Linköping Studies in Science and Technology

Dissertation No. 1089

Analysis and Optimisation of Distributed Embedded Systems with Heterogeneous Scheduling Policies

by

Traian Pop

Linköping University
INSTITUTE OF TECHNOLOGY

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

Linköping 2007

Analysis and Optimisation of Distributed Embedded Systems with Heterogeneous Scheduling Policies

LINKÖPINGS UNIVERSITET

ISBN 978-91-85715-27-5   ISSN 0345-7524
PRINTED IN LINKÖPING, SWEDEN
BY LIU-TRYCK

To
Ruxandra

Abstract

The growing amount and diversity of functions to be implemented by the current and future embedded applications (like, for example, in automotive electronics) have shown that, in many cases, time-triggered and event-triggered functions have to coexist on the computing nodes and to interact over the communication infrastructure. When time-triggered and event-triggered activities have to share the same processing node, a natural way for the execution support can be provided through a hierarchical scheduler. Similarly, when such heterogeneous applications are mapped over a distributed architecture, the communication infrastructure should allow for message exchange in both time-triggered and event-triggered manner in order to ensure a straightforward interconnection of heterogeneous components.

This thesis studies aspects related to the analysis and design optimisation for safety-critical hard real-time applications running on hierarchically scheduled distributed embedded systems. It first provides the basis for the timing analysis of the activities in such a system, by carefully taking into consideration all the interferences that appear at run-time between the processes executed according to different scheduling policies. Moreover, due to the distributed nature of the architecture, message delays are also taken into consideration during the timing analysis. Once the schedulability analysis has been provided, the entire system can be optimised by adjusting its configuration parameters. In our work, the entire optimisation process is directed by the results from the timing analysis, with the goal that in the end the timing constraints of the application are satisfied.

The analysis and design methodology proposed in the first part of the thesis is applied next on the particular category of distributed systems that use FlexRay as a communication protocol. We start by providing a schedulability analysis for messages transmitted over a FlexRay bus, and then by proposing a bus access optimisation algorithm that aims at improving the timing properties of the entire system.

For all the problems that we investigated, we have carried out extensive experiments in order to measure the efficiency of the proposed solutions. The results have confirmed both the importance of the addressed aspects during system-level design, and the applicability of our techniques for analysing and optimising the studied systems.

Acknowledgements

THE PUBLICATION OF THIS THESIS would have not been possible without the generous support and patient guidance of my advisors: Prof. Petru Eles and Prof. Zebo Peng. I am convinced that nobody else except for them would have done a better job in supervising my PhD studies. After all these years, I am still amazed by Petru's energy and dedication for his work, and I constantly admire his thoroughness when approaching a problem. Also, I have always appreciated Zebo's managing skills, by fully enjoying the freedom he has often allowed us in our work.

The working environment here at IDA[1] is probably one of the best I'll ever experience in my life. To every staff member who contributed in one way or another to the smooth publication process of this thesis, a sincere "thank you".

My ESLAB colleagues, past and present, deserve a special mention for being some of my closest friends during the last years. I would have never enjoyed my PhD studies so much if it weren't for them.

I would also like to thank the members of the IDA[2] group in Braunschweig, Germany, for their hospitality during my visit to their laboratory in November 2004. The close contact with their work has given me a better understanding for some of the topics presented in this thesis.

Naturally, a big influence on the person that I am today is coming from my parents and my sister. I hope that my current achievements will lighten their lives and bring them the feelings of satisfaction that they truly deserve.

In the end, I would like to dedicate this thesis to my wife, Ruxandra, as a small token of appreciation for her patience and for the joy she brings into my life.

Thank you all,
Traian Pop

1. Institutionen för Datavetenskap
2. Institut für Datantechnik und Kommunicationsnetze

Contents

Chapter 1
Introduction

THIS THESIS DEALS with specific issues related to the system-level design of distributed real-time embedded systems implemented with mixed, event-triggered (ET) and time-triggered (TT) task sets that communicate over bus protocols consisting of both static (ST) and dynamic (DYN) phases. We have focused on the scheduling of heterogeneous TT/ET systems and we have studied the factors which influence the efficiency of the scheduling process. We have also identified several optimisation problems specific for this type of heterogeneous systems, and we have approached these problems in the context of design optimisation heuristics.

This chapter starts by presenting the framework of our thesis, namely the area of distributed embedded real-time systems. We make a short introduction to event-triggered and time-triggered execution of tasks, as well as a brief description of static and dynamic transmission of messages. We introduce both homogeneous and heterogeneous TT/ET distributed embedded systems and we focus on the later ones, as they constitute the motivation behind this work.

Analysis and design of distributed embedded systems has been and will be a prolific area of research, considerably boosted by the variety of communication protocols which are involved. This thesis is not the first and definitely not the last contribution in this area. In Section 1.3, the reader is acquainted with other work related to the one presented in our thesis, while in Section 1.4 we outline our contributions to the field of analysis and design of embedded real-time systems.

Finally, Section 1.5 is a feed forward to the following chapters.

1.1 Design Flow of Distributed Embedded Systems

Today, *embedded systems* find their place in more and more applications around us, starting with consumer electronics and appliances and ending with safety critical systems in applications such as aerospace/avionics, railway, automotive industry, medical equipment, etc. Quite often, such systems are also *real-time systems*, as they are constrained to perform certain tasks in a limited amount of time; failure to comply with the timing requirements leads to consequences whose gravity can vary from almost imperceptible loss of quality in an MPEG decoder, up to catastrophic events, like fatal car crashes when braking and air-bag systems fail to react in time. Depending on the nature of the timing constraints, real-time systems can be classified into *soft* real-time systems, in which deadlines can be occasionally missed without the system reaching an intolerable state, and *hard* real-time systems, in which missing a deadline is intolerable because of its possible consequences [Kop97]. This thesis focuses on hard real-time systems.

Designing a hard real-time embedded system requires procedures for guaranteeing that all deadlines will be met. If such guarantees cannot be provided, then the system is considered *unschedulable* and most likely, its implementation will not meet the requirements in terms of timeliness.

The continuous increase in range and number of applications entailing the use of embedded systems [Tur99] is closely followed by an increase in complexity of the applications themselves. Complex environments need more and more complex control embedded systems. The growing complexity of real-time embedded systems is also considerably increased by their heterogeneous nature, which goes along several dimensions, like:

- applications can be data or control intensive;
- the system functionality implies both hard and soft timing requirements;
- the controlled environment can generate discrete or continuous stimuli;
- components inside an embedded computer system can interact among themselves using different synchronisation mechanisms;
- hardware implementations are based on heterogeneous architectures in which one can find application-specific instruction processors (ASIPs), digital signal processors (DSPs), general purpose processors, protocol processors, application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs), etc., all organised in various topologies and interconnected by diverse shared buses, point-to-point links or networks;
- the system includes both analog and digital components.

In this thesis, we have studied another dimension of heterogeneity, resulted from the two different approaches to the design of real-time embedded systems:

- the *time-triggered* approach, in which the processing and communication activities are initiated at predetermined points in time;
- the *event-triggered* approach, in which activities happen when a significant change of state in the system occurs.

As we will see in Chapter 2, the systems which we consider support both time-triggered and event-triggered processing and communication activities.

In Figure 1.1 we present a system-level design flow (adapted from [Ele02]) that starts from a high-level system specification, which may be expressed in several languages, including natural language. The system specification is later refined into an abstract formal model (which can be captured in one or several modelling languages). Starting from the system model, the methodology follows a design exploration stage in which various system architectures are selected, different ways to map the functionality on the available resources are evaluated, and several alternatives for scheduling and synthesis of the communication parameters are examined,

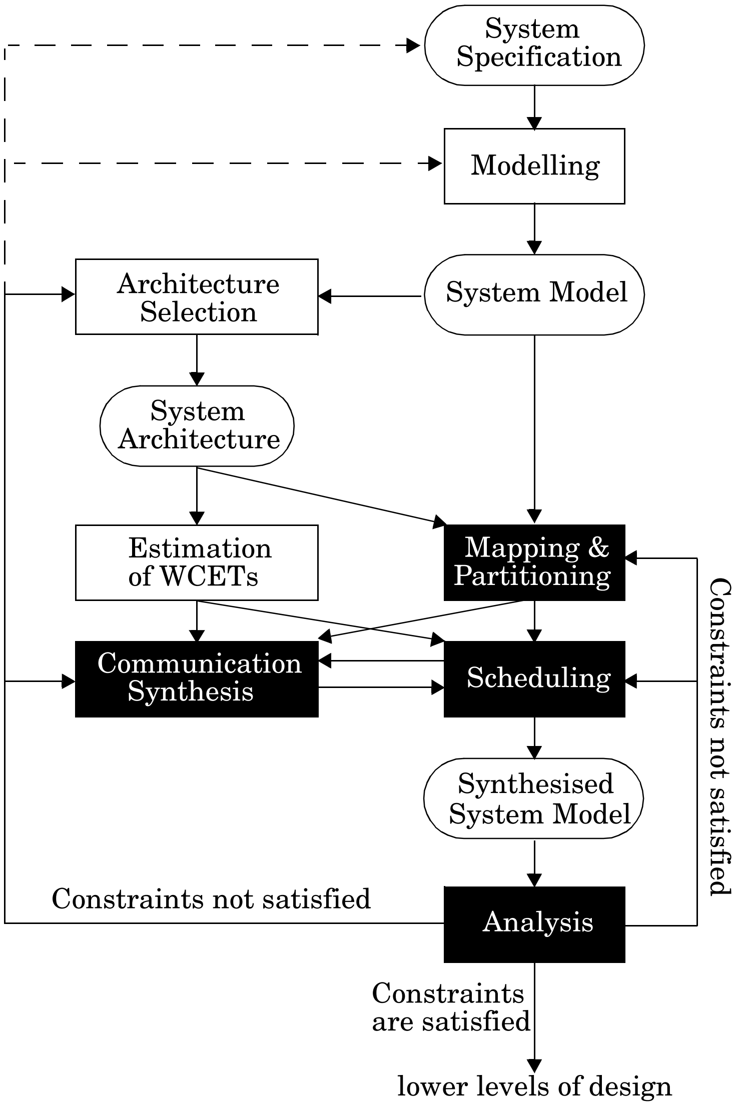


Figure 1.1: System Level Design Flow

so that in the end, the resulted model of the system will meet the requirements imposed for the current design.

