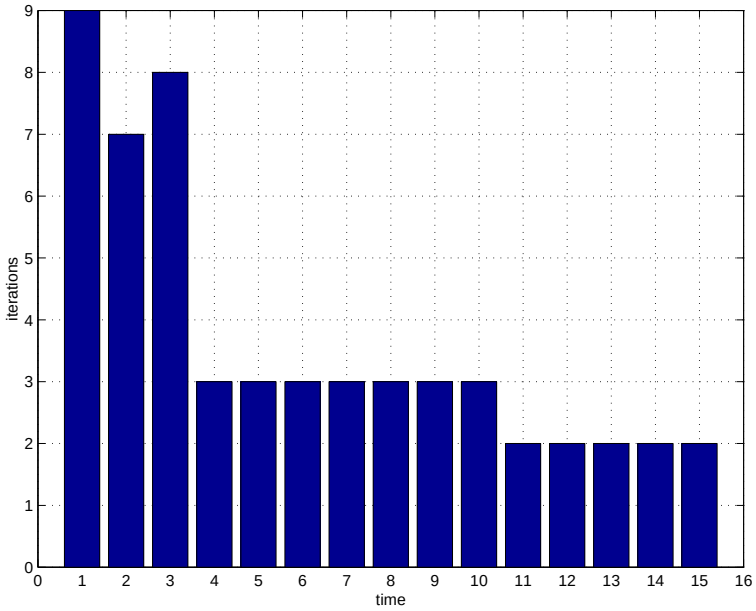


Figure 4.4 Number of iterations required at each step.

structure and it is composed of two independent 2×2 subsystems, which are governed by two distinct scalar command. The only coupling between the two subsystems is, in this simple example, represented by the sharing of a common communication resource. We applied the MPC scheme presented in this chapter, using as optimization method, the Frank and Wolfe algorithm.

The evolution of the state variables of the two subsystems are shown, along with the projection of the safe set, in Figure 4.2. In Figure 4.3 we report the evolution of the computed command variables. As it is possible to see the algorithm implicitly computes an optimal schedule for the bus enforcing that at most one command variable is nonzero at each instant. If Figure 4.4 we report the number of iterations at each step. Each iteration entails the solution of a linear program. As it is possible to see the algorithm complexity is, in this practical application, fairly acceptable.

4.3 Dealing with parametric dependence

Model Predictive Control, as we have shown, naturally leads to parametric optimization problems. Post-optimal and sensitivity analysis [25] permits to study the variations of the solutions with respect to the parameters. This can be used to move much of the computation effort off-line. An example of this procedure applied to a linear MPC optimization problem is in [10]. In [9], as said above, a similar technique is applied to multi-parametric mixed integer linear programs arising from MPC of hybrid systems. Results on the dependence of the optimal solutions from a parameter have been produced in the past for the standard linear complementarity problem [24, 77], but to the authors best knowledge no result is still available for the PGLCP. In this section, we propose some preliminary result which could make inroad to this type of application.

We will refer to the PGLCP:

$$\begin{aligned} &\mathbf{e}^T\mathbf{w} = 0 \\ &\Gamma\mathbf{e} + \Delta\mathbf{w} + \Omega\mathbf{z} = \rho + \Theta\mathbf{x}, \\ &\mathbf{e} \geq \mathbf{0}, \mathbf{w} \geq \mathbf{0}, \mathbf{z} \geq \mathbf{0}, \end{aligned} \tag{P. 14}$$

and to its bilinear equivalent

$$\begin{aligned} &\min \mathbf{e}^T\mathbf{w} \\ &\Gamma\mathbf{e} + \Delta\mathbf{w} + \Omega\mathbf{z} = \rho + \Theta\mathbf{x}, \\ &\mathbf{e} \geq \mathbf{0}, \mathbf{w} \geq \mathbf{0}, \mathbf{z} \geq \mathbf{0}. \end{aligned} \tag{P. 15}$$

where, $\mathbf{e} \in \mathbb{R}^q$, $\mathbf{w} \in \mathbb{R}^q$ and $\mathbf{z} \in \mathbb{R}^p$, $\rho \in \mathbb{R}^l$, $\Theta \in \mathbb{R}^{l \times n}$. $\mathbf{x} \in \mathbb{R}^n$ is a vector of parameters.

It is convenient to introduce some definitions and notations. If $\mathbf{u} \in \mathbb{R}^q$ is a vector and $B \subseteq \{1, \ldots, q\}$ is a set of indices, we denote by $\mathbf{u}_B$ the vector obtained taking the components having index in B from $\mathbf{u}$. By $\overline{B}$ we denote the set $\{1, \ldots, q\} \backslash B$. If A is a matrix we denote by A_B the matrix obtained taking the rows having index in B from A; similarly A^B denotes the matrix obtained taking from A the columns having index in B.

Let $\mathbf{e}^*, \mathbf{w}^*, \mathbf{z}^*$ be an optimal solution of Problem *P.*14 and let $I = \{1, \ldots, q\}$, $J = \{1, \ldots, p\}$. The *basis* associated with $(\mathbf{e}^*, \mathbf{w}^*, \mathbf{z}^*)$ is *defined* by:

- the index set $B \subseteq I$ such that $\mathbf{e}^*_B = 0$ and $\mathbf{e}^*_{\overline{B}} > 0$,
- the index set $H \subseteq I$ such that $\mathbf{w}^*_H = 0$ and $\mathbf{w}^*_{\overline{H}} > 0$,
- the index set $G \subseteq J$ such that $\mathbf{z}^*_G = 0$ and $\mathbf{z}^*_{\overline{G}} > 0$,

Based on the notation introduced above, $\overline{B} = I \backslash B$, $\overline{H} = I \backslash H$ and $\overline{G} = J \backslash G$. In order for $\mathbf{e}^*, \mathbf{w}^*, \mathbf{z}^*$ to be a solution of Problem *P.*14 it must be: $B + H = I$. Now we can give the following:

Definition 2 *The basis set associated to a basis* $(B,\ H,\ G)$ *is a set* $X(B,\ H,\ G)$*, in the parameter space of Problem* P. *14, such that* $\forall \mathbf{x} \in X(B,\ H,\ G)$ *the problem has a solution with basis* $(B,\ H,\ G)$.

Remark 2 *It is useful to think of the meaning of basis set in the domain of the problem being considered. Roughly speaking a basis set is a region for which the stabilizing control law has a fixed schedule of the bus over the control horizon of MPC. From the technological point this can be interpreted as an assignment of the slots for a TDMA scheduler.*

It is easy to show the following:

Lemma 5 *The basis set associated to a basis of Problem* P. *14 is a convex set.*

Proof:
Let $\mathbf{r} = \rho + \Theta \mathbf{x}$ be the right hand side of the equality constraints in Problem *P.* 14. Assume that for $\mathbf{r} = \mathbf{r}^{(1)}$ and $\mathbf{r} = \mathbf{r}^{(2)}$ the problem has solutions with basis $(B,\ H,\ G)$. Let $(\mathbf{e}^{(1)}, \mathbf{w}^{(1)}, \mathbf{z}^{(1)})$ be a solution with basis $(B,\ H,\ G)$

obtained for $\mathbf{r} = \mathbf{r}^{(1)}$, and similarly let $(\mathbf{e}^{(2)}, \mathbf{w}^{(2)}, \mathbf{z}^{(2)})$ be a solution with basis $(B,\ H,\ G)$ obtained for $\mathbf{r} = \mathbf{r}^{(2)}$. It is immediate to verify that:

$$\alpha(\mathbf{e}^{(1)}, \mathbf{w}^{(1)}, \mathbf{z}^{(1)}) + (1-\alpha)(\mathbf{e}^{(2)}, \mathbf{w}^{(2)}, \mathbf{z}^{(2)})$$

with $\alpha \in [0,\ 1]$ is a feasible solution for $\mathbf{r} = (1-\alpha)\mathbf{r}^{(1)} + \alpha\mathbf{r}^{(2)}$ and it has basis $(B,\ H,\ G)$. Hence, the basis set is convex with respect to $\mathbf{r}$. The proof is completed by observing that convexity is preserved by the inverse affine mapping yielding $\mathbf{x}$ from $\mathbf{r}$. •

This lemma is applicable to any solution of the PGLCP. From now on we will concentrate on the points that respect that respect Karush-Kuhn-Tucker condition for the *P.* 15. Clearly we shall make the following assumption:

Assumption 1 *Every solution of Problem* P. *15 respecting Karush-Kuhn-Tucker (KKT) conditions (if any) is a global optimizer of the problem with cost function* 0.

For the sake of commodity, we shall also use the following:

Assumption 2 *For each basis* $(B,\ H,\ G)$ *the matrix:*

$$H = \begin{bmatrix} \Delta^{\overline{H}} & \Gamma^{\overline{B}} & \Gamma^{\overline{B}}\Delta_{\overline{B}} + \Delta^{\overline{H}}\Gamma^{T}_{\overline{H}} & \Omega^{\overline{G}} \\ \mathbf{0} & \mathbf{0} & \Omega^{T}_{\overline{G}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \Delta^{T}_{\overline{H}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \Delta^{T}_{\overline{B}} & \mathbf{0} \end{bmatrix}$$

has full row rank.

Now we are in condition to prove the following:

Theorem 3 *Consider Problem* P. *15 under Assumptions 1 and 2. Let* $R_{(B,H,G)} \subseteq X(B,\ H,\ G)$ *be the set for which Problem* P. *15 has KKT solutions with basis* $(B,\ H,\ G)$. *The following statements hold:*

- $R_{(B,H,G)}$ *is a convex polyhedron;*
- *inside* $R_{(B,H,G)}$ *the KKT solutions* $(\mathbf{e}, \mathbf{w}, \mathbf{z})$ *of Problem* P. *15 are affine functions of parameter* $\mathbf{x}$.

Proof:
Consider a KKT solution $(\mathbf{e}, \mathbf{w}, \mathbf{z})$ of Problem *P.* 15 with basis $(B,\ H,\ G)$. If $(\mathbf{e}, \mathbf{w}, \mathbf{z})$. The primal feasibility conditions can be written as:

$$\begin{gathered}\mathbf{e}_B = \mathbf{0}, \mathbf{w}_H = \mathbf{0}, \mathbf{z}_G = \mathbf{0}, \\ \mathbf{e}_{\overline{B}} > \mathbf{0},\ \mathbf{w}_{\overline{H}} > \mathbf{0},\ \mathbf{z}_{\overline{G}} > \mathbf{0},\ \Gamma^{\overline{B}}\mathbf{e}_{\overline{B}} + \Delta^{\overline{H}}\mathbf{w}_{\overline{H}} + \Omega^{\overline{G}}\mathbf{z}_{\overline{G}} = \rho + \Theta\mathbf{x}.\end{gathered} \tag{4.17}$$

Clearly for $(\mathbf{e}, \mathbf{w}, \mathbf{z})$ to be a solution of Problem *P.* 14 it must be $I = H + B$. Moreover $(\mathbf{e}, \mathbf{w}, \mathbf{z})$ is a KKT solution if and only there exist dual variables $\mu^{(e)} \in \mathbb{R}^q$, $\mu^{(w)} \in \mathbb{R}^q$ and $\lambda \in \mathbb{R}^l$ such that:

$$\mu^{(e)} = \mathbf{w} - \Gamma^T\lambda,\ \mu^{(w)} = \mathbf{e} - \Delta^T\lambda, \tag{4.18}$$

$$\mu^{(e)} \geq 0,\ \mu^{(w)} \geq 0,\ \Omega^T\lambda \leq 0, \tag{4.19}$$

and

$$\mathbf{e}^T\ \mu^{(e)} = 0,\ \mathbf{w}^T\ \mu^{(w)} = 0,\ \overline{\mathbf{z}}^T(\Omega^T\overline{\lambda}) = 0. \tag{4.20}$$