In Figure 1.1 we marked with dark rectangles the phases in the design process which are covered in this thesis. First, we developed a method for scheduling and schedulability analysis of the activities in a heterogeneous TT/ET embedded system. This analysis method is then used for guiding the design process, and in particular we concentrated on the problems of *mapping* of functionality, *communication synthesis* and the specific aspect of *partitioning* the functionality into TT and ET activities.

1.2 Heterogeneous ET/TT Systems

In this thesis, we consider heterogeneous embedded systems in the sense that they consist of both time-triggered (TT) and event-triggered (ET) activities. In this section, we present the characteristics of such activities, the typical mechanisms used for implementation and the advantages and disadvantages inherent to each approach.

1.2.1 EVENT/TIME-TRIGGERED TASK EXECUTION

We start by describing first the execution mechanism of tasks in an ET and then in a TT system. In this thesis we consider that the functionality of the system is decomposed into a set of interacting tasks (2.4). A *task* is defined as “a computation that is executed by the CPU in a sequential fashion” [But97].

1.2.1.1 EVENT-TRIGGERED TASKS

In the event-triggered approach, the execution of a task is initiated by the occurrence of a certain event which is related to a change in the system state. For example, in Figure 1.2, task τ_1 is initiated by event E_1 which appears at times t_1 and t_2. If the resources needed by task τ_1 are available at moment t_1 (for example, the CPU is idle), then task τ_1 starts its execution. The mechanism behaves similarly at moment t_2.

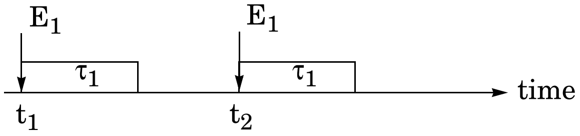


Figure 1.2: ET Task Execution

Usually, the system functionality is composed of several tasks and their execution might lead to resource conflicts, like in the case when two tasks are simultaneously ready for execution and only one of them can make use of the processing capabilities of the system. Typically, such conflicts are solved by assigning priorities to tasks and executing the task with the highest priority. We present below one of the simplest and most common approaches, the fixed priority approach, in which the priorities are statically assigned off-line to tasks and do not change at run time.

In order to implement a fixed priority policy for task execution, a real-time kernel has a component called *scheduler* which has two main responsibilities:

- to maintain/update the prioritised queue of ready tasks;
- to select from the queue and execute the ready task with the highest priority.

The timeline in Figure 1.3 presents how two conflicting ET tasks are executed by such a real-time kernel. In the first case, the kernel implements a preemptive policy for task execution. When task τ_2 is initiated by the occurrence of event E_2, task τ_1 will be interrupted because it has a lower priority than the priority of task τ_2. Task τ_1 is placed in the ready queue and it will resume its execution only after task τ_2 finishes. In the

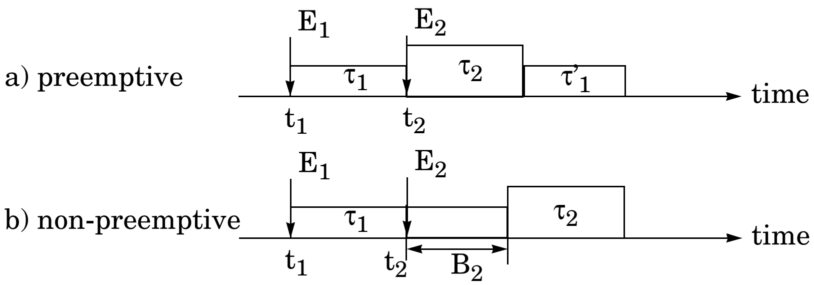


Figure 1.3: Concurrent ET Execution of Tasks

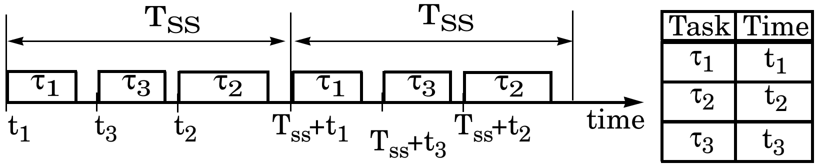
second case, the execution is non-preemptive and task τ_2 has to wait until task τ_1 finishes execution. In this case, even if task τ_2 has a higher priority than task τ_1, it will be blocked for an amount of time B_2 and it will have to stay in the ready queue until a subsequent activation of the scheduler will find the processor available.

The advantages of the event-triggered approach are its flexibility and an efficient usage of the available resources. However, taking into consideration the overheads related to task switching, scheduler activation, etc. considerably increases the difficulty of the schedulability analysis for such types of systems.

1.2.1.2 TIME-TRIGGERED TASKS

In a time-triggered system, the execution of tasks is initiated at pre-determined moments in time. The main component of the real-time kernel is the time interrupt routine and the main control signal is the clock of the system. The information needed for task execution is stored in a data structure called *schedule table*, where each task has a pre-assigned start time. The schedule table is obtained through a static scheduling algorithm, which is executed off-line and which eliminates the possible conflicts between tasks by imposing appropriate start times.

For example, in Figure 1.4, we consider three periodic tasks running on a single processor, each task being executed with period *T*. The schedule table on the right side of the figure shows that the executions of the three tasks τ_1, τ_2 and τ_3 are started at moments t_1, t_2 and t_3. Each start time in the table is computed off-line in such a way that the execution of a task is finished before the next start time stored in the schedule table. After a certain



Task	Time
τ_1	t_1
τ_2	t_2
τ_3	t_3

Figure 1.4: Time Triggered Execution of Tasks

time T_{SS}, called the period of the static cyclic schedule, the kernel performs again the same sequence of decisions.

The period T_{SS} is computed as the least common multiple of the periods of the individual tasks in the system. The case presented above is a very particular one, as all three tasks have the same period T, which gives a perfectly harmonised system, and therefore $T_{SS} = T$. In the general case, one may notice that the size of the schedule table increases substantially if the task periods are not harmonised.

Also, a time-triggered system based on a static schedule table has a low flexibility and is usually inappropriate for dynamic environments for which it provides an inefficient processor utilisation.

However, the time-triggered approach has several important advantages. Being highly predictable, deterministic, easier to be validated and verified, it is particularly suitable for safety-critical applications [Kop97].

1.2.1.3 TASK EXECUTION FOR HETEROGENEOUS TIME/EVENT-TRIGGERED SYSTEMS

When time-triggered and event-triggered activities have to share the same processing node, the operating system for that node has to support concurrent execution of both categories of tasks. A natural way for such an execution support can be provided through a hierarchical scheduler.

The activities on a processing node that uses a hierarchical scheduler are tasks in one of the following categories:

- *schedulers* that implement a scheduling policy each;
- *application tasks* that implement a part of the system functionality.

The execution of each application task is controlled by a scheduler. The execution of each scheduler is controlled by another scheduler, or, in the case of the top scheduler, by the operating system itself. Such an organisation leads to a hierarchy of schedulers, each with their own scheduling pol-

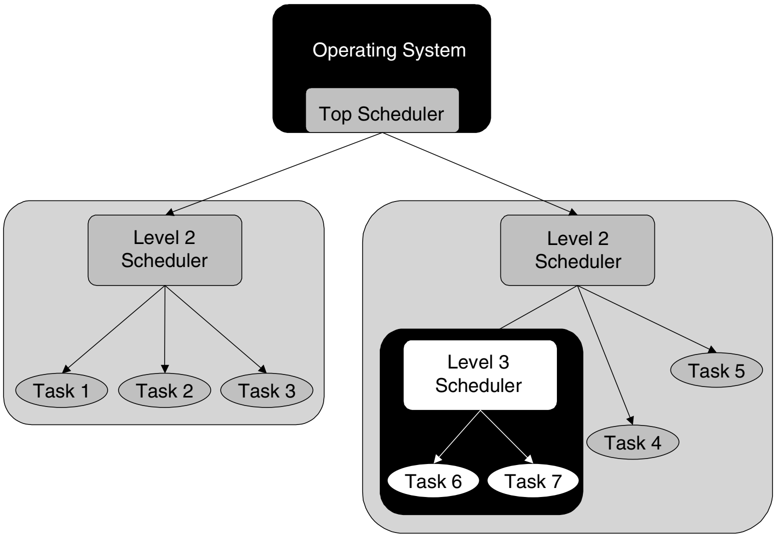


Figure 1.5: Hierarchy of Schedulers

icies and their own set of tasks to control (see Figure 1.5 for an example of such a hierarchy).

1.2.2 STATIC/DYNAMIC COMMUNICATION

The previous section presented activation mechanisms of tasks in a real-time system. We continue with a similar discussion but in the context of communication activities in architectures based on broadcast buses.

There are two main features characteristic to broadcast buses:

- all nodes connected to the communication channel (the bus) receive the same messages; and
- only one node can send messages at a time on the bus. This feature enforces the usage of a bus arbitration method.

In the following sub-sections, we discuss two approaches to communication in distributed real-time systems:

1. Dynamic communication (DYN), in which the communication activities are triggered dynamically, in response to an event.
2. Static communication (ST), in which the communication activities are

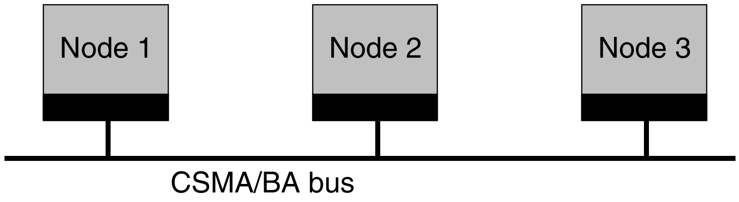
triggered at pre-determined moments in time. For such a case, each node in the system knows (from design time) exactly when and which messages are sent on the bus, as well as how long their transmission takes.