Using (4.17) and (4.18) we can write:

$$\mathbf{w}_{\overline{H}} = \mu^{(e)}_{\overline{H}} + \Gamma^T_{\overline{H}}\lambda,\ \mathbf{e}_{\overline{B}} = \mu^{(w)}_{\overline{B}} + \Delta^T_{\overline{B}}\lambda,\ \mu^{(w)}_B + \Delta^T_B\lambda = 0,\ \mu^{(e)}_H + \Gamma^T_H\lambda = 0. \tag{4.21}$$

Furthermore, (4.17), (4.19) and (4.20) yield $\mu^{(e)}_{\overline{B}} = \mathbf{0}$, $\mu^{(w)}_{\overline{H}} = \mathbf{0}$ and $\Omega^T_{\overline{G}}\lambda = \mathbf{0}$. Define $C = H \cap B$. As $I = B + H$, we have $\overline{H} = B\backslash C$ and $\overline{B} = H\backslash C$. Hence the following correspondence holds:

$$\begin{aligned}\mu^{(w)}_B + \Delta^T_B\lambda = 0 \leftrightarrow \begin{cases}\mu^{(w)}_C + \Delta^T_C\lambda = 0 \\ \Delta^T_{\overline{H}}\lambda = 0\end{cases} \\ \mu^{(e)}_H + \Gamma^T_H\lambda = 0 \leftrightarrow \begin{cases}\mu^{(e)}_C + \Delta^T_C\lambda = 0 \\ \Delta^T_{\overline{B}}\lambda = 0\end{cases}\end{aligned} \tag{4.22}$$

It is now possible to write the following linear system:

$$H\begin{bmatrix}\mu^{(e)}_{\overline{H}} \\ \mu^{(w)}_{\overline{B}} \\ \mu^{(e)}_C \\ \mu^{(w)}_C \\ \lambda \\ \mathbf{z}_{\overline{G}}\end{bmatrix} = \begin{bmatrix}\rho + \Theta\mathbf{x} \\ \mathbf{0}\end{bmatrix} \tag{4.23}$$

where:

$$H = \begin{bmatrix}\Delta^{\overline{H}} & \Gamma^{\overline{B}} & \mathbf{0} & \mathbf{0} & \Gamma^{\overline{B}}\Delta_{\overline{B}} + \Delta^{\overline{H}}\Gamma^T_{\overline{H}} & \Omega^{\overline{G}} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \Omega^T_{\overline{G}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \Delta^T_C & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \Delta^T_C & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \Delta^T_{\overline{H}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \Delta^T_{\overline{B}} & \mathbf{0}\end{bmatrix}$$

Observe that H is a square matrix and that, if Assumption 2 holds, it is invertible. Therefore, it is possible to compute exactly the vector of unknown dual and primal variables:

$$\begin{bmatrix} \mu_{\overline{H}}^{(e)} \\ \mu_{\overline{B}}^{(w)} \\ \mu_{C}^{(e)} \\ \mu_{C}^{(w)} \\ \lambda \\ \mathbf{z}_{\overline{G}} \end{bmatrix} = H^{-1} \begin{bmatrix} \rho + \Theta \mathbf{x} \\ \mathbf{0} \end{bmatrix}. \tag{4.24}$$

Using now (4.18) leads to the conclusion that the solutions $(\mathbf{e}, \mathbf{w}, \mathbf{z})$ change affinely with parameter $\mathbf{x}$. Moreover, using primal feasibility (4.17) and dual feasibility (4.19) it is possible to define a polyhedral region in the space of $\mathbf{x}$ for the basis set $R_{(B,H,G)}$. •

Remark 3 *While assumption 2 is purely technical, and it can probably be released with little effort, assumption 1 is more substantial but there is a good deal of results identifying classes of systems for which it holds. Whether and when such conditions apply to the presented application is the subject for future investigations.*

4.4 Conclusions and future work

In this chapter, we analyzed the problem of synthesizing a closed loop control law for a multi-actuator system, for which commands are issued through a shared communication resource. The use of model predictive control allows to account for several types of constraints, including the mutually exclusive use of a communication resource, which we called exclusivity condition. The exclusivity condition naturally leads us toward the formulation of the problem as a PGLCP. Other modeling technique, such as MLD, can also be applied but exhibit a greater spatial complexity. The PGLCP can be solved online using efficient algorithms developed in the operations research literature. We presented one of these algorithms and shown its application on a nontrivial numerical example. Finally, we showed some properties on the parametric dependence of the solution of the problem from the system state used to initiate the optimization problem.

Our future work will be aimed at investigating the latter results. We want to find efficient partitioning techniques for the state space, based on the

PGLCP formulation, to solve the stabilization problem by a switching set of TDMA scheduled controllers. From the modeling point of view, we plan to investigate extensions to the model presented in this chapter. In particular, we want to deal with the problem of state estimation when sensor measurements are conveyed through a shared resource and, eventually, with the combined estimation/control problem where communication constraints regard both sensor measurements and actuator commands.

II

CAD

5

An object-oriented library for simulating real-time control systems

An eye for an eye makes the whole world blind.
- Mahatma Gandhi

The tool described in this chapter focuses on one of the most familiar problems in real-time control software design, i.e. how the performance of a controller is affected by architectural and implementation choices (e.g. the decomposition of feedback controllers into tasks, the allocation of computation resources to tasks, the scheduling of the shared resources, etc). Realistic and quantitative answers to this question during the early phases of the development are a precious tool for product development.

The concept of performance evaluation for a real-time controller can be developed along different directions. Most of the research in the area of real-time computing has studied the performance of concurrent software systems under the viewpoint of their timing behaviour. Ever since the seminal work of Liu and Layland [47], a fundamental performance metric is considered to be the tasks' schedulability, i.e. the ability for a set of tasks to execute respecting their assigned deadlines. For some classes of real-time applications (qualified as soft real-time), a more useful performance metric is represented by the probability for each task to execute respecting its deadlines [1, 70, 40]. At a higher level of abstraction, the "collective" timing per-

formance of a set of tasks has been evaluated in terms of end-to-end delay, output jitter, and other metrics [27].

The compliance of a controller's timing behaviour with some specified requirements (e.g. schedulability) is not always sufficient to characterize performance at the system level. Classical performance metrics normally used during the control synthesis consider the step response (rise time, overshoot, etc.) or the closed loop transfer function. Quadratic cost functions, or other metrics such as $\mathcal{H}_2/\mathcal{H}_\infty$ norms, are the foundation of popular procedures for analytical control synthesis. However, during the control synthesis phase, effects deriving from the implementation architecture are not usually taken into account. The difficulties in finding tractable analytical models for the stochastic delays deriving from data dependencies and scheduling jitter and the lack of adequate modeling and simulation tools, induce the control designers to synthesize control laws assuming null or fixed delays from the underlying implementation platform. As a consequence, even a software design complying with the deadline constraints can result into a poorly performing system. These problems are detected only during the late phases of the design cycle, and the solution is often sought by cycling through a long series of costly trial-and-error iterations between the different phases of the development cycle.

The tool presented in this chaper, called RTSIM, aims at alleviating these difficulties, permitting us to efficiently deal with different aspects of a controller's synthesis. The main goal of RTSIM is to perform joint simulations of a real-time controller and of the controlled plant, collecting performance measures either on the timing behaviour of the controller or on the quality of the plant dynamics. Specifically, a designer is allowed to specify:

- a set of plants (specified through their differential models) connected to a distributed control system by means of sensors and actuators,
- the functional behaviour of the controller,
- the architectural components of the implementation (real-time tasks, RTOS, shared resources),
- the mapping of functional behaviours onto the architectural components.

By leveraging a complete orthogonalization of the functional and architectural designs, RTSIM enables: 1) an easy comparison of different implementation approaches for the same functionalities, 2) a performance based

tuning of such design parameters as the tasks' activation rates/scheduling priorities. The tool is organized as a collection of C++ libraries that include programming facilities for defining stochastic parameters (e.g. for tasks' execution times, network packets dimensions, etc), for collecting performance statistic and for recording events of interest on execution traces.

A very important feature of the tool is that it encompasses the best known solutions for real-time CPU scheduling (either on single or on multiprocessor boards) and for bounded delay sharing of resources, as predefined library classes. The functional specification of the system is provided by interconnecting a set of reusable components, according to a syntax closely related to well-known dataflow paradigms[1] Another important feature of the tool is the presence of a well defined programming framework guiding users in developing their own functional and architectural components. Once the design of the controller has been settled and properly tuned, its implementation on a real-time operating system is straightforward. The fine grained modeling of such software architectural components as real-time tasks, schedulers, synchronization protocols and so on, enables a very accurate simulation of the system's performance.

As far as the simulation of the plant is concerned, RTSIM exploits the functionality of a powerful mathematical library, called OCTAVE [19], embodying *state of the art* solutions for the integration of differential equations.

5.1 State of the art

The best known tool suite for simulating control systems is MATLAB. The MATLAB/Simulink platform is an excellent choice to model and simulate a plant and a functionally described controller. Moreover, it permits one to automatically generate a prototype on a target real-time operating system (by the use of the *Real-Time Workshop* tool). However, it is not possible to immediately to model generic Hardware/Software architecture and scheduling algorithms. To cope with this shortcoming, a MATLAB tool to simulate a real-time scheduler in a Simulink block is proposed in [20]. This allows, to a given degree, the simulation of timing properties and the assessment of the

[1]The term "dataflow" generally denotes a subclass of Kahn processes [31], introduced by Dennis in 1975 [17]. However, since many software environments claim variants of this model even if their semantics bear little resemblance with that proposed by Davis, throughout this chapter a loose meaning for this term will be used. Therefore, dataflow will intuitively denote a directed sequence of transformations applied on data flowing from inputs to outputs.

performance of real-time controllers against changes in the timing attributes of the tasks. The most important feature of this tool is the good integration with the MATLAB/Simulink environment. On the other hand, the lack of a clear separation between functional and architectural specifications hinders the application of the tool to complex systems having event driven and/or time driven activities.

An interesting product, mainly targeted to the automotive industry, is Ascet-SD, by Etas engineering tools. The tool includes an easy to use graphical interface that permits modeling the functionalities of a controller in a Simulink like environment. The main focus of Ascet-3D is the generation of high quality real-time code for prototyped or production hardware.

In recent years many interesting tools have been proposed for the analysis and simulation of complex real-time systems, networks and kernels. One of the first softwares aimed at simulating real-time scheduling was produced by Audsley et al. [4]. The tool permits modeling a system of real-time periodic and aperiodic tasks through a scripting language.

A well-known commercial product in this class is TimeWiz, by Timesys corp., which is mostly aimed at the analysis of the timing behaviour of a real-time system with respect to schedulability constraints. The toolset is being integrated with a UML design framework which allows one to describe complex systems in a fairly general way. However, the tool does not allow one to perform hybrid simulations of a digital controller along with the continuous dynamics of the controlled plant; thus it is not possible to interactively evaluate the performance of control systems against changes in the task architecture and/or in the scheduling policies.

The idea of separating functional and architectural specification is well supported by the VCC tool, produced by Cadence corp. Functional behaviours can be specified using different syntaxes (including the C/C++ language) and the tool permits one to map a given functionality either on hardware components (e.g. Asic) or on software (e.g. concurrent tasks) in order to pursue different performance/cost tradeoffs. The performance assessment in VCC regards mainly the timing behaviour of the components and the simulation of a continuous time plant is not directly supported.

The GIOTTO programming language [68] has been devised to develop hybrid control applications consisting of periodic tasks. The model of computation is primarily aimed at the design and prototyping of time-predictable control system by the usual paradigm of separating the functional from the timing behaviour (hard schedulability requirements). Time predictability (schedulability) is obtained by restricting the design to a time-triggered ar-

chitecture [34]. A remarkable advantage of this paradigm is the elimination of input and output jitters. However, the introduced delays can be a very pessimistic solution in many cases. Moreover, the time triggered approach does not easily cope with event-driven systems.