1.2.2.1 DYNAMIC COMMUNICATION

In the case of DYN communication, the trigger which initiates the process of sending a message is the generation of the message itself (by the sending task).

We will give an example of how messages are sent over the CAN bus, which is one of the most used event-triggered communication approaches ([Bos91]). The CAN protocol is based on a CSMA/BA (Carrier Sense Multiple Access with Bitwise Arbitration) arbitration policy, and for this purpose each message in the system has a unique identifier associated to it. Whenever the communication controller in a node receives a message to be sent on the bus, it will have first to wait until the bus is available. When no activity is identified on the bus any more, the message will be sent, preceded by its unique identifier. The identifier of a message acts like a priority, in the sense that if there are several nodes which transmit at the same time on the bus, only the message with the highest priority will go through and the other ones will have to wait the next moment when the bus becomes available. The collisions between messages whose transmission start at the same time are avoided by a non-destructive bitwise arbitration based on the message identifier.

The collision avoidance mechanism implemented with bitwise arbitration is illustrated in Figure 1.6, where three messages m_1, m_2 and m_3 are simultaneously generated on three different nodes. All three messages start being transmitted at the same time. Each message is preceded on the bus by the sequence of several bits representing its priority. The bus is usually hardwired in such a way that it will always have the same value in the case a collision appears. This means that if two nodes transmit two different bits simultaneously, then only the *dominant* bit will be sensed on the bus. The example in Figure 1.6 considers the case where the dominant bit is 1, and as a result, after 3 bits have been sent on the bus, the first node



Identifiers	Status of the Bus (as seen from each node)
m_1: 11010010	1 1 1 <- $Node_1$ stops transmitting
m_2: 11100110	1 1 1 0 1 <- $Node_2$ stops transmitting
m_3: 11101000	1 1 1 0 1 0 0 0 <- $Node_3$ starts transmitting m_3

$\longrightarrow$ time

Figure 1.6: CSMA/BA Bus - Bitwise Arbitration

gives up the transmission, as it sensed a higher priority on the bus than the one sent by itself. The second node gives up after transmitting 5 bits. Having the highest value for the identifier, the message transmitted by the third node will go undeterred on the bus, while messages m_1 and m_2 will be resent only after transmission of m_3 will finish (of course, the bus access mechanism will decide again which of the remaining messages goes first).

1.2.2.2 Static Communication

In Section 1.2.1.2 we presented the time-triggered execution of tasks. Similarly, static (ST) communication activities are initiated at predetermined moments of time. A consistent behaviour of such a distributed multiprocessor time-triggered system requires that the clocks in all the nodes in the system are synchronised to provide a global notion of time [Kop97]. Such a synchronisation can be efficiently achieved through the communication protocol.

In this section, we detail the time-triggered communication mechanism as it appears in the case of a TDMA bus. As we already mentioned, in the case of a TDMA bus the bandwidth is divided into timeslots and each such slot is assigned off-line to a node in the system. During its timeslot, a node has the exclusive right to send messages on the bus. At run-time, if a node

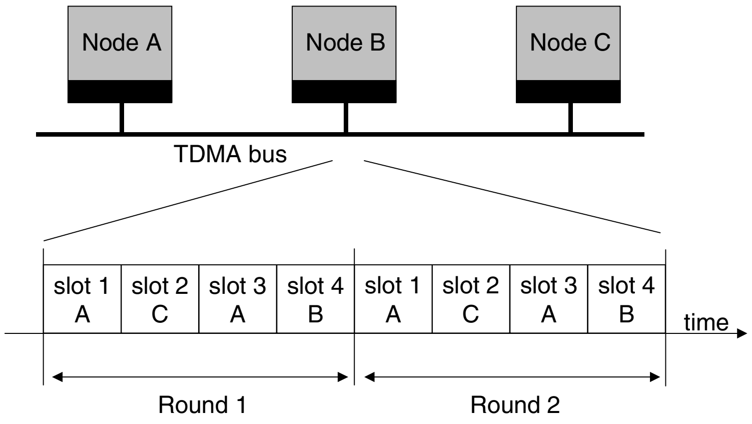
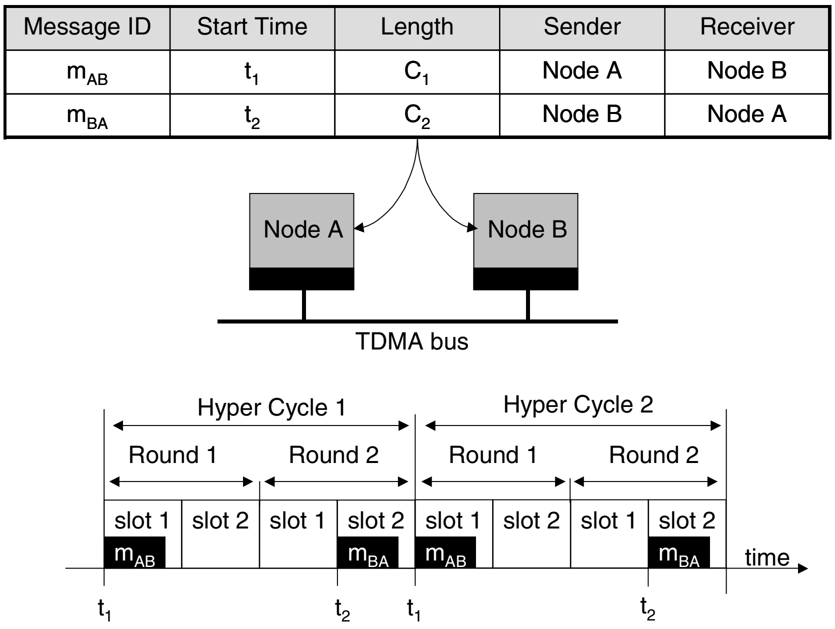


Figure 1.7: TDMA Bus

has a message to send, it will have to wait until the system time has advanced to the start of its pre-assigned slot. The periodic sequence in which the timeslots are ordered represents a TDMA round.

For example, in Figure 1.7, one can see a distributed system with three nodes connected to a TDMA bus. The bus cycle is composed of four time slots, each slot being associated to a node. $Node_A$, for example, can send messages only during $slot_1$ and $slot_3$ of each TDMA round, $Node_B$ can send only during $slot_4$, while $Node_C$ can send only during the second slot of each round. In this way it is guaranteed that only one node transmits on the bus at a time. The TDMA round in the example consists of the sequence of slots 1, 2, 3 and 4.

A typical TDMA based communication protocol is the Time-Triggered Protocol (TTP) [TTP01C]. In the case of TTP, every node stores locally the information related to each of the messages in the system: sender/ receiver, starting time of transmission, message length, etc. A node will send a message on the bus whenever the global current time reaches one of the start time values which are stored locally. For example, in Figure 1.8, $Node_A$ starts sending a message m_{AB} at time t_1 relative to the start of each bus round, during its pre-assigned slot in the first round of the schedule, according to the information stored locally. At the same time, the commu-



Message ID	Start Time	Length	Sender	Receiver
m_{AB}	t_1	C_1	Node A	Node B
m_{BA}	t_2	C_2	Node B	Node A

Figure 1.8: Statically Scheduled TT Communication

nication controller in $Node_B$ will know from its own local table that at time t_1 it will have to start reading message m_{AB}. At time t_2, another message is scheduled to be transmitted on the bus from $Node_B$ towards $Node_A$. The static schedule illustrated in Figure 1.8 expands along two bus cycles, called *rounds*, and the sequence of such two consecutive rounds forms a *hyper cycle*. The static schedule stored locally in each node is repeated periodically with a period equal to the length of such a hyper cycle.

It is largely accepted that the static properties inherent to the TDMA communication considerably diminish the flexibility of the system. Unless bandwidth is reserved from the design time, adding another sending node in the system requires a reconfiguration of the bus round, which usually triggers many other updates and validations of the system design.

However, the determinism associated with the TDMA communication has several major advantages: timing properties of the system are easily

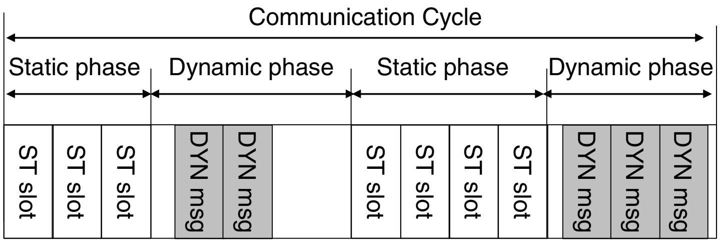


Figure 1.9: Heterogeneous ST/DYN Communication Cycle

guaranteed, system composability is straightforward when extensions are planned, etc.[Kop97].

1.2.2.3 HETEROGENEOUS STATIC/DYNAMIC PROTOCOLS

Nowadays, protocols which support both static and dynamic communication are being developed and placed on the market. Examples in this sense are Flexray [Fuh00], WorldFIP [Wor03] and FTT-CAN [Ple92]. The main motivation behind their appearance was to provide a bus support which combines the advantages of both ST and DYN approaches into powerful and versatile protocols.

In order to avoid the interferences between ST and DYN communication, interference that may have a negative impact on the properties of the ST messages, such a mixed protocol has to enforce a temporal isolation between the two types of traffic. The most common solution is based on the so called *communication cycle* that is split into ST and DYN phases that repeat periodically: TT messages are sent during a ST phase, while ET messages are sent during a DYN phase ([Raj93], [Ple92]).

In Figure 1.9, we present a generalised model of such a protocol, called Universal Communication Model (UCM [Dem01]), in which the communication cycle contains several *static* (ST) and *dynamic* (DYN) phases. A system based on such a protocol will send the ST messages during ST slots according to a pre-defined TDMA scheme and to an associated static schedule, while the DYN messages are packed on-line into frames and sent during the DYN phases according to an arbitration mechanism (like CSMA/BA [Bos91] or mini-slotting [ARI629]).

The Universal Communication Model allows for the modelling and exploration of a large range of mixed ST/DYN communication protocols for bus based systems. This is why in the first part of this thesis, we model the communication on the bus using UCM (Section 2.2).

1.2.3 DISTRIBUTED EMBEDDED SYSTEMS WITH HETEROGENEOUS SCHEDULING POLICIES

There has been a lot of debate in the literature on the suitability of the event-triggered paradigm as opposed to the time-triggered one, for implementation of real-time systems [Aud93], [Kop97], [Xu93]. Several arguments have been brought concerning composability, flexibility, fault tolerance, jitter control or efficiency in processor utilisation. The same discussion has also been extended to the communication infrastructure, which can also be implemented according to the time-triggered or event-triggered paradigm.

An interesting comparison of the TT and ET approaches, from a more industrial, in particular automotive, perspective, can be found in [Loc92]. Their conclusion is that one has to choose the right approach depending on the particularities of the scheduled tasks. This means not only that there is no single "best" approach to be used, but also that, inside a certain application the two approaches can be used together, some tasks being time-triggered and others event-triggered.

The growing amount and diversity of functions to be implemented by the current and future embedded applications (like for example, in automotive electronics [Koo02]) has shown that, in many cases, time-triggered and event-triggered functions have to coexist on the computing nodes and to interact over the communication infrastructure (see for example in Figure 1.10 the illustration of such a heterogeneous functionality mapped over a distributed architecture). When time-triggered and event-triggered activities have to share the same processing node, a natural way for the execution support can be provided through a hierarchical scheduler. Similarly, when such heterogeneous applications are mapped over a multiprocessor architecture, the communication infrastructure should allow for both

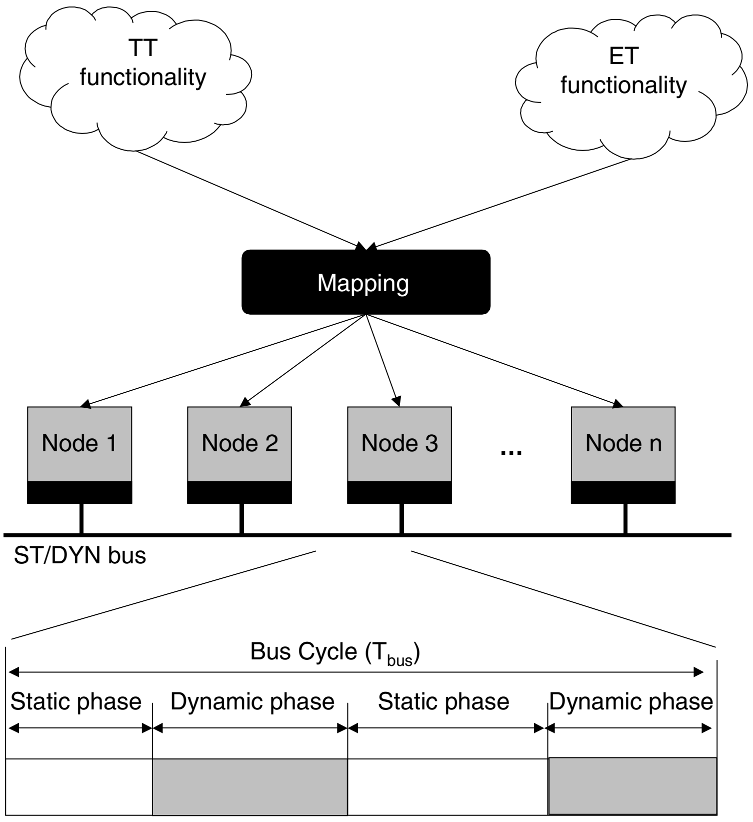


Figure 1.10: Heterogeneous TT/ET Distributed System

static and dynamic message exchange in order to ensure a straightforward interconnection of heterogeneous functional components.

Safety-critical hard real-time distributed applications running on such hierarchically scheduled multiprocessor architectures are difficult to analyse. Due to the hierarchical nature of the schedulers, various execution interferences have to be carefully accounted for during the timing analysis that determines the worst-case response times of the system activities. Moreover, due to the distributed nature of the architecture, message delays have to be taken into consideration during the analysis. Such an analysis is further complicated by the particular characteristics of the communication protocol that mixes both static and dynamic transmission of messages.

In order to cope with the complexity of designing such heterogeneous embedded systems, only an adequate design environment can effectively support decisions leading in an acceptable time to cost-efficient, reliable and high performance solutions. Developing flexible and powerful tools for the design and analysis of such kind of heterogeneous systems represents the motivation behind the work presented in this thesis.

1.3 Related Work

This section presents an overview of the previous research in the area of analysis and system level design for distributed embedded systems. We concentrate in particular on scheduling and communication synthesis, with focus on the time-triggered and event-triggered aspects.

1.3.1 SYSTEM LEVEL DESIGN

System level design methodology is continuously evolving [Mar00], from ad-hoc approaches based on human designer's experience, to hardware/ software codesign, and currently to platform-based design [Keu00] and function-architecture codesign [Bal97], [Dav99], [Lav99], [Tab00].

The design flow presented in Figure 1.1 illustrates only some of the main problems that appear during the system level phases of design. For a deeper insight into system level design aspects with focus on hardware/ software trade-offs, the reader is referred to the surveys in [Wol94], [Mic97], [Ern98], [San03], [Wol03], [Mar03] and [Wol06].

System modelling has received a lot of attention, as powerful computational models and expressive specification languages are needed in order to capture heterogeneous system requirements and properties at different levels of abstraction [Edw97], [Edw00], [Lav99], [Mul04]. Typical hardware architectures for embedded systems have evolved from simple ones (involving only one processor and one ASIC), to distributed and heterogeneous ones (involving multiprocessor architectures distributed over a large area or integrated on a single chip [Ben02]). Such an evolution has directly increased the complexity of the problems related to architecture selection, mapping, partitioning and scheduling of functionality and has led to the apparition of new approaches like those proposed in [Bec98], [Bli98], [Dav99], [Lee99], [Wol97], [Yen97], [Hu03], [Jer04], [Nur04], [Thi04] and [Ben05].

1.3.2 SCHEDULING AND SCHEDULABILITY ANALYSIS OF REAL-TIME SYSTEMS

Task scheduling and schedulability analysis have been intensively studied for the past decades. The complexity of the scheduling problems have been analysed in [Ull75], [Sta94]. The reader is referred to [Aud95], [Bal98] and [But05] for surveys on this topic.

A comparison of the two main approaches for scheduling hard real-time systems (i.e., *static cyclic scheduling* and *fixed priority scheduling*) can be found in [Loc92] and [Lön99].

The *static cyclic (non-preemptive) scheduling* approach has been long considered as the only way to solve a certain class of problems [Xu93]. This was one of the main reasons why it received considerable attention. Solutions for generating static schedules are often based on *list scheduling* in which the order of selection for tasks plays the most important role [Cof72], [Jor97] (see also 3.5). However, list scheduling is not the only alternative, and branch-and-bound algorithms [Jon97], [Abd99], mixed integer linear programming [Pra92], constraint logic programming [Kuc97], [Eke00], or evolutionary [Sch94] approaches have also been proposed.

For event-triggered tasks, in this thesis we are interested both in static and dynamic priority based scheduling policies. In our work we will focus our attention on *fixed priority scheduling* (FPS) and *earliest-deadline-first scheduling* (EDF). For both policies, determining whether a set of tasks is schedulable involves two aspects:

1. *The assignment of priorities* to system activities, i.e. what priority should be associated with each task and message in the system so that the task set is schedulable.
2. *The schedulability test*, which determines whether all activities in the

system will meet their deadlines under the current policy.

In the case of EDF scheduling, the priorities are assigned dynamically, at run-time, according to the criticality of each ready task, i.e. tasks that are closer to their deadline will receive higher priorities.

In the case of *fixed priority scheduling*, the priorities are associated to tasks off-line, before the system is deployed. In order to solve the problem of assigning priorities to system activities so that the system is schedulable, two main policies have been developed; they both work under restricted assumptions, i.e. the task set to be scheduled is composed of periodic and independent tasks mapped on a single processor:

a. rate-monotonic (RM) [Liu73] which assigns higher priorities to tasks with shorter periods; it works under the constraint that task deadlines are identical with task periods.

b. deadline-monotonic (DM) [Leu82] which assigns higher priorities to tasks with shorter relative deadlines; this policy assumes that task deadlines are shorter than task periods.

Under a particular set of restrictions regarding the system specification, such policies are optimal. However, if, for example, tasks are not independent, then the optimality does not hold any more for RM and DM policies. Therefore, in [Aud93], the authors proposed a priority assignment in the case of tasks with arbitrary release times. Their algorithm is of polynomial complexity in the number of tasks. However, for the case of multi-processor/distributed hard real-time systems, obtaining an optimal solution for priority assignment is often infeasible, due to complexity reasons. A solution based on simulated annealing has been proposed in [Tin92], where the authors present an algorithm that simultaneously maps the tasks on processors and assigns priorities to system activities so that the resulted system is schedulable. In order to avoid the large amount of computation time required by such a general-purpose approach, an optimised priority assignment heuristic called HOPA has been suggested in [Gut95], where the authors iteratively compute deadlines for individual tasks and messages in the system, while relying on the DM policy to assign priorities to the tasks. Their algorithm has shown a better efficiency than the one proposed in [Tin92], both in quality and especially in speed, making it appropriate for being used inside a design optimisation loop that requires many iterations. As an example, HOPA has been adapted for the design optimisation of multi-cluster distributed embedded systems [PopP03b].