An integrated design of real-time control systems encompassing performance and schedulability concerns was first proposed by Seto et al. [58]. In this work an optimization procedure for the activation frequencies of control threads is proposed; the goal is maximizing the controller's performance under schedulability constraints. The paper is inspired to the evaluation approach for embedded controllers suggested by Shin et al. [60]. Other noteworthy results on this problem are presented by Kim et al. [33]; the authors first map the classical control design parameters onto the end-to-end requirements of the controller and then apply the method of period calibration [27] to derive the execution parameters of each thread so that the end-to-end requirements are respected. A tool like RTSIM may be a very useful aid to validate the assumptions and the result of these methods and of any other co-design procedure.

5.2 Design process and modelling primitives

The construction of a simulation model for RTSIM is carried out considering two orthogonal viewpoints: the *functional behaviour* of the controller and the *HW/SW architecture* of its implementation. In Figure 5.1, an overview on a typical design process based on RTSIM is depicted.

The functional design, starting from the mathematical model of the plant and of its interactions with the environment, produces a model of the functional behaviour. The functional behaviour specifies a sequence of operations to be performed on data flowing through the controller. Such operations include the computation of the feedback control law, the extraction of meaningful information from sensors and so on. The functional design also produces a set of timing constraints based on the dynamics of the plant and on the physical limitations of sensors and actuators.

The architectural design can be carried out almost independently. This activity leads to the definition of a model consisting of software tasks, schedulers, network protocols and so on.

The functional design is then mapped onto the architectural design, wrapping up the functional components into corresponding architectural entities having specified requirements in terms of execution time, length of messages

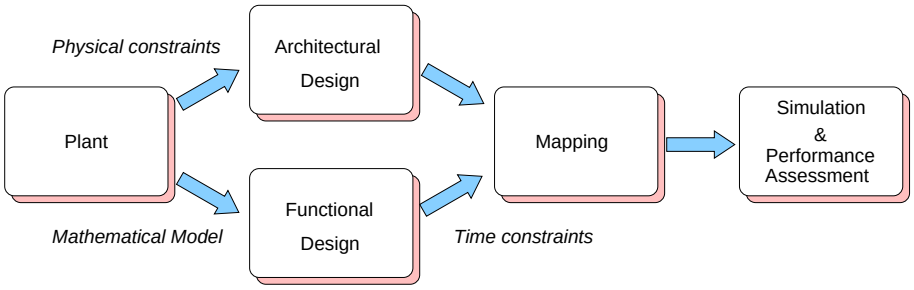


Figure 5.1 Typical design process for the specification and the simulation of a real-time controller.

and so on. In this phase, the timing constraints are translated into real-time constraints on the processes and on the messages on the network.

The separation of the functional and architectural viewpoints permits us to easily test and compare different implementations for the same functional specification in order to identify the solution which best fits the performance/cost tradeoffs of the project.

Finally the system model, composed of its functional and architectural specification, can be simulated obtaining different types of results. A first possibility is to analyze the execution traces (by an appropriate visual tool) to verify if the design meets the desired timing constraints. Moreover, statistics can be collected on the occurrence of events measuring such quantities as the average delay, the jitter and so forth. Most importantly, fundamental information can be derived on the control system's performance by using typical control theoretical metrics (overshoot, rise time, integral cost functions). If the resulting performance is not satisfactory, it is easily possible to return back to any of the previous phases and change the system parameters, the system components (schedulers, communication protocols) and even the entire architecture.

In the rest of this section, the most important modeling primitives of RTSIM for defining both the functional and the architectural specification are introduced. A simple example will show how these primitives are applied to a practical case.

Modeling the functional behaviour

The separation between the functional and architectural specification is aided, in the RTSIM tool, by the use of a dataflow approach for the functional modeling of the system. Dataflow models are a well-suited tool in the design of

real-time software [66, 73] and they are provided, in different flavours, by a variety of tools including Simulink, Ptolemy [42], and GIOTTO [68].

The functional abstractions of RTSIM are essentially of two types: *computing units* and *storage units*. Computing units are used to perform the computation while storage units are used to exchange data between different computing units or between the controller and the external environment.

A **computing unit** is endowed with a set of input ports and output ports which must be connected to storage units. Each computing unit can respond to three different external commands. The first command, called *read* is used to acquire external data from the storage units connected with its input ports. The second one, called *execute*, computes an output value, while the third one, called *write*, is used to write the output into the storage units connected with the output ports. A computing unit can have an internal state (i.e. state remaining between two consecutive invocation). Notice that no particular model is required to specify the *execute* method. Thus, a computing unit can be a finite state machine, a digital filter, a proportional integral derivative (PID) controller, or whatever is needed in the controller's structure. A set of common use computing units such as matrix gains, digital filters, discrete time systems are predefined library objects and can be used in constructing a model of the system without any further programming effort.

Storage units are of three types: *input buffers*, *memory buffers* or *output buffers*. Input buffers serve as an interface between the environment and the controller. From the point of view of the environment they can be thought of as sensors performing a measure on a continuous time quantity. RTSIM offers also the possibility of modeling sensors whose measurement are affected by band-limited white noise. From the controller's side, an input buffer models an I/O card whose content changes when a sampling command is received. Output buffers can be used to model actuators and can only be connected to the output ports of a computing unit. They model digital to analog converters, i.e. when a computing unit writes new data into an output buffer, the value is held up to the next writing. Memory buffers can be accessed either for reading or for writing operations and they realize communications among different computing units.

It is important to observe that when a functional model is constructed no particular assumption is made either on the hardware implementation of a storage unit, or on the way concurrent access requests should be scheduled.

Example. An example of functional design is reported in Figure 5.2. The addressed problem is the control of a simple physical device (an inverted pen-

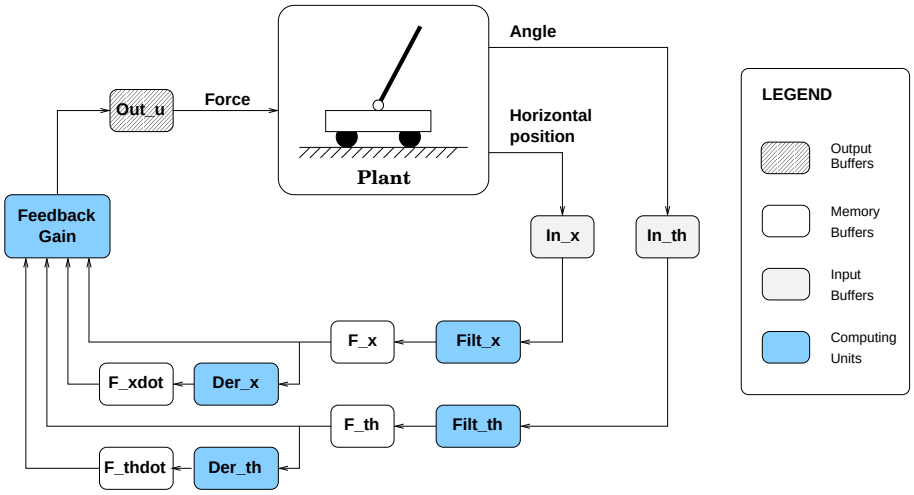


Figure 5.2 Functional design of a simple controller for an inverted pendulum.

dulum). The pendulum is mounted on a cart moving on a one-dimensional track. The horizontal position x and the pendulum angle θ are acquired through a couple of sensors and their values are stored into two input buffers (named `In_x` and `In_th` respectively). Data held in the input buffers are processed by the computing units `Filt_x` and `Filt_th` in order to extract the meaningful information and to filter out the sensor noise: the results are stored into the `F_x` and `F_th` memory buffers. Two digital filters, namely `Der_x` and `Der_th`, are derivative blocks and are used to estimate the linear and angular velocities. Finally the four estimated state variables are used by a computing unit (`FeedbackGain`) to compute the force to be applied to the cart which is stored into an output buffer (`Out_u`). It is worth observing that the computing units shown in this scheme are instances of library predefined objects (four digital filters and a matrix gain).

Modeling the architecture of the system

In our model, a **task** (or process) is a finite or infinite sequence of requests for execution, or *jobs*. Each job executes a piece of code (a sequence of instructions) implementing some functional behaviour. When a job is activated, we say that it *arrives* and the activation time is called *arrival time*. Depending on the pattern of arrival times, tasks can be classified as:

Periodic : if the arrivals are separated by a constant interval of time, called "period";

Sporadic : if the arrivals are separated by variable intervals of time with a lower bound, called *minimum inter-arrival time*;

Aperiodic : if a lower bound is not known on the inter-arrival times.

In real-time systems, tasks have time constraints, often expressed as **deadlines**: for example, a typical time constraint for a periodic task is that each job must finish before the next activation. Another typical constraint is on the completion jitter (the interval of time between two consecutive job completions).

The **instructions** of a task are used to model its timing behaviour. Basically, an instruction is modeled by an execution time (which can be deterministic or stochastic) and can be associated with the *read*, *write* or *execute* command of a computing unit. In this way, one or more computing units can be easily mapped onto a task.

Tasks are assigned to the computational resources (**nodes**) of the system. Each node consists of one or more processors and a real-time operating system (kernel) endowed with a scheduling policy and a synchronization protocol. The state of the art algorithms for CPU scheduling (such as Fixed Priority, Rate Monotonic [47], Earliest Deadline First (EDF) [47], Proportional share [67]) are provided as predefined objects, both for single processor and multi-processor systems. The performance of the schedulers can be enhanced by using aperiodic servers (Polling server [44], Sporadic Server [61], Constant Bandwidth Server [1], etc). Priority inversion in accessing mutually exclusive resources [59] can be avoided by using appropriate synchronization protocols implemented in the tool, such as the Priority Ceiling Protocol [59] or the Stack Resource Policy [6].

Finally, the system can be comprised of several computational nodes connected by **network links**. Tasks on different nodes can communicate by means of *real-time messages*. A communication resource is modeled by a shared physical link, an access protocol and a real-time message scheduler.

Example. A better understanding of what is really meant in RTSIM by "architecture of the system" can be achieved by getting back to the example shown in Figure 5.2.

Suppose, in the case of the inverted pendulum, that the horizontal position is computed from the images grabbed by a camera, whereas a potentiometer is used to acquire the angle. In this case the computation workload necessary to compute x (associated to computing unit `Filt_x`) is much

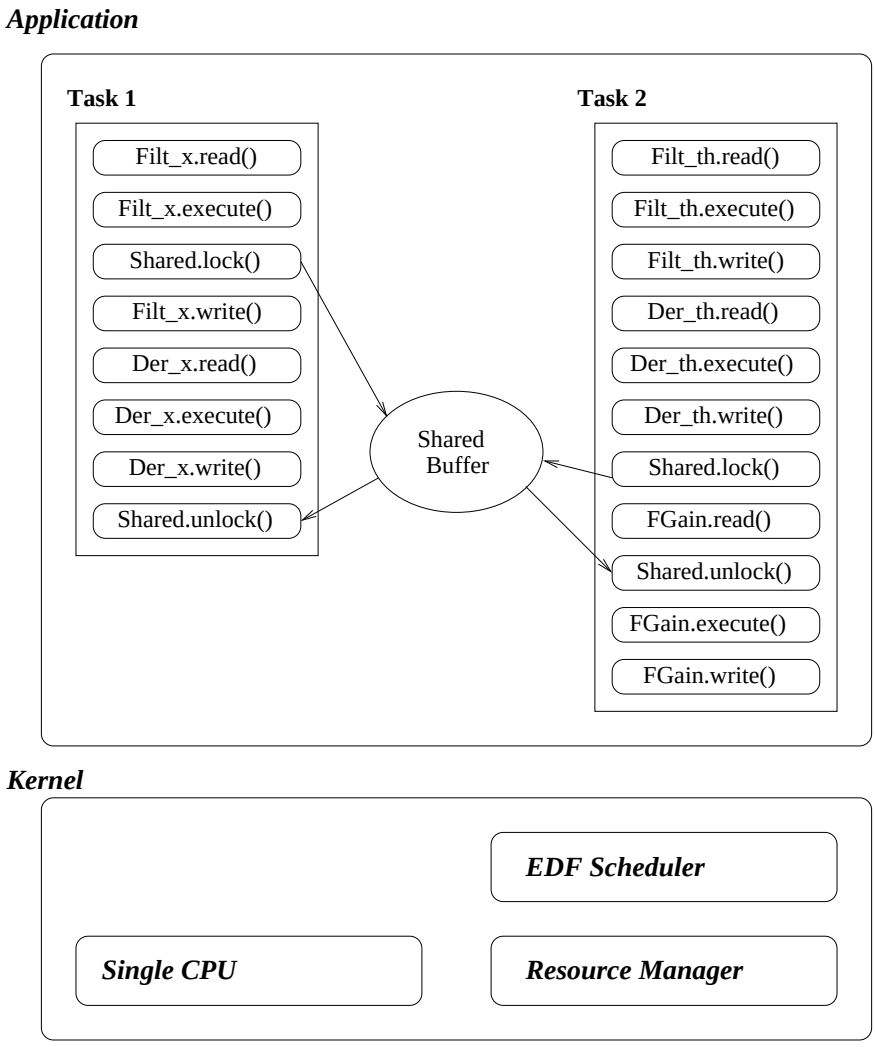


Figure 5.3 Architectural design for the example shown in Figure 5.2. The instructions inside each task are executed sequentially at every activation.

higher than the workload necessary to compute θ (associated to computing unit `Filt_th`). Thus, a possible architecture for the system can be based on two periodic real-time tasks, `Task 1` and `Task 2`. In particular, `Task 1` triggers the actions on computing unit `Filt_x` and `Der_x` in order to compute x and to estimate the $\dot{x}$ horizontal velocity. `Task 2` triggers the same operations on computing units `Filt_th` and `Der_th`.