As mentioned above, another main issue in the context of fixed priority scheduling is that of the schedulability tests. In this regard, there are two main approaches used:

a. *utilisation based tests*, in which the schedulability criterion is represented by inequations involving processor utilisation and utilisation bounds. However, such approaches are valid only under restricted assumptions [Liu73], [Bin01], [Leu82], [And01].

b. response time analysis, in which determining whether the system is schedulable or not requires first the computation of the worst-case response time of the tasks and/or messages. The worst case response time of an activity is represented by the longest possible time interval between the instant when that activity is initiated in the system and the moment when the same activity is finished. If the worst case response time resulted for each task/message is lower or equal than the associated deadline for that activity, then the system is schedulable.

Response time analysis is usually more complex but also more powerful than the utilisation based tests. The main reason for this is because response time analysis can take into consideration more factors that influence the timing properties of tasks and messages in a system.

The response time analysis in [Leh89] offers a necessary and sufficient condition for scheduling tasks running on a mono-processor system, under fixed priority scheduling and restricted assumptions (independent periodic tasks with deadlines equal with periods). In order to increase the range of target applications, relaxing such restrictive assumptions is necessary. Moreover, considering the effects of more and more factors that influence the timing properties of the tasks decreases the pessimism of the analysis by determining tighter worst case response times and leading to a smaller number of false negatives (which can appear when a system that is practically schedulable cannot be proven so by the analysis). Over the time, extensions have been offered to response time analysis for fixed priority scheduling by taking into account task synchronisation [Sha90], arbitrary deadlines [Leh90], precedence constraints between tasks [Pal99] and tasks with varying execution priorities [Gon91], arbitrary release times [Aud93], [Tin94c], tasks which suspend themselves [Pal98], tasks modelling code with conditional branches [Bar98], tasks running on multiprocessor systems [Tin94a], [Pal98], etc. The fixed priority analysis has been also adapted for the situation when tasks are running under the EDF scheduling policy [Pal03].

In spite of the fact that the duality between different implementations of scheduling algorithms has been suggested in [Dob01a] and [Dob01b], where fixed priority scheduling has been adapted in such a way that it emulates static cyclic schedules which are generated off-line, the growing amount and diversity of functionality that has to be implemented on current embedded systems has led to the necessity for concurrently using several scheduling policies in the implementation of the application running on a given system. In [Gon03], the authors present the schedulability analysis for a hierarchical scheduling policy called EDF-within-fixed-priorities, that combines fixed priority and EDF scheduling. The assignment of server parameters for a two-level hierarchical scheduler based on a resource reservation approach has been studied in [Lip04]. In [Ric02] and [Ric03], the authors model the multiprocessor heterogeneous systems as components that communicate through event streams and propose a technique for integrating different local scheduling policies based on such event-model interfaces. Another compositional approach is presented in [Wan05], where the authors propose real-time interfaces and a component model that support incremental design of real-time systems.

1.3.3 COMMUNICATION IN REAL-TIME SYSTEMS

Many safety-critical applications, following physical, modularity or safety constraints, are implemented using distributed architectures composed of several different types of hardware units (called nodes), interconnected in a network. For such systems, the communication between functions implemented on different nodes has an important impact on the overall system properties, such as performance, cost and maintainability.

There are several communication protocols for real-time networks. Among the protocols that have been proposed for in-vehicle communication, the Controller Area Network (CAN) [Bos91], the Local Interconnection Network (LIN) [LIN07], and SAE's J1850 [SAE94] are currently in use on a large scale [Nav05]. Moreover, only a few of the proposed protocols are suitable for safety-critical applications where predictability is mandatory [Rus01].

Communication activities can be triggered either dynamically, in response to an event (event-driven), or statically, at predetermined moments in time (time-driven). Therefore, on one hand, there are protocols that schedule the messages statically based on the progression of time, such as the SAFEbus [Hoy92], SPIDER [Min00], TTCAN [ISO02], and Time-Triggered Protocol (TTP) [Kop03]. The main drawback of such protocols is their lack of flexibility. On the other hand, there are communication protocols where message scheduling is performed dynamically, such as Byteflight [Ber03] introduced by BMW for automotive applications, CAN [Bos91], LonWorks [Eche07] and Profibus [Pro01].

A lot of work has been concentrated on coping with some of the disadvantages of the ST/DYN approaches and on trying to combine their advantages. For example, in [PopP01a], [PopP01b] and [PopP04b], the authors present a method for dealing with flexibility in TTP based systems by considering consecutive design stages in a so called *incremental design flow*. Similarly, a large number of schemes have been targeted at improving the real-time properties of communication protocols that may be cathegorised as extremely dynamic, like is the case of Ethernet [Fan05].

The aspects related to communication in real-time systems are receiving a continuously increasing attention in the literature. Building safety critical real-time systems requires consideration for all the factors that influence the timing properties of a system. For the case of distributed systems, in order to guarantee the timing requirements of the activities in the system, one has to also consider the effects of communication aspects like the communication protocol, bus arbitration, clock synchronisation, packaging of messages, characteristics of the physical layer, etc. Due to the variety of communication protocols, scheduling and schedulability analysis involving particular communication protocols has become a prolific area of research. Following the similar model as the one developed for determining task response times under rate monotonic analysis, message transmission times have been analysed for protocols like TTP bus [Kop92], Token Ring [Ple92], [Tab00], FDDI [Agr94], Profibus [Tov99], ATM [Erm97], [Han97] and CAN bus [Tin94b], [Dav07].

In the case of bus-based distributed embedded systems, one of the main directions of evolution for communication protocols is towards mixed protocols, which support both ST and DYN traffic. The proponents of the Time-Triggered Architecture showed that TTP can be enhanced in order to transmit event-triggered messages, while still maintaining time composability and determinism of the system, properties which are normally lost in event-triggered systems [Kop92]. A modified version of CAN, called Flexible Time-Triggered CAN [Alm99], [Alm02], and a similar extension for Ethernet called FTT-Ethernet [Ped05] have been provided under the Flexible Time-Triggered paradigm [Ped03], in which the communication cycles are divided into asynchronous and synchronous windows. Several other mixed communication protocols can be found in [Fuh00], [Wor03].

Following this trend, a large consortium of automotive manufacturers and suppliers has recently proposed such a hybrid type of protocol, namely the FlexRay communication protocol [Fle07]. FlexRay allows the sharing of the bus among static and dynamic messages, thus offering the advantages of both worlds. Due to its flexible properties and growing support from its target industry, FlexRay will possibly become the de-facto standard for in-vehicle communications. However, before it can be successfully deployed in applications that require predictability, timing analysis techniques are necessary to provide bounds for the message communication times [Nav05].

FlexRay is composed of static (ST) and dynamic (DYN) segments, which are arranged to form a bus cycle that is repeated periodically. The ST segment is similar to TTP, and employs a generalized time-division multiple-access (GTDMA) scheme. The DYN segment of the FlexRay protocol is similar to Byteflight and uses a flexible TDMA (FTDMA) bus access scheme.

Although researchers have proposed analysis techniques for dynamic protocols such as CAN [Tin95], TDMA [Tin94a], ATM [Erm97], Token Ring protocol [Str89], FDDI protocol [Agr94] and TTP [PopP00a], none of these analyses is applicable to the DYN segment in FlexRay. In

[Din05], the authors consider the case of a hard real-time application implemented on a FlexRay bus. However, in their discussion they restrict themselves exclusively to the static segment, which means that, in fact, only the classical problem of communication scheduling over a TDMA bus [PopP04a], [Ham05] is considered. The performance analysis of the Byteflight protocol, which is similar to the DYN segment of FlexRay, is discussed in [Cen04]. The authors, however, assume a very restrictive "quasi-TDMA" transmission scheme for time-critical messages, which basically means that the DYN segment would behave as an ST segment (similar to TDMA) in order to guarantee timeliness.

1.4 Thesis Contributions

The studies covered in this thesis consider distributed embedded systems implemented with heterogeneous, event-triggered and time-triggered task sets, which communicate over bus protocols consisting of both static and dynamic phases.

We have considered that the time-triggered activities are executed according to a static cyclic schedule, while the event-triggered activities follow a fixed priority scheduling or an EDF scheduling policy, which is preemptive for the execution of tasks and non-preemptive for the transmission of messages. For message exchange over the bus we have considered two heterogeneous communication protocols: UCM in the first part and FlexRay in the second part.

The main contributions of this thesis are threefold. First, we propose a holistic schedulability analysis for heterogeneous TT/ET task sets which communicate through mixed ST/DYN communication protocols [PopT02], [PopT03a]. Such an analysis presents two aspects:

a. It computes the response times of the ET activities while considering the influence of a static schedule;

b. It builds a static cyclic schedule for the TT activities while trying to minimise the response times of the ET activities.

Second, we show how the scheduling and schedulability analysis can be used inside a design optimisation loop in order to improve the timing properties of the system [PopT03b], [PopT05], [PopT07b].

Third, we apply the analysis and optimisation methodology developed in the first two steps on a particular case of systems that use FlexRay as a communication protocol [PopT06], [PopT07a], [PopT07c].

1.5 Thesis Overview

The remaining part of the thesis can be divided into two parts:

The first part (Chapters 2-4) deals with distributed embedded real-time systems that use heterogeneous scheduling policies. Chapter 2 presents the system model we used. In Chapter 3, we present our analysis method for deriving response times for all tasks and messages in such an embedded system. In Chapter 4, we first discuss some optimisation aspects which are particular to the studied systems, and then we define and solve the design optimisation problem that aims at improving the overall system schedulability.

The second part of the thesis (Chapters 5-7) concentrates on the same category of distributed embedded systems, but for the particular case when the communication protocol is FlexRay. First we introduce the specifics of FlexRay in Chapter 5. Then, in Chapter 6 we present the timing analysis that determines the worst-case response times of messages transmitted over a FlexRay bus. Following a similar line of thought like in the first part of the thesis, Chapter 7 aims again at improving the overall timing characteristics of the system by optimising the structure of the bus cycle.

Finally, in Chapter 8 we draw some conclusions and discuss possible research directions for the future.

Chapter 2
System Model

IN THIS CHAPTER we present the system model that we use during scheduling and design optimisation. First, we briefly describe the hardware architecture and the structure of the bus access cycle. Then, we present our hierarchy of schedulers that implements the software architecture for a system which is able to run both event-triggered and time-triggered activities. The last part of this chapter presents the abstract representation which we use for modelling the applications that are assumed to implement the functionality of the system.

2.1 Hardware Architecture

We consider architectures consisting of nodes connected by a unique broadcast communication channel. Each node consists of:

- a *communication controller* which controls the transmission and reception of both ST and DYN messages;
- a *CPU* for running the processes mapped on that particular node;
- *local memories* for storing the code of the kernel (ROM), the code of the processes and the local data (RAM); and

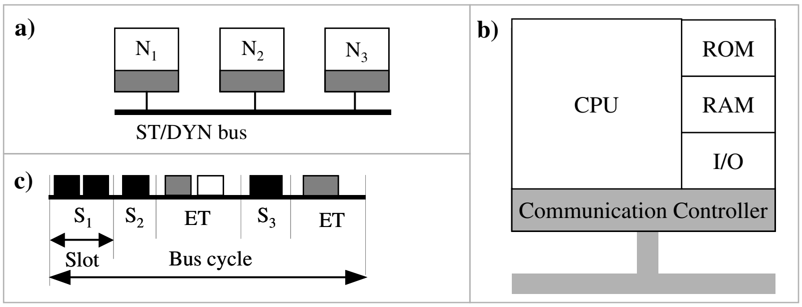


Figure 2.1: System Architecture

- *I/O interfaces* to sensors and actuators.

Such hardware architectures are common in applications such as automotive electronics and robotics. In Figure 2.1.a, we illustrate a heterogeneous distributed architecture composed of three nodes interconnected by a bus based infrastructure. The model considered for the processing nodes in the architecture is depicted in Figure 2.1.b.

2.2 Bus Access

Every node in the architecture has a communication controller that implements the static and dynamic protocol services. The controller runs independently of the node's CPU. In this first part of the thesis we model the heterogeneous bus access scheme using the Universal Communication Model [Dem01]

The bus access is organized as consecutive cycles, each with the duration T_{bus}. We consider that the communication cycle is partitioned into static (ST) and dynamic (DYN) phases (Figure 2.1.b).

- ST phases consist of time slots, and during a slot only the node associated to that particular slot is allowed to transmit ST messages. The transmission times of ST messages are stored in a schedule table.

- During a DYN phase, all nodes are allowed to send messages and the conflicts between nodes trying to send simultaneously are solved by an arbitration mechanism which allows the transmission of the message with the highest priority. Hence, the DYN messages are organized in a prioritised ready queue.

2.3 Software Architecture

For the systems we are studying, we have designed a software architecture that runs on the CPU of each node. The main component of the software architecture is a real-time kernel. The real-time kernel contains three scheduler types organized *hierarchically* (Figure 2.2):

1. The top-level scheduler is a static cyclic scheduler (SCS), that is responsible for the activation of TT tasks and transmission of ST messages based on a schedule table, and for the activation of the FPS scheduler. As a consequence, TT tasks and ST messages are activated at predetermined points in time, and their execution/transmission is non-

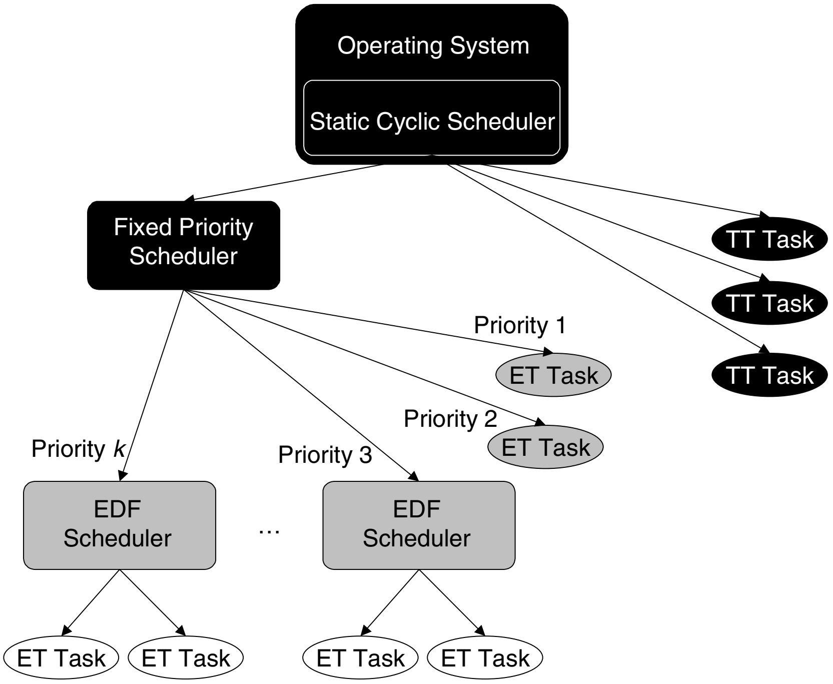


Figure 2.2: Software Architecture

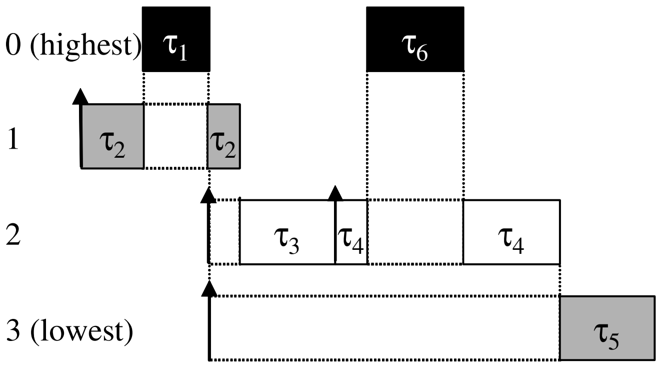


Figure 2.3: Execution of Tasks in a Hierarchical Scheduler

preemptable.

2. The second level in the hierarchy consists of a fixed-priority scheduler (FPS) that activates the execution of ET tasks and transmits DYN messages based on their priorities. It also activates the EDF schedulers that are described below. Tasks and messages scheduled under FPS are initiated whenever a particular event is noted. We consider that the execution of ET tasks under the fixed priority scheduling is preemptable.
3. The third level in the hierarchy consists of a set of schedulers that follow the earliest-deadline-first (EDF) scheduling policy. In the case that on a node there are several activities that share the same priority inside the FPS scheduler on the second level, then their execution is controlled by such an EDF scheduler that activates ET tasks and sends DYN messages based on their deadlines. We consider that the execution of ET tasks under the EDF policy is preemptable.

From this point and throughout the rest of the thesis, we will use the terms “SCS tasks”, “EDF tasks” or “FP tasks” whenever we want to emphasise the scheduling policy under which certain tasks are executed.

When several tasks are ready on a node, the task with the highest priority is activated, and preempts the other tasks. Let us consider the example in Figure 2.3, where we have six tasks sharing the same node. Tasks τ_1 and τ_6 are scheduled using SCS, τ_2 and τ_5 are scheduled using FPS, while tasks τ_3 and τ_4 are scheduled with EDF. The priorities of the FPS and EDF tasks are indicated in the figure. The arrival time of these tasks is depicted with an upwards pointing arrow. Under these assumptions, Figure 2.3

presents the worst-case response times of each task. The SCS tasks, τ_1 and τ_6, will never compete for a resource because their synchronization is performed based on the schedule table. Moreover, since SCS tasks are non preemptable and their start time is off-line fixed in the schedule table, they also have the highest priority (denoted with priority level “0” in the figure). FPS and EDF tasks can only be executed in the *slack* of the SCS schedule table.

FPS and EDF tasks are scheduled based on their priorities. Thus, a higher priority task such as τ_2 will interrupt a lower priority task such as τ_3. In order to integrate EDF tasks with FPS, we use the approach in [Gon03], by assuming that FPS priorities are not unique, and that a group of tasks having the same FPS priority on a processor are to be scheduled with EDF. Thus, whenever the FPS scheduler notices ready tasks that share the same priority level, it will invoke the EDF scheduler which will schedule those tasks based on their deadlines. Such a situation is present in Figure 2.3 for tasks τ_3 and τ_4. There can be several such EDF priority levels within a task set on a processor. Higher priority EDF tasks can interrupt lower priority FPS tasks (as is the case with τ_3 and τ_4 which preempt τ_5) and EDF tasks. Lower priority EDF tasks will be interrupted by both higher priority FPS and EDF tasks, and SCS tasks.

TT activities are triggered based on a local clock available in each processing node. The synchronization of local clocks throughout the system is provided by the communication protocol.

2.4 Application Model

We model an application as a set of task graphs. Nodes in the graphs represent tasks, and arcs represent communication (and implicitly dependency) between the connected tasks. An edge from a task τ_{ij} to τ_{ik} indicates that the output of τ_{ij} is the input of τ_{ik}. The set of all tasks is denoted with $\mathcal{T}$. A task becomes ready after all its inputs have arrived and it issues its outputs when it terminates. A message will become ready after its sender task has finished, and becomes available for the receiver task after its transmission has ended.

- A task can belong either to the TT or to the ET domain. We consider that the scheduling policy for each task is known: TT tasks are scheduled using SCS, while ET tasks are scheduled under FPS and EDF.

- Communication between tasks mapped to different nodes is performed by message passing over the bus. Such a message passing is modelled as a communication task inserted on the arc connecting the sender and the receiver tasks. The communication time between tasks mapped on the same node is considered to be part of the task execution time. Thus, such a communication activity is not modelled explicitly. For the rest of the thesis, when referring to messages we consider only the communication activity over the bus.

- A message can belong either to the static (ST) or to the dynamic (DYN) domain. We consider that static messages are those sent during the ST phases of the bus cycle, while dynamic messages are those transmitted during the DYN phases.