The main architectural components for this example are depicted in Figure 5.3: each task is represented by a box containing the list of instructions executed every period. The two tasks communicate by means of a shared buffer accessed in mutual exclusion (through the `shared.lock()` and `shared.unlock()` instructions). The concurrent execution of the two tasks is possible using a scheduler component (named `EDFScheduler`) endowed with the Earliest Deadline First scheduling policy [47]. A *resource manager* is used to select the access policy: in this example we use a simple blocking policy. Both the task scheduler and the resource manager are components of a software layer modeling a real-time operating system (`Kernel`).

Of course, this is only one of many possible choices for the hardware/software architecture. This particular choice aims at computational efficiency by concentrating in one task all activities that may be performed at the same rate. A potential drawback of this choice is the lack of modularity. For example, `Task 2` could be replaced by two tasks, the first operating the `F_th` and `F_thdot` computing units, and the second operating the gain unit (`FeedbackGain`). In this way, it could be possible to change "on-line" the way x position is acquired to cope with a potential sensor fault or with a mode change. Another possibility, in case a very high loop rate was needed for stability reasons, is to use two different CPU boards connected by a network link, one performing `Task 1` (which is computationally expensive), and the other one performing `Task 2`. More generally, this simple example shows that the choice of the hardware/software architecture is the solution to a potentially complex problem involving performance issues, cost limitations and physical constraints. This is the reason why decoupling architectural and functional design turns out to be a convenient choice.

Moreover, even with the architecture shown in Figure 5.3, the developer has some degree of freedom in setting the parameters. The choice of the scheduling algorithm, the resource manager and the task activation rates can influence the delay of the two tasks and this in turn impacts upon the stability of the system and the "quality" of the control. For this reason, it is desirable to know in advance which scheduling strategy and which combination of parameters must be assigned in order to maximize the performance of the control strategy.

Assessing performance

Once a system has been modeled, a designer is provided with different opportunities to simulate the system and evaluate the quality of the design. A simulation consists of a sequence of events associated with relevant situations in the architectural model of the system (i.e. task arrivals, task terminations, deadline expirations etc.), which may trigger actions in the functional model. Therefore, events are the fundamental element of any simulation and they can be used in a variety of ways to evaluate the system's performance. With this respect, the first possibility a designer is offered, is to record all events of a simulation, or a meaningful subset, into a trace file. The toolset comprises a utility, called RTTracer, which interprets a trace file and visualizes events in a clear form (see Figure 5.6). In order to facilitate portability RTTracer is entirely written in Java. The application of RTTracer is particularly useful for performing a "temporal" debugging of a complex system when simulations reveal a failure in respecting deadlines for some task or network message. The second important possibility is to define statistical probes, which can be attached to objects to measure the occurrence of events. Statistics can be collected over multiple runs when such parameters as computation times are assigned to vary stochastically according to specified distributions. The main use of this feature is to derive such measures of the system's performance as jitter, latency of data, end-to-end delays on pipelines of tasks and so forth. Finally, particular types of input buffers can be used to measure the evolution of some quantities of interest in the plant (very much like in Simulink). Such units can be connected to files in order to record the time evolution of the observed quantities. In a similar way it is possible to define performance probes which can, for instance, integrate over time the squared norm of the measured quantity.

Example. In order to show some of the possibilities offered by RTSIM, we get back to the example of the inverted pendulum introduced in the previous sections. The code for this example is included in the official distribution of RTSIM (it can be downloaded from the web site `http://rtsim.sssup.it`), where the interested reader can find the exact parameters of the simulation.

The state space of the pendulum is composed of four variables: $[x,\ \dot{x},\ \theta,\ \dot{\theta}]^T$, where x is the linear position, $\dot{x}$ is the linear velocity, θ is the pendulum angle and $\dot{\theta}$ is the angular velocity. In the simulations presented in this section, the pendulum starts from the state $[-0.1,\ 0,\ 0,\ 0]^T$ and has to be stabilized into the origin of the state space $[0\ ,\ 0,\ 0,\ 0]^T$.

The functional and the architectural model of the controller have been

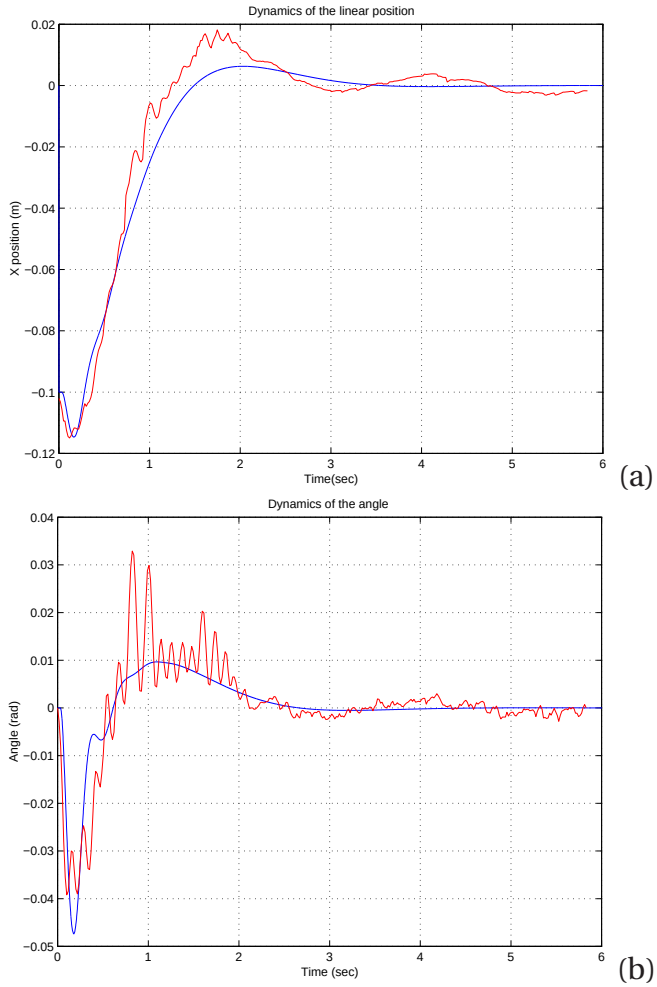


Figure 5.4 Dynamics of the x (a) and θ (b) variables for a simulation run compared with an experimental realization.

introduced above. In order to provide an experimental validation for the use of the tool, we realized a physical implementation of the system based on the SHARK [23] kernel (for details see the Web site `http://shark.sssup.it`). The execution times of the tasks were profiled and imported into the simulation model.

A first element of information on the correctness of the system's behaviour can be obtained by visually inspecting the execution traces of the tasks. In Figure 5.6 the RTTracer output for a simulation is shown. The assumed hard real-time algorithm is the classic Earliest Deadline First. In order for the compliance of the control design with some performance expectation to be verified, it is very important to show the evolution of state variables in time.

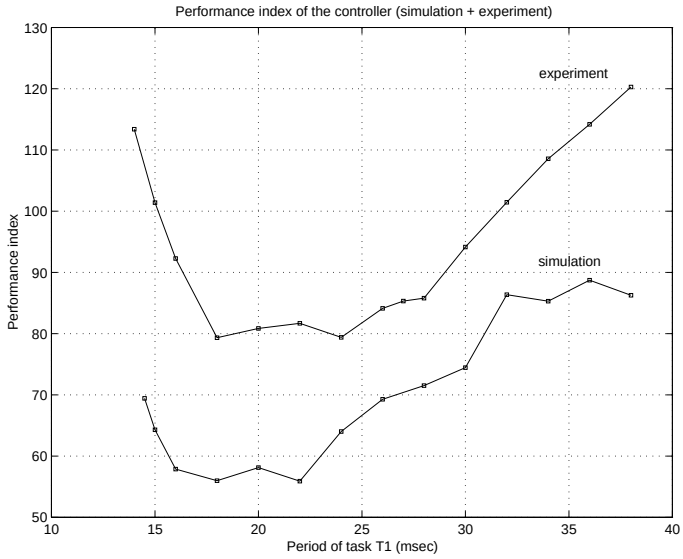


Figure 5.5 Performance index variations with respect to the activation period of `Task 1`.

In Figure 5.4, the dynamics of x and θ obtained from a simulation run are shown. In order to verify the quality of the simulation we report on the same plot also data obtained from an experimental realization. For both simulation and experimental dynamics convergence to zero takes approximatively four seconds.

In order to achieve a quantitative assessment of the influence of the scheduling choices on the control performance, it is necessary to introduce a performance index. A possible choice, as proposed by Shin et al. [60], is the use of a quadratic function:

$$J = E\{\int_0^+ \infty(\mathbf{x}^T Q\mathbf{x} + Ru^2\} \tag{5.1}$$

where:

- E–.″ denotes the expectation value (calculated over stochastically varying parameters),
- $\mathbf{x}$ denotes the state vector,
- u denotes the command variable,
- the Q matrix and the R constant are two weighting factors.

As said above, a particular type of input buffer can be attached to the state and to the input variables in order to compute $\int_0^+ \infty(\mathbf{x}^T Q\mathbf{x} + Ru^2$ as the simulation takes place. The expectation value can easily be approximated by attaching a statistical probe to the storage unit and by collecting the measures over a sufficient number of runs.

The simulations were aimed at evaluating the impact of the task frequencies. The schedulability of tasks for this algorithm is ensured, provided that $\frac{C_1}{T_1} + \frac{C_2}{T_2} \leq U_l$, where T_1 and T_2 are the activation periods of the tasks, C_1, C_2 are the worst case execution times and $U_l = 1$. Residual computation activities (for data logging and man/machine interfaces) where considered by using a lower utilization bound: $U_l = 0.8$.

The simulated and the experimental plots for the performance index are reported in Figure 5.5. In the horizontal axis period T_1 is varied while T_2 is chose accordingly to the relation $\frac{C_1}{T_1} + \frac{C_2}{T_2} = 0.8$. The performance index for each point was evaluated averaging the result of twenty execution and simulation runs. As a remark, the evaluation of each point required approximately forty seconds on a PC with an Athlon 1.2 Ghz processor running the Linux operating system.

As it is possible to see, if high values are chosen for T_1, the system tends to instability and the value of the performance index increases. Similarly, if T_1 becomes too small there is a steep degradation of the performance. The latter phenomenon is due to the corresponding value of T_2, which tends to increase according to the schedulability relation. The best performance is achieved by a trade-off choice for the periods. The behaviour of the cost function is pretty similar in the two plots, except for the higher values of the experimental data. This difference, which is also evident in the plots in Figure 5.4, is due to the adoption of a simplified model for the plant. As a matter of fact, such aspects as the transfer function of the motor, the sensors and process noise and the nonlinearities on the actuators were neglected in the construction of the plant model, since the accuracy level obtained with the simplified model was deemed satisfactory for the purposes of this work.

5.3 Description of the tool

Summarizing the illustration above, RTSIM consists of a collection of C++ libraries containing three types of objects:

- continuous time plants,

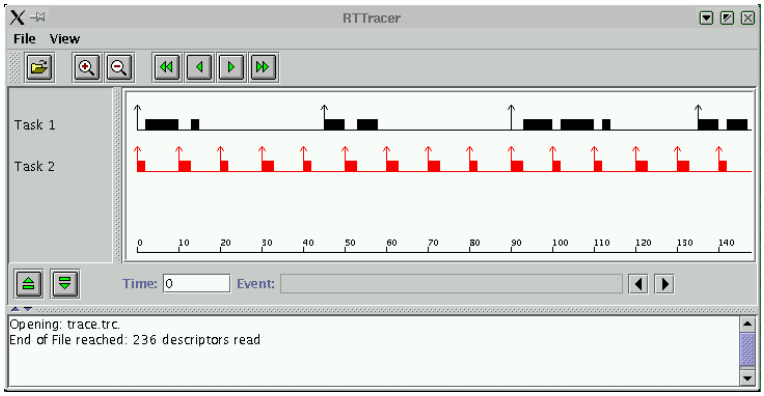


Figure 5.6 Graphical output of a trace of a RTSIM simulation.

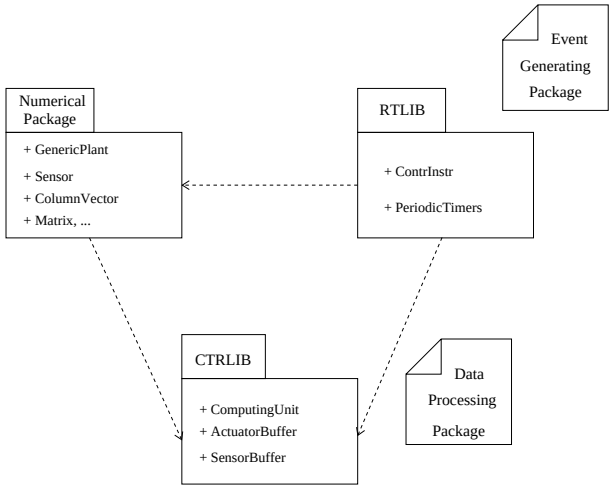


Figure 5.7 Main components involved in a RTSIM based simulation of a real-time controller

- functional components of control software, and
- architectural components of control software.

The distinction of these conceptual domains dictated a decomposition of the software into three interacting packages, as shown in Figure 5.7.

The package denoted as “Numerical Package” is used to model and simulate plants. Objects living in this package evolve in continuous time and they are described by means of differential equations. The package called “CTRLIB” is used to construct the functional model of the system. Objects belonging to this package do not posses an intrinsic concept of time evolution:

their actions are triggered by objects belonging to other packages (in particular to RTLIB). The "RTLIB" package is used to describe the architectural components a functional model is mapped onto. Objects evolve according to a discrete event model of computation [41]: they react to events and are able to generate other events in their turn.

When designing the class hierarchies for the packages, we wanted to achieve a high degree of decoupling so as to facilitate an autonomous evolution of the tool along the three different dimensions. For instance, in our intentions, a developer should be able to extend the library of computing units with new algorithms without caring too much for the structure of kernels or scheduling algorithms and vice versa. In order to achieve this goal, structural relations between components and their interactions had to be captured through a set of clear interfaces. Particularly, for what concerns the interaction between the three packages, we could leverage an important property of the addressed systems: meaningful interactions between plants and controllers take place only on the occurrence of a specific set of events generated by RTLIB. On one hand, in the time interval separating two writings on the output buffer, the differential equations of a plant can be integrated assuming constant values in the actuators[2]. On the other hand the plant state can be observed through the objects simulating the sensors only when an event associated with sampling is generated. Hence, a substantial role in the RTSIM simulation environment is played by the generation of discrete events for RTLIB. This is achieved by using the Metasim library, which is a small software layer developed at the Retis Lab of Scuola Superiore S. Anna. Metasim provides the basic classes for writing generic discrete event simulations [13, 37, 39] and a clear framework to use them.

The remainder of this section is devoted to a short description of the three packages (both structural and behavioural) and of their most important interactions. For obvious space constraints, the description is far from complete. The interested reader is referred to the technical documentation of the tool [45]. The components of the libraries and their behaviour are described by the UML graphical notation [56].

The RTLIB Package

RTLIB is a library designed to simulate the timing behaviour of a real-time software system. It models entities like real-time tasks, scheduling algorithms, single and multi processor nodes, and network links.

[2] More sophisticated actuator schemes such as first order hold or analog loops can easily be modeled in the plant description.

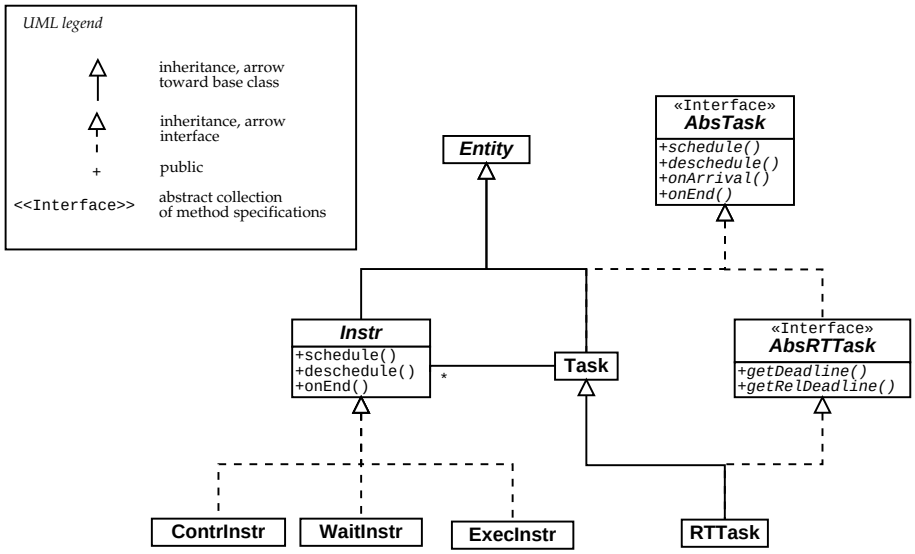


Figure 5.8 Class diagram representing the Task family of classes.

Tasks. One of the most important entities needed to specify a software architecture is the task. The family of classes for modeling tasks is shown in Figure 5.8 as a UML class diagram. In order to de-couple the interface of a task from its internal implementation, we decided to provide an abstract interface `AbsTask` that exposes the basic methods to handle a task (`schedule`, `deschedule`, `onArrival`, `onEnd`). This same interface is used by all entities that can be scheduled: for example, an aperiodic server will implement the `AbsTask` interface (see the server section below).

The `Task` class contains a list of instructions, which are modeled by the `Instr` class. Examples of instructions are:

- `ExecInstr` that models a piece of sequential code with a certain execution time; the execution time is described by a `RandomVar` object: hence it is possible to model a portion of code with an arbitrarily distributed random execution time;
- `WaitInstr` and `SignalInstr` that model the wait and signal system calls for concurrent access to shared resources using semaphores; and
- the `ControlInstr` family of classes that model the execution of computing units.

A programmer inserts instructions into tasks, just as she/he would write a

real implementation. Instructions are executed sequentially [3] and have a duration, which can either be deterministic or specified as a random variable.

In the types of applications we want to model, tasks have timing requirements. The most common constraint is the *deadline*: the *absolute deadline* of a job is the instant of time by which the job must finish; the *relative deadline* of a task is the interval of time between the arrival time and the absolute deadline of each job.

A real-time task is modeled by the abstract interface `AbsRTTask` which derives from the `AbsTask` (Figure 5.8). It comprises the getDeadline() and getRelDeadline() methods, which return respectively the absolute and the relative deadline of a task.

Kernels. The `Kernel` family of classes models a computational resource, like single processor or multi-processor nodes. As in the case of tasks, we found it useful to introduce an abstract interface, `AbsKernel`, capturing the minimum set of services required to any type of kernel. In particular we identified the following services:

- task insertion into a ready queue (method activate),
- task extraction from the ready queue (method suspend),
- task dispatch (method dispatch): the currently executing task is revoked use of the CPU, which is assigned to the first task in the ready queue. In multiprocessor systems the kernel performs this operation on each processor under its control.

The kernel interface also includes methods to handle the most important events a kernel can receive: the arrival of a new task (method onArrival) and the termination of a task's job (method onEnd).

Notice that, at this point, we have not yet introduced any notion of "task priority". In fact, different scheduling policies compare tasks based on different parameters. For example, the Rate Monotonic scheduler requires a static priority to be assigned to each task, whereas the Earliest Deadline First

[3] Thus far, this model has proven sufficiently expressive, since we restricted the application of the tool to modeling classical "data-flow" oriented real-time control applications. In the future, we plan to model also multimodal applications for which a direct support for branches will be necessary. The addition of this feature requires slight modification to the structure of RTLIB and it is planned for future revisions.

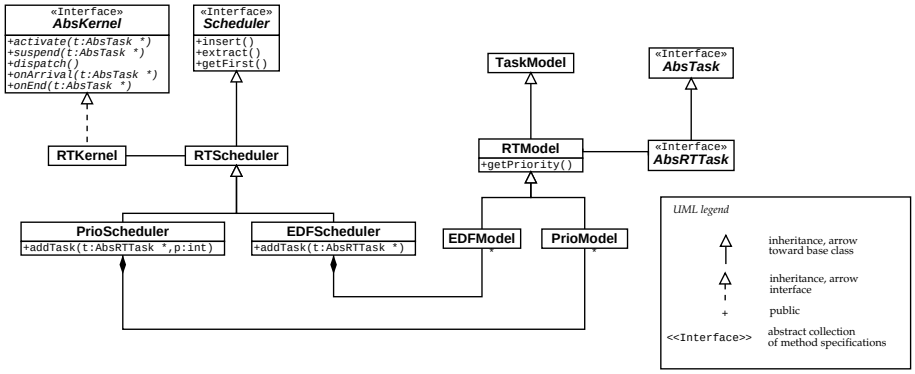


Figure 5.9 Class diagram representing the Kernel family of classes.

scheduler uses the absolute deadline of a job to determine the task priority. Moreover, some scheduling policies (like Proportional Share or Round Robin) do not use any priority at all.

Hence, the ordering of tasks in the ready queue depends on the scheduling policy, which is implemented by the `Scheduler` family of classes. Each one implements a different queuing policy: for example, `EDFScheduler` implements the Earliest Deadline First scheduling algorithm, `PrioScheduler` implements a generic Fixed Priority scheduling algorithm, and so on. The scheduling parameters are not stored in the task class, but in the *wrapper* class `TaskModel`: thus, the task implementation is independent from the scheduling algorithm (as in the *Adapter Pattern* [26]). The `TaskModel` hierarchy of classes is similar to the Scheduler hierarchy: every scheduler corresponds to a task model. In Figure 5.9 the inheritance relationships between these classes are summarized.

The current distribution of RTLIB provides single processor and multiprocessor kernels as predefined components, with any of the following scheduling policies: FIFO, EDF, fixed priority (FP) and rate monotonic, and EEVDF [67]. For the multi-processor versions of EDF and FP, it is possible to allow/disallow migration: in the latter case, tasks must be statically allocated to processors.