- All tasks in a certain task graph belong to the same domain, either ET, or TT, which is called the domain of the task graph. The messages belonging to a certain task graph can belong to any domain (ST or DYN). Thus, in the most general case, tasks belonging to a TT graph, for example, can communicate through both ST and DYN messages. However, in this thesis we restrict our discussion to the situation when TT tasks communicate through ST messages and ET tasks communicate through DYN messages.

- Each task τ_{ij} (belonging to the task graph Γ_i) has a period T_{ij}, and a deadline D_{ij} and, when mapped on node $Node_k$, it has a worst case execution time $C_{ij}(Node_k)$. The node on which τ_{ij} is mapped is denoted as $\mathcal{M}(\tau_{ij})$. Each ET task also has a given priority $Prio_{ij}$. Individual release times or deadlines of tasks can be modelled by introducing dummy tasks in the task graphs; such dummy tasks have an appropriate execution time and are not mapped on any of the nodes [Ele00a].

- All tasks τ_{ij} belonging to a task graph Γ_i have the same period T_i

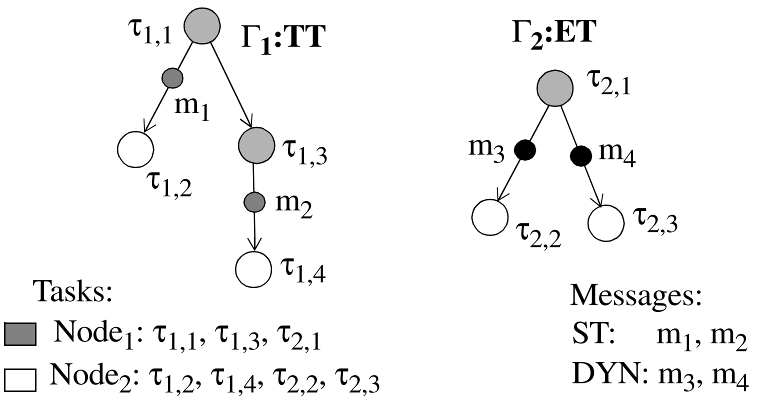


Figure 2.4: Application Model Example

which is the period of the task graph. The period of a message is identical with that of the sender task. If communicating tasks are of different periods, they are combined into a larger graph capturing all task activations for the hyper-period (LCM of periods).

- We also consider that the size of each message *m* is given, which can be directly converted into communication time C_m on the particular bus, knowing the speed of the bus and the size of the frame that stores the message:

$$C_m = Frame\_size(m)\ /\ bus\_speed. \tag{2.1}$$

Figure 2.4 shows an application modelled as two task-graphs Γ_1 and Γ_2 mapped on two nodes, $Node_1$ and $Node_2$. Task-graph Γ_1 is time-triggered and task-graph Γ_2 is event-triggered. Data-dependent tasks mapped on different nodes communicate through messages transmitted over the bus, which can be either statically scheduled, like m_1 and m_2, or dynamic, like the messages m_3 and m_4.

In order to keep the separation between the TT and ET domains, which are based on fundamentally different triggering policies, communication between tasks in the two domains is not included in the model. Technically, such a communication is implemented by the kernel, based on asynchronous non-blocking send and receive primitives (using proxy tasks if the sender and receiver are on different nodes). Such messages are typically non-critical and are not affected by hard real-time constraints.

Chapter 3 Scheduling and Schedulability Analysis

In this chapter we present an analytic approach for computing worst-case task response times and worst-case message transmission delays for heterogeneous TT/ET systems.

3.1 Problem Formulation

Given an application and a system architecture as presented in Chapter 2, the following problem has to be solved: construct a correct static cyclic schedule for the TT tasks and ST messages (a schedule which meets all time constraints related to these activities), and conduct a schedulability analysis in order to check that all ET tasks and DYN messages meet their deadlines. Two important aspects should be noticed:

1. When performing the schedulability analysis for the ET tasks and DYN messages, one has to take into consideration the interference from the statically scheduled TT tasks and ST messages.
2. Among the possible correct schedules for TT tasks and ST messages, it is important to construct one which favours, as much as possible, the schedulability of ET tasks and DYN messages.

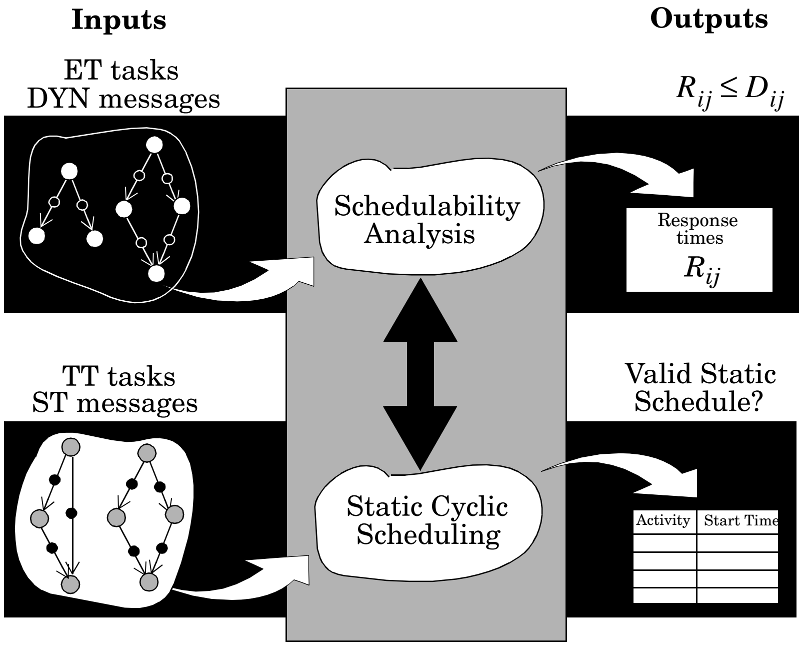


Figure 3.1: Scheduling and Schedulability Analysis for Mixed TT/ET Distributed Embedded Systems

In the next sections, we will present the schedulability analysis algorithm proposed in [Pal98] for distributed real-time systems and we will show how we extended this analysis in order to consider the interferences induced by an existing static schedule. Section 3.2 presents a general view over our approach for the global scheduling and schedulability analysis of heterogeneous TT/ET distributed embedded systems. Section 3.3 describes the regular schedulability analysis for FPS and EDF tasks sharing the same resources, as developed in [Gon03]. Section 3.4 extends the schedulability analysis so that SCS tasks are taken into consideration when computing the response times of FPS and EDF activities. In Section 3.5 we present our complete scheduling algorithm, which statically schedules the TT activities while trying to minimise the influence of the TT activities onto the ET ones. The performance of our approach is evaluated in Section 3.6, where we present the experimental results.

It has to be mentioned that our analysis is restricted, for the moment, to the model in which TT tasks communicate only through ST messages, while communication between ET tasks is performed by DYN messages. However, this is not an inherent limitation of our approach. For example, schedulability analysis of ET tasks communicating through ST messages has been presented in [PopP00b] and [PopP03a].

3.2 Holistic Scheduling

Figure 3.1 illustrates our strategy for scheduling and schedulability analysis of heterogeneous TT/ET distributed embedded systems: the activities to be scheduled are the TT and ET task graphs, consisting of TT tasks/ST messages and ET tasks/DYN messages respectively. The TT activities are statically scheduled and, as an output, a static cyclic schedule will be produced. Similarly, the worst case response times of the ET activities are determined using the schedulability analysis that will be described in detail in the following sections. As a result, the system is considered schedulable if the static schedule is valid and if the ET activities are guaranteed to meet their deadlines. For the case of a mixed TT/ET system, building a static cyclic schedule for the TT activities has to take into consideration both the characteristics of the mixed ST/DYN communication protocol and our assumption that execution of TT tasks is non-preemptible, while the execution of an ET task can be interrupted either by a TT task or by another ET task which has a higher priority. This means that the static schedule will have not only to guarantee that TT activities meet their deadlines, but also that the interference introduced from such a schedule will not increase in an unacceptable way the response times of ET activities. In conclusion, an efficient scheduling algorithm requires a close interaction between the static scheduling of TT activities and the schedulability analysis of the ET activities.

3.3 Schedulability Analysis of Event-Triggered Task Sets

In order to determine if a hierarchically scheduled system is schedulable, we used as a starting point the schedulability analysis algorithm for EDF-within-FPS systems, developed in [Gon03]. In this section, we present our extension to this algorithm, which allows us to compute the worst case response times for the FPS and EDF activities when they are interfered by the SCS activities.

An ET task graph Γ_i is activated by an associated event which occurs with a period T_i. ET tasks (FPS or EDF) and DYN messages are modelled similarly, by considering the bus as a processing node and accounting for the non-preemptability of the messages during the analysis. Each activity τ_{ij} (task or message) in an ET task graph has an offset ϕ_{ij} which specifies the earliest activation time of τ_{ij} relative to the occurrence of the triggering event. The delay between the earliest possible activation time of τ_{ij} and its actual activation time is modelled as a jitter J_{ij} (Figure 3.2). The response time $R^p{}_{ij}$ of the p-th job of a task τ_{ij} is the time measured from the occurrence of the associated event until the completion of the p-th job of τ_{ij}. For example, Figure 3.2 depicts the execution of two jobs for each of the tasks τ_{ij} and τ_{ij+1}. The worst-case response time R_{ij} of a task τ_{ij} is the longest possible response time for that task, considering all its possible jobs:

$$R_{ij} = max(R^p{}_{ij}), \forall p \tag{3.1}$$

Similarly, each task τ_{ij} has a best case response time $R_{b,ij}$:

$$R_{b,\,ij} = min(R^p{}_{ij}), \forall p \tag{3.2}$$

Offsets are the means by which dependencies among tasks can be modelled for the schedulability analysis. For example, if in Figure 3.2, task τ_{ij+1} is data dependant on task τ_{ij}, then such a relation can be enforced by associating to τ_{ij+1} an offset ϕ_{ij+1} which is equal or greater than the worst

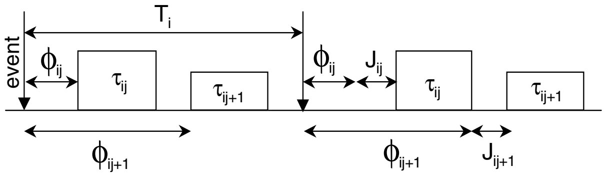


Figure 3.2: Tasks with Offsets

case response time R_{ij} of its predecessor, τ_{ij}. In this way, it is guaranteed that task τ_{ij+1} starts only after its predecessor has finished execution.