Example. The notification mechanism and the way events are handled in RTLIB are better explained with a practical example. The sequence diagram shown in Figure 5.10 captures a snapshot of the system described in Figure 5.3 when a preemption occurs: while `Task 1` is executing, `Task 2` is activated (arrives) and, having a higher priority, preempts `Task 1`.

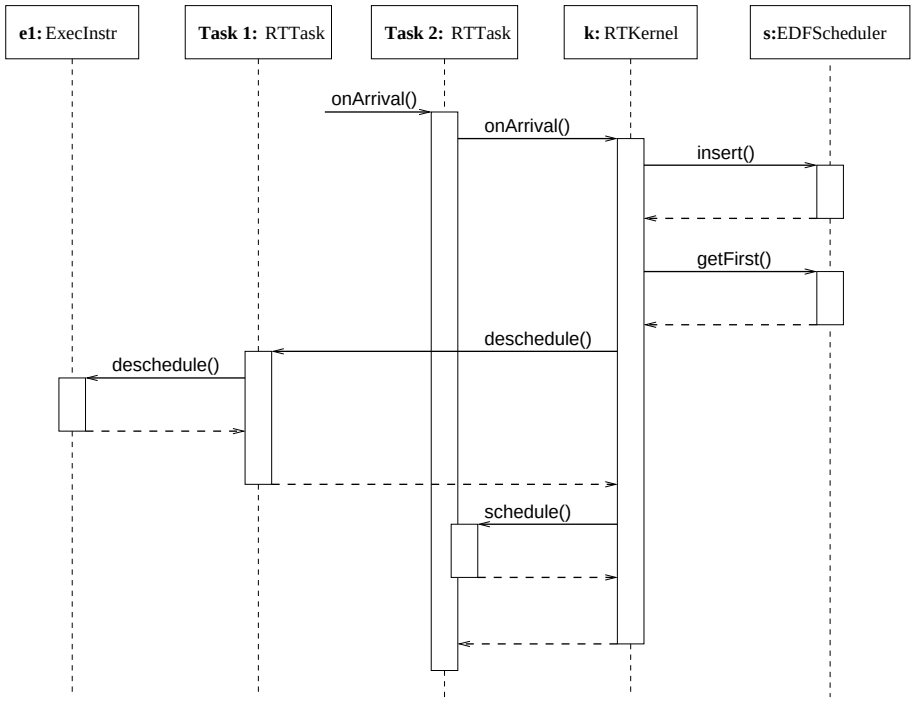


Figure 5.10 Sequence diagram: `Task 2` preempts `Task 1`.

When `Task 2` is activated, its arrival event is processed: as a consequence, the `onArrival()` method of `Task 2` is invoked. After updating its internal status (for example recording the arrival time and resetting the current instruction pointer to the first instruction), `Task 2` calls the `onArrival()` method of the kernel. The kernel, in turn, inserts the task in the ready queue (calling `s.insert()`), and checks if this task is now the first element in the queue. If so, a preemption must occur: the current executing `Task 1` yields the processor and `Task 2` becomes the current executing task.

Hence, `Task 1` must be signaled calling its `deschedule()` method; in turn, it calls the `deschedule()` method of its currently executing instruction. Finally, `Task 2` is signaled calling its `schedule()` method.

Servers. When soft real-time aperiodic tasks are to be scheduled together with hard real-time periodic tasks, the goal is to improve the response time of the aperiodic tasks without compromising the schedulability of the hard real-time tasks. A popular conceptual framework for modeling the behaviour of such systems is to associate a *server* to the soft aperiodic tasks. A server is characterized by certain parameters specifying exactly its performance expectations. Several aperiodic service mechanisms have been proposed un-

der RM [44, 43, 3, 71] and under EDF [62, 28, 64, 63, 1, 46] scheduling.

The `Server` class models these algorithms.

We noticed that in almost all the aperiodic server mechanisms, a server is treated as a particular kind of task and is inserted in the ready queue together with the other regular tasks. For this reason, we decided to derive the server class from the `AbsTask` interface, so that the scheduler does not need to distinguish a regular task from a server. The main advantage is that, when implementing the server algorithm, the scheduler module can be reused without any modification. On the other side, a server handles aperiodic tasks just as a kernel does: when several aperiodic requests are pending, the server must choose which one must be serviced next. For this reason, the server class also derives from the `AbsKernel` interface. In this way, a task has not to distinguish whether it is served by a server or by a regular kernel, and we can re-use the same code for the task class. In the current RTLIB distribution, the polling server, deferrable server (DS), sporadic server (SS), total bandwidth server (TBS), and constant bandwidth server (CBS) are provided as predefined components.

Sharing other resources. Sometimes, tasks access mutually exclusive resources: for example, tasks can access the same memory block that is protected by a mutex semaphore. For example, tasks can access the same memory block that is protected by a mutex semaphore.

In RTSIM, this can be simulated by means of a class `Semaphore` and of a `Resource Manager`, which is the entity that manages the operations on a semaphore, holding the blocked tasks in queues. Tasks can operate on semaphores by means of `WaitInstr` and `SignalInstr` instructions.

In Figure 5.11 the relationship among the classes is shown while in Figure 5.12 we show a possible scenario of execution.

When a task executes a `WaitInstr` instruction, the `Resource Manager` checks if the semaphore is free by invoking `lock(Semaphore *s)`. In the considered scenario, the semaphore is locked, thus the task must be blocked: the resource manager invokes the `Kernel::suspend()` method to block the task and `Kernel::dispatch()` methods, in order to schedule another task.

In the current implementation of RTLIB, a simple locking policy, the Priority Inheritance protocol (PIP), the Priority Ceiling protocol (PCP), and the Stack Resource Policy (SRP) are provided as predefined components. In the case where one of these protocols is used, the corresponding resource manager has to interact with Scheduler component to change the task priority

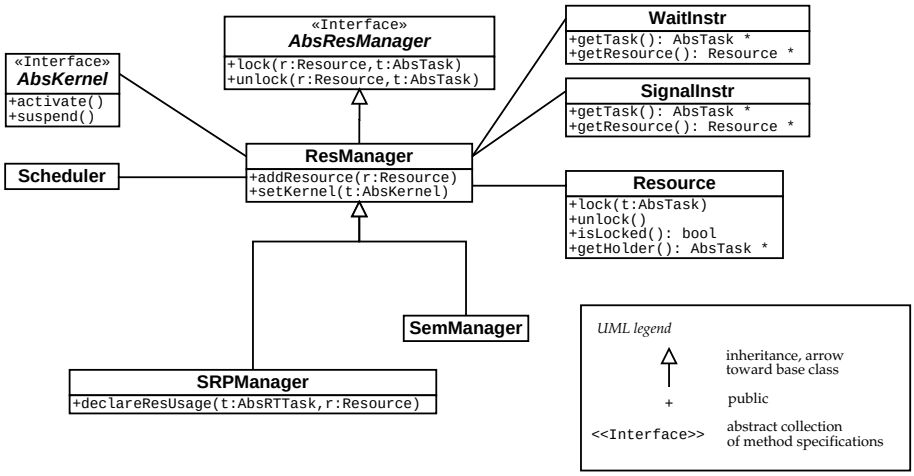


Figure 5.11 Class diagram representing the Resource Manager family of classes.

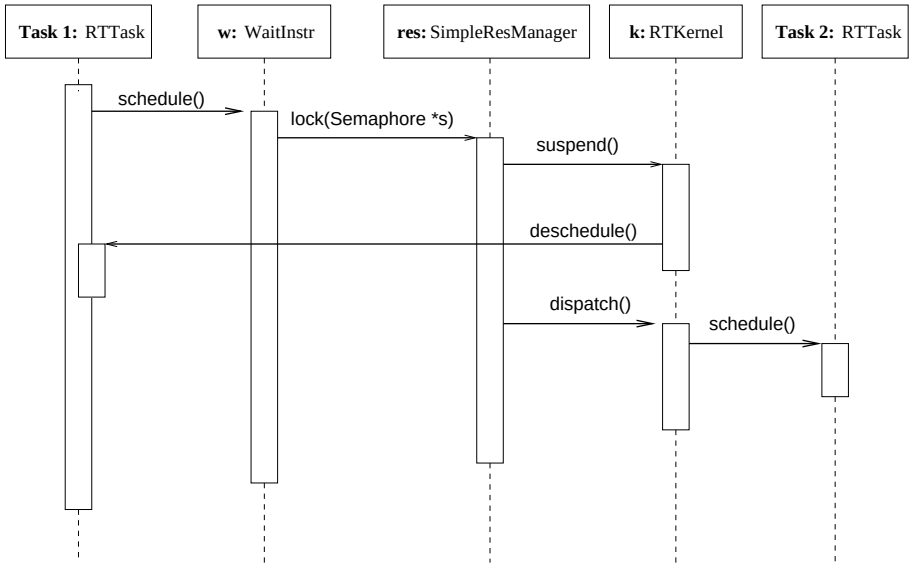


Figure 5.12 Sequence diagram showing a locking operation on a semaphore.

according to the protocol. This justifies the relation between the Resource Manager and the Scheduler component in Figure 5.11.

Networks. Every kernel may have one or more network interfaces, modeled by the `NetInterface` family of classes, each one connected to a network link, modeled by the `NetLink` family of classes. For each network link class, there is a corresponding network interface class.

A task can send a message, modeled by the `Message` class, to another task passing it to the appropriate network interface of its kernel. The `Message` class implements the `AbsTask` interface: in this way, it can be handled by a `Scheduler`. A network interface has a pointer to a `Scheduler` object for implementing the message en-queuing policy. It realizes the medium access protocol, such as the Ethernet or CAN bus protocol. In particular, the `CANInterface` has a pointer to a function that transforms the message priority (or deadline) in a CAN priority[4].

Two additional instructions have been defined:

- `SendInstr` instruction: takes as parameters the name of the destination task and a function object for building new messages.
- `ReceiveInstr` instruction: if a message has already arrived for the task, it gets the message, otherwise it blocks the task waiting for a message from the network interface.

In the current distribution of RTLIB, the Ethernet network and the CAN bus are provided as predefined components.

The Numerical Package

The main purpose of the numerical package is to provide programming models for continuous time plants. A plant is described by means of its state variables, differential equations and so on. From a structural viewpoint, the numerical package is a software layer built on the top of a library which provides some services, such as differential equation integration and linear algebra operations. The current implementation is based on the OCTAVE library, which is a freely available tool encompassing the best known algorithms for numerical computation. The presence of a software abstraction layer allows

[4] High level protocols (like TCP/IP) have not been implemented for they are well beyond the scope of this work.

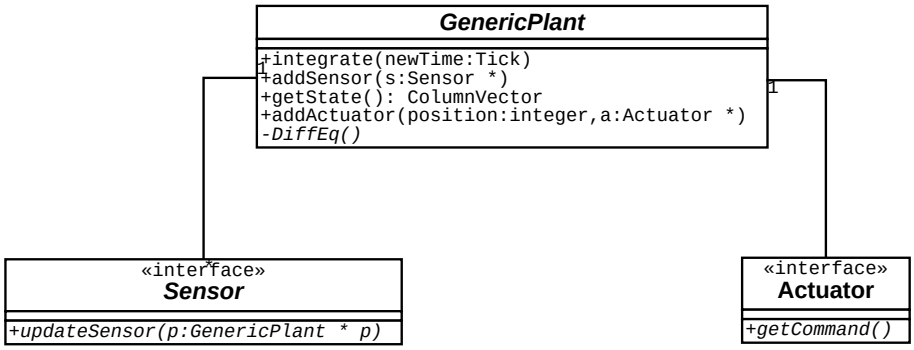


Figure 5.13 Class diagram representing the components of the numerical package to be used for modeling plants.

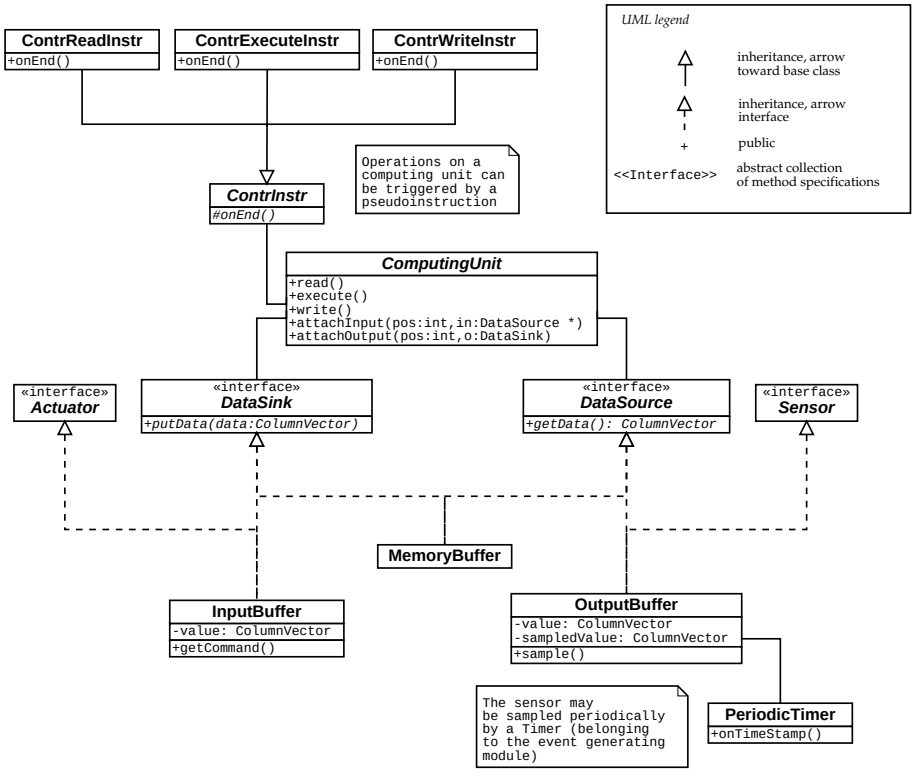


Figure 5.14 The most important classes used to model the functional behaviour of a controller.

us to replace OCTAVE with any other similar solution without affecting the structure of the simulator. As well as permitting the definition of a plant, the numerical package also exports a set of useful classes for linear algebra, such as `Matrix`, `ColumnVector` and so on.

User-defined plants are derived from an abstract class named `GenericPlant` (see Figure 5.13). The inheritance mechanism permits us to add plant specific information by inserting new data members in the derived class. The differential equations are specified by providing a definition to the abstract method `DiffEq`.

The plant evolution can be observed by a set of objects implementing the `Sensor` interface. Formally speaking, if the state of the plant is represented by the column vector $\mathbf{x}$, a `Sensor` realizes an output function $\mathbf{y} = h(\mathbf{x}, t)$. The programmer is required to implement function h by writing a virtual method, called **updateSensor**, which can read the plant state by issuing a call to the `getState` method of the plant. The mechanism used to update the value of the sensor is based on the *observer pattern* [26].

The evolution of a plant can be influenced by a set of actuators. An actuator is an object implementing the `Actuator` interface. Each actuator is registered into a position, denoted by an integer number. This convention is to simplify the writing of differential equations. The integration of the plant differential equations is performed by issuing a call to the `integrate` method exported by the plant.

CTRLIB

The functional model of the system is expressed using the classes of the CTRLIB package. CTRLIB offers two types of components: computing units and storage units. Both of these components are framed within a hierarchy of classes. The structure of the basic classes of CTRLIB is shown in Figure 5.14.

In order to specify a new type of computing unit, the programmer has to derive it from the abstract class `ComputingUnit` and has to provide an implementation for three pure virtual methods: `read()`, `execute()` and `write()`. Once the class is defined, the programmer can instantiate objects from it to be used in different contexts. For example, a class implementing a PID controller is likely to be a reusable component.

A `ComputingUnit` is connected to a set of inputs, which are objects implementing the `DataSource` interface, and to a set of outputs which implement the `DataSink` interface. Each computing unit can be associated with special

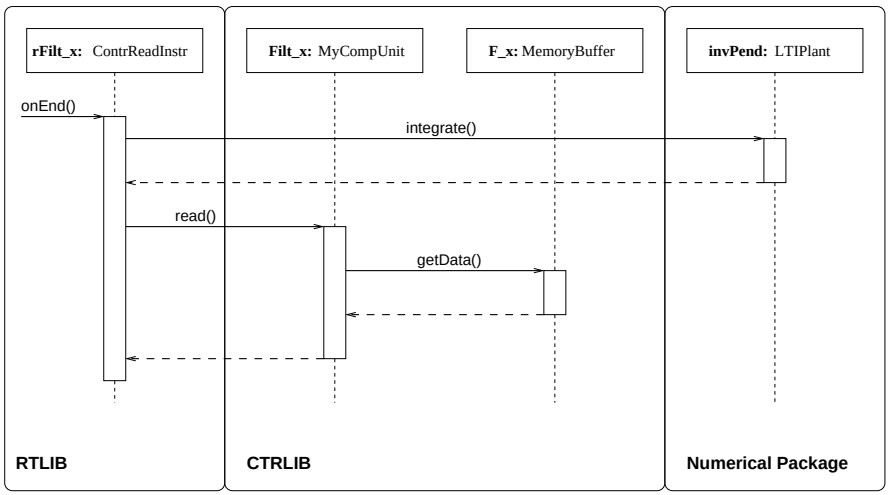


Figure 5.15 Sequence diagram showing the interactions which take place when an end event for a instruction is handled.

instructions triggering the execution of the `read()`, `execute()` and `write()` operation. Such instructions derive from the `ContrInstr` class.

Input buffers are realized as classes implementing both the `Sensor` and `DataSource` interfaces. A predefined method, called `sample()`, is used to sample the value of the sensor upon the occurrence of certain events. A particular choice can be the use of a RTLIB object implementing a periodic timer. Another possibility is to have the `sample()` method called by an instruction of a task. The sampled value can be read by a computing unit calling the `getValue()` method.

Output buffers are objects implementing both the `Actuator` and the `DataSink` interfaces. Thus, they export the `putValue()` method to the computing units and the `getCommand()` method to the plant. Memory buffers implement both the `DataSource` and `DataSink` interfaces and are used to exchange information between the different computing units. Output and memory buffers can be used with no other efforts than defining the width of the data vector when an object is instantiated. In order to simplify the simulation code, the creation of memory buffers connecting different computing units can be made in a semi-automatic fashion by appropriate programming facilities.

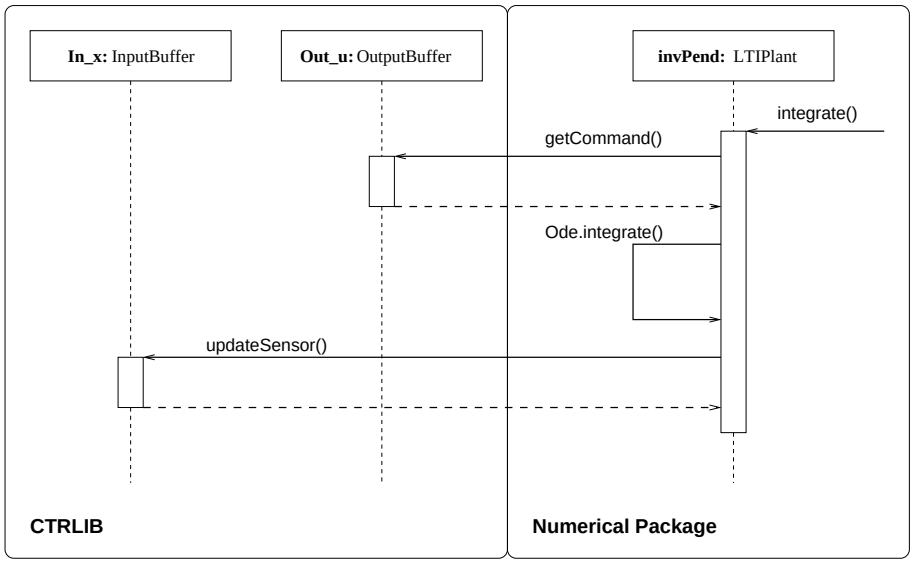


Figure 5.16 Sequence diagram showing how the integration is performed.

Some insight into the hybrid simulation

This section is devoted to showing the main interactions between the different components of the RTSIM tool suite when the libraries are employed to perform a hybrid simulation between a continuous time plant and a digital controller, whose timing evolution is simulated by a RTLIB discrete event model.

In order to highlight the interactions between different components of RTSIM that take place upon the occurrence of some meaningful events, consider the sequence diagram in Figure 5.15. The boxes represent RTSIM objects involved in a simulation. The diagram is partitioned according to the three different packages objects belong to. The diagram shows a sequence of method calls that follows the termination event of the **rFilt˙x** instruction. This event is handled by the `onEnd()` method of the **rFilt˙x** object. The first action performed by **rFilt˙x** calls the `integrate()` method on the **invPendulum** object, which determines the integration of the differential equation up to the current instant of time. The second action is a call on the `read()` method of the computing unit associated with the instruction, which, in its turn, reads the data from the buffer.

It is also interesting to observe how the integration is performed by detailing the sequence of operations performed by calling the `integrate()` method (diagram in Figure 5.16). At the beginning of the integration the

value of the command variables, contained in the output buffer, are acquired through the `getCommand()` method. Then, the integration can be performed (by calling the `Ode.integrate()` function of the OCTAVE library) assuming constant values for the input throughout the integration interval. At the end of the integration, the values contained in the input buffers, which model the sensors, are updated.

5.4 Future extensions

In this chapter a tool for the joint simulation of a plant and of a real-time embedded controller has been presented. By using hybrid techniques the tool supports realistic modeling for many implementation related issues, which are not usually accounted for during controller design. The tool consists of a complete set of C++ libraries for modeling, simulating and gathering statistical profiles of performance metrics. The application of the tool is particularly useful whenever a given control design is based on heterogeneous dataflows from the environment inducing the use of a complex Hardware/Software implementation. In these cases, the tool provides important guidelines in the choice of such parameters as the sampling rates of sensors and, more generally, permits evaluation of different architectural alternatives. In the future the simulation capabilites of the tool will be extended to include the possibility of handling discrete events generated from the plants and mode changes in the controller. The future activities of the RTSIM team will also be aimed at the integration of the tool in more complex design environments, including visual modeling tools and automatic code generation for real-time execution environments.

III

conclusions

6
Summary

"Show me a sane man and I will cure him for you.
- Carl Gustav Jung

This thesis presented a novel methodology for design of embedded controllers (E.C). The problem has been first analyzed from different viewpoints and both conceptual and Computer Aided Design tools have been proposed. The proposed techniques have been successfully applied to meaningful case studies thus demonstrating the effectiveness of the approach.

In Chapter 2 we provided a general outline of the envisioned methodology and of its key cultural references: i.e. control design with information constraints and real-time scheduling. The work unfolded around the key concept of platform: i.e. a set of abstraction modeling relevant aspects of the architecture during the control design.

In Chapter 3 the platform describes a set of time-triggered concurrent tasks sharing a common CPU by means of a RTOS. Each task was assumed to implement a feedback controller on a plant. By using the stability radius as a performance metric, we showed that it is very easy to evaluate if an architecture is adequate for a given performance specification. Moreover, once the architecture has been selected, it is very easy to set up an optimization procedure over maximizing the minimum stability radius attained by the different control loops.

In Chapter 4 the platform based design has been applied to a distributed control problem: a multi-actuator system has to be stabilized but commands to the different actuators are issued through a shared communication link. We discussed possible formulation of the problem and showed the numerical efficiency of a solution based on the Parametric Generalized Linear Complementarity Problem. Exploiting parametric dependence of the solutions,

the state space can be tasseled in different polyhedral regions. In each region the solution to the control design problem is based on a fixed TDMA schedule for the bus.

In Chapter 5 we showed a Cad co-simulation tool that has been constructed to perform *a posteriori* validation and accurate performance assesment before the implementation. The tool separates architectural and functional viewpoint on the design and allows one to explore different architectural mappings of a functional design collecting performance measures expressed in the control application domain. The most important feature of the tool is the fine grained modeling of the best known real-time scheduling algorithms.

Bibliography

[1] L. Abeni and G.Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, december 1998. IEEE.

[2] Luca Abeni. Progetto e realizzazione di meccanismi di sistema per applicazioni real-time multimediali. Master's thesis, Universit degli studi di Pisa, Aprile 1997.

[3] N.C. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(8):284–292, Sep 1993.

[4] N.C. Audsley, A. Burns, M.F. Richardson, K. Tindell, and A.J. Wellings. Stress: A simulator for hard real-time systems. *Software: Practice and Experience*, 6(24), 1994.

[5] T. P. Baker. A stack-based allocation policy for realtime processes. In *IEEE Real-Time Systems Symposium*, december 1990.

[6] T.P. Baker. Stack-based scheduling of real-time processes. *Journal of Real-Time Systems*, 3, 1991.

[7] F. Balarin and other. *Hardaware-Software Co-Design of Embedded Systems: the polic approach*. Kluwer Academic Publishers, 1997.

[8] S.K. Baruah, N.K. Cohen, C.G. Plaxton, and D.A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 6, 1996.

[9] A. Bemporad, F. Borrelli, and M. Morari. Optimal controllers for hybrid systems: Stability and piecewise linear explicit form. In *Proc. 39th IEEE Conf. on Decision and Control*, December 2000.

[10] A. Bemporad, F. Borrelli, and M. Morari. Model predictive control based on linear programming - the explicit solution. *submitted to IEEE Transactions on Automatic Control*, 2001.

[11] A. Bemporad and M. Morari. *Robustness in Identification and Control*, volume 245 of *Lecture Notes in Control and Information Sciences*, chapter Robust Model Predictive Control: A Survey. Springer-Verlag, 1999.

[12] A. Bicchi, A. Marigo, and B. Piccoli. Quantized control systems and discrete nonholonomy. *IEEE Trans. on Automatic Control*, 2001.

[13] G. Booch. *Object oriented design with applications*. Benjamin/Cummings Publishing Company, Inc., 1991.

[14] S. Boyd and C. H. Barratt. *Linear controller design: limits of performance*. Prentice Hall, 1991.

[15] Roger Brockett. Minimum attention control, 1997.

[16] David F. Delchamps. Extracting state information from a quantized output record. *Systems and Control Letters*, 1989.

[17] J.B. Dennis. First version dataflow procedure language. Technical report, Massachusetts In. of Tecnology, Lab. Comp. Sc., 1975.

[18] V. Dua and E.N. Pistikopoulos. An algorithm for the solution of multiparmatric mixed integer linear programming problems. *Annals of operations research*, 99, 2001.

[19] John Eaton et al. http://bevo.che.wisc.edu/octave.

[20] J. Eker and A. Cervin. A matlab toolbox for real-time and control systems co-design. In *Proc. of The Real-Time Computiong Systems and Applications*, Hong Kong, China, December 1999.

[21] N. Elia and S. Mitter. Stabiliztion of linear systems with limited information. *IEEE Trans. on Automatic Control*, 2001.

[22] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.

[23] Paolo Gai, Luca Abeni, Massimiliano Giorgi, and Giorgio Buttazzo. A new kernel approach for modular real-time systems development. In *Proceedings of the 13th IEEE Euromicro Conference on Real-Time Systems*, June 2001.

[24] B. Gailly, M. Installe, and Y. Smeers. A new resolution method for the parametric linear complementarity problem. *European Journal of Operational Research*, 128:639–646, 2001.

[25] T. Gal. *Postoptimal anlyses, parametric programming and related topics.* De Gruyter, Berlin, 1995.

[26] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software.* Addison Wesley, 1997.

[27] R. Gerber, S. Hong, and M. Saksena. Guaranteeing real-time requirements with resource-based calibration of periodic processes. *IEEE Transaction on Software Engineering*, 21(27), 1995.

[28] T.M. Ghazalie and T.P. Baker. Aperiodic servers in a deadline scheduling environment. *Journal of Real-Time System*, 9, 1995.

[29] W.P.M.H. Heemels, B. De Schutter, and A. Bemporad. Equivalence of hybrid dynamical models. *Automatica*, 37:1085–1091, 2001.

[30] Dimitris Hristu and Kristi Moransen. Limited communication control. *System and Control Letters*, 37(4):193–205, July 1999.

[31] G. Kahn. The semantics of a simple language for parallel programming. In *Proceedings of the IFIP Congress 74*, Amstrdam, 1974.

[32] S. Keerthi and E. Gilbert. Optimal infinite-horizon feedback laws for a general class of constrained discrete-time-systems. stability and moving-horizon approximations. *J. Optim. Theory Appl.*, 57(13):265–293, 1988.

[33] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi, and H. Shin. Visual asessment of a real-time system design: a case study on a cnc controller. In *Proceedings of the IEEE Real-time Systems Symposium*, 1996.

[34] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabla, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: The mars approach. *IEEE Micro*, 9(1), February 1989.

[35] H. Kopetz and G. Grnsteidl. Ttp-a protocol for fault-tolerant real-time systems. *IEEE Computer*, January 1994.

[36] Hermann Kopetz. The time-triggered model of computation. *Proceedings of the 19th IEEE Systems Symposium (RTSS98), December 1998*, Dec. 1998.

[37] W. Kreutzer. *Systems Simulation - Programming Styles and Languages*. Addison-Wesley, 1986.

[38] K. Kuetzer, S. Malik, R. Newton, J.M. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Transaction on coputer-aided design of integrated circuits and systems*, 21(27), 2000.

[39] A.M. Law and W.D. Kelton. *Simulation modeling and analysis*. McGraw-Hill Book Company., 1991.

[40] Chen Lee, Raj Rajkumar, John Lehoczky, and Dan Siewiorek. Pratical solutions for qos-based resource allocation. In *IEEE Real Time System Symposium*, Madrid, Spain, December 1998.

[41] E. Lee and A. Sangiovanni-Vincentelli. A unified framework for comparing models of computation. *Transaction on Computer aided Design of Integrated Circuits and Systems*, 17(12):1217–1229, 1998.

[42] Edward A. Lee. Computing for embedded systems. In *IEEE Instrumentation and Measurement Technology Conference*, Budapest, Hungary, May 2001.

[43] J.P. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1992.

[44] J.P. Lehoczky, L. Sha, and J.K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1987.

[45] G. Lipari and L. Palopoli. A framework for simulationg distributed embedded real-time controllers. Technical report, RETIS-LAB, Scuola Superiore S.Anna, 2002.

[46] Giuseppe Lipari and Giorgio Buttazzo. Schedulability analysis of periodic and aperiodic tasks with resource constraints. *Journal of Systems Architecture*, 46:327–338, 2000.

[47] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1), 1973.

[48] C. Van Loan. The sensitivity of the matrix expoential. *Siam Journal of Numerical Analysis*, 1977.

[49] Klein M., Ralya T., Pollak B., and Gonzales Harbour M. *A Practitioner's Handbook for Real-Time Analysis: Guide to rate-monotonic analysis for real-time systems*. Kluwer Academic Publishers, 1993.

[50] Jan Maciejowski. *Predictive Control with Constraints*. Prentice Hall, 2001.

[51] O. L. Mangasarian and J. Pang. The extended linear complementarity problem. *Siam Journal on Matrix Analysis and Applications*, 16:359–368, 1995.

[52] G. N. Nair and R. J. Evans. State estimation under bit rate constraints. In *Proc. of the 37th IEEE Conference on Decision and Control*, Tampa, Florida, December 1998.

[53] M. Di Natale. Scheduling the can bus with earliest deadline techniques. In *Proc. of 21st IEEE Real-Time Systems Symposium*, pages 259 –268, Orlando, FL, 2000.

[54] R. Rajkumar, L. Abeni, D. De Niz, S. Gosh, A. Miyoshi, and S. Saewong. Recent developments with linux/RK. In *Proceedings of the Real Time Linux Workshop*, Orlando, Florida, December 2000.

[55] H. Rehbinder and M. Sanfridson. Scheduling of a limited communication channel for optimal control. In *Proc. of the 39th IEEE Conference on Decision and Control*, Sidney, Australia, December 2000.

[56] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[57] B. De Schutter and B. De Moor. The extended linear complementarity problem and its applications in the analysis and control of discrete event systems and hybrid system. In *Hybrid systems V - Lecture notes on computer science*, pages 70–85, 1999.

[58] D. Seto, J.P. Lehoczky, L. Sha, and K.G. Shin. On task schedulability in real-time control systems. In *IEEE Real Time System Symposium*, December 1996.

[59] Lui Sha, Ragunathan Rajkumar, and john P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE transaction on computers*, 39(9), September 1990.

[60] K.G. Shin, C.M. Krishna, and Y. Lee. A unified method for evaluationg real-time computer controllers and its application. *IEEE Transactions on Automatic Control*, AC30(4):357–366, April 1985.

[61] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Journal of Real-Time Systems*, 1, July 1989.

[62] M. Spuri and G. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1994.

[63] M. Spuri and G.C. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Journal of Real-Time Systems*, 10(2), 1996.

[64] M. Spuri, G.C. Buttazzo, and F. Sensini. Robust aperiodic scheduling under dynamic priority systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1995.

[65] J.A. Stankovic, M. Spuri, K. Ramamritham, and G.C. Buttazzo. *Deadline Scheduling for Ral-Time Systems – EDF and Related Algorithms*. Kluwer Academic Publisher, 1998.

[66] D.B. Stewart, R.A. Volpe, and P.K. Khosla. Design of dynamically reconfigurable real-time software using port-based objects. *IEEE trans. on Software Engineering*, 23(12), 1997.

[67] Ian Stoica, Hussein Abdel-Wahab, Kevin Jeffay, Sanjoy K. Baruah, Johannes E. Gehrke, and C. Greg Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *IEEE Real Time System Symposium*, 1996.

[68] C.M. Kirsch T. Henzinger, B. Horowitzm. Embedded control systems development with giotto. In *Proc. of ACM SIGPLAN 2001 Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES'2001)*, June 2001.

[69] S. Tatikonda, A. Sahaim, and S. Mitter. Control of lqg systems under communication constraints. In *Proc. of the 37th IEEE Conference on Decision and Control*, Tampa, Florida, December 1998.

[70] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W.-S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Real-Time Technology and Applications Symposium*, pages 164–173, Chicago,Illinois, January 1995.

[71] K. Tindell, A. Burns, and A. Wellings. An extendible approach for analysing fixed priority hard real-time tasks. *Journal of Real Time Systems*, 6(2):133–151, Mar 1994.

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

[73] M. Trngren. Fundamentals of implementing real-time control applications in distributed computer systems. *J. of Real-time systems*, 14:219–250, 1998.

[74] A.J. van der Schaft and J.M. Schumacher. Complementarity modeling of hybrid systems. *IEEE Transactions on Automatic Control*, (4), 1998.

[75] W.S. Wong and R. Brockett. Systems with finite bandwidth constraints - part i: State estimation problems. *IEEE Trans. on Automatic Control*, 42(9), 1997.

[76] W.S. Wong and R. Brockett. Systems with finite bandwidth constraints - part ii: Stabilization with limited infromation feedback. *IEEE Trans. on Automatic Control*, 44(5), 1999.

[77] B. Xiao. The linear complementarity problem with a parametric input. *European Journal of Operational Research*, 81, 1995.

[78] Y. Ye. A fully polynomial approximation algorithm for computing a stationary point of the general linear complementarity problem. *Mathematics of Operations Research*, 18:334–345, 1993.

[79] A. Zheng and M. Morari. Stability of model predictive control with mixed constraints. *IEEE Trans. Auto. Cont.*, 40:1818–1823, 1995.