In [Gon03], the authors have developed a schedulability analysis algorithm for ET tasks running under a hierarchical FPS/EDF scheduling policy. The worst-case response time R_{ab} of a task τ_{ab}, regardless if it is scheduled under FP or EDF, is computed by considering:

- all possible critical instants initiated by the higher priority tasks τ_{ac} from Γ_a. A *critical instant* is the moment when task τ_{ab} is released. At the same time, higher priority tasks may be released, delaying the execution of task τ_{ab}.
- all the jobs p of τ_{ab} that appear during an interval of time called a *busy period*. For FPS tasks, the busy period is an interval of time that starts at the critical instant t and ends when all higher priority tasks (released at or after t) and task τ_{ab} finish execution. For EDF tasks, the busy period is defined in a similar way, but the release of task τ_{ab} may take place at a later instant in the busy period. This is caused by the fact that tasks with shorter deadlines may be released earlier than the critical instant, meaning that for EDF tasks the start of the busy period does not necessarily coincide with the critical instant.

For all considered situations, a response time $R_{abc}(p)$ is computed, which leads to the value of the worst-case response time R_{ab} for τ_{ab} to be determined as:

$$R_{ab} = \max_{c} \left(\max_{p} \left(R_{abc}(p)\right)\right), \forall \tau_{ac} \in hp(\tau_{ab}), \forall p \qquad (3.3)$$

Response times for the FPS and EDF tasks respectively are obtained using workload equations. Given a task τ_{ab} and an interval of time t, a workload equation computes the amount of interference that τ_{ab} can suffer during t from higher priority tasks and from tasks with earlier deadlines.

For FPS tasks, the worst case response times are influenced only by higher priority tasks, so the completion time of an activation p of task τ_{ab}, when the critical instant is initiated by a higher priority task τ_{ac} is given by:

$$R_{abc}(p) = w_{abc}(p) + \phi_{ab} - (p-1) \cdot T_a - \varphi_{abc} \tag{3.4}$$

where ϕ_{ab} is the offset of τ_{ab}, T_a is the period of task-graph Γ_a, φ_{abc} is the phase between event arrivals and the critical instant, and $w_{abc}(p)$ is the busy period. The phase φ_{abc} is computed as:

$$\varphi_{abc} = T_a - (\phi_{ac} + J_{ac} - \phi_{ab}) mod\ T_a \tag{3.5}$$

where ϕ_{ac} and J_{ac} are the offset and the jitter of the task τ_{ac} that initiates the critical instant. The busy period is determined using the following equation:

$$w_{abc}(p) = B_{ab} + (p - p_{0,abc} + 1) \cdot C_{ab} + W_{ac}(\tau_{ab}, w_{abc}(p)) + \sum_{\forall i \neq a} W_i'(\tau_{ab}, w_{abc}(p)) \tag{3.6}$$

where B_{ab} is the blocking time of τ_{ab}; the second term captures the jobs of τ_{ab} that appear during the considered time interval; W_{ac} is the worst case interference produced by higher priority tasks in Γ_a when the critical instant is initiated by τ_{ac}:

$$W_{ik}(\tau_{ab}, t) = \sum_{j \in hp_a(\tau_{ab})} \left(\left\lfloor \frac{J_{ij} + \varphi_{ijk}}{T_i} \right\rfloor + \left\lceil \frac{t - \varphi_{ijk}}{T_i} \right\rceil \right) \cdot C_{ij} \tag{3.7}$$

and W'_i is the worst-case interference produced by higher priority tasks in a task-graph Γ_i different than Γ_a. This interference is computed using the following equation:

$$W_i'(\tau_{ab}, t) = max(W_{ik}(\tau_{ab}, t)), \forall k \in hp_i(\tau_{ab}) \tag{3.8}$$

For EDF tasks, the worst-case response times are influenced by higher priority tasks and by EDF tasks running at the same priority level as the task under analysis. Consequently, the worst-case interference produced by a task-graph Γ_i on the execution of the analysed task τ_{ab} will contain two parts that are added together: the EDF interference W_{ik}^{EDF}, and the FPS interference W_{ik}^{FP}. The worst-case interference W_{ik}^{FP} of strictly higher priority tasks in a task-graph is computed based on Equation (3.7). For the tasks running at the same priority level, the scheduling policy is EDF, which leads to the following modification of the interference equation:

$$W_{ik}^{EDF}(\tau_{ab}, t, D) = \sum_{\tau_{ij} \in ep_i(\tau_{ab})} min_0(wT_{ijk}(t), wD_{ijk}(t, D)) \cdot C_{ij} \tag{3.9}$$

where D is the deadline of the analysed task τ_{ab}, $ep_i(\tau_{ab})$ is the set of tasks of equal priority in Γ_i, and the two members of the *min* function have the following equations:

$$wT_{ijk}(t) = \left\lfloor \frac{J_{ij} + \varphi_{ijk}}{T_i} \right\rfloor + \left\lceil \frac{t - \varphi_{ijk}}{T_i} \right\rceil \tag{3.10}$$

and

$$wD_{ijk}(t, D) = \left\lfloor \frac{J_{ij} + \varphi_{ijk}}{T_i} \right\rfloor + \left\lceil \frac{D - D_{ij} - \varphi_{ijk}}{T_i} \right\rceil . \tag{3.11}$$

The interference equation from Equation (3.8) becomes:

$$W_i'(\tau_{ab}, t) = max(W_{ik}^{FP}(\tau_{ab}, t) + W_{ik}^{EDF}(\tau_{ab}, t, D) \tag{3.12}$$

Besides the situations discussed for FPS analysis in Equation (3.3), when computing the worst-case response time for an EDF task τ_{ab}, we have to consider in addition all the possible scenarios in which τ_{ab} has a deadline larger or equal than the deadline of the other EDF tasks in the system running at the same priority $Prio_{ab}$:

$$R_{ab} = \max_c (\max_p (R_{abc}^A(p))), \forall \tau_{ac} \in hp(\tau_{ab}), \forall p, \forall A \tag{3.13}$$

where A is a time instant when the job p under analysis is released and has the deadline equal with that of another EDF task running on the same priority level.

The response time for a job p of task τ_{ab} running under EDF is determined using an equation similar to Equation (3.4):

$$R_{abc}^A(p) = w_{abc}{}^A(p) - \varphi_{abc} - (p-1) \cdot T_a - A \tag{3.14}$$

Let us see how these equations can be integrated in an algorithm that determines the worst-case response times of all the ET activities in the

```
1 do
2   Done = true
3   for each transaction $\Gamma_a$ do
4     for each activity $\tau_{ab}$ in $\Gamma_a$ do
5       for each activity $\tau_{ac}$ in $\Gamma_a$ do
6         if $Prio_{ac} \geq Prio_{ab}$ and $\mathcal{M}(\tau_{ac}) = \mathcal{M}(\tau_{ab})$ then
7           for each job p of $\tau_{ab}$ do
8             Consider that $\tau_{ac}$ initiates critical instant
9             Compute $R_{abc}(p)$
10            if $R_{abc}(p) > R_{ab}^{max}$ then
11              $R_{ij}^{max} = R_{abc}(p)$
12            endif
13          endfor
14        endif
15      endfor
16      if $R_{ab}^{max} > R_{ab}$ then -- larger response time found
17        $R_{ab} = R_{ab}^{max}$
18        Done = false
19        for each successor $\tau_{ac}$ of $\tau_{ab}$ do
20          $J_{ac} = R_{ab} - R_{ab}^{b}$ -- update jitters
21        endfor
22      endif
23    endfor
24  endfor
25 while (Done!= true)
```

Figure 3.3: Schedulability Analysis Algorithm

system. Figure 3.3 shows the pseudocode for the schedulability analysis proposed in [Gon03]. According to this algorithm, the worst case response time R_{ab} of each task τ_{ab} is computed by considering all critical instants initiated by each task τ_{ac} mapped on the same node $\mathcal{M}(\tau_{ab})$ and with a higher or equal priority than $Prio_{ab}$ (lines 3-6). For each such instant, the associated response time is computed using Equations (3.4) and (3.14) (line 9). According to the same schedulability analysis, jitters are taken into consideration when the algorithm computes the length of the busy windows and, implicitly, the response times of the tasks [Pal98]. This means that the length of the busy window depends on the values of task jitters, which, in turn, are computed as the difference between the worst-case and best-case response times of the preceding tasks (for example, if τ_{ab} precedes τ_{ac} in Γ_a, then $J_{ac} = R_{ab} - R^b{}_{ab}$, like in lines 20-21 in Figure 3.3). Because of this cyclic dependency (response times depend on jitters and jitters depend on response times), the process of computing R_{ab} is an iterative one: it starts by assigning $R^b{}_{ab}$ to R_{ab} and then computes the values for jitters J_{ab}, busy windows $w_{abc}(p)$ and then again the worst-case response times R_{ab}, until the response times converge to their final value. Such a fixed-point iteration is captured in the algorithm by the *do-while* loop in lines 1-25. The convergence of the analysis is captured by the boolean variable *Done*, that is set to false each time a larger response time is found for the analysed tasks (line 18). If the variable *Done* is still true after an iteration inside the *do-while* loop, then the algorithm ends, since it has determined the worst-case response times for all the ET activities in the system.

In order to be able to analyse systems that use the hierarchical combination of SCS, FPS and EDF scheduling policies described in Section 2.3, the technique presented in this section has to be enhanced so that it considers the interference from an existing static schedule. We will describe such an extension in the next section.

3.4 Schedulability Analysis of Event-Triggered Activities under the Influence of a Static Cyclic Schedule

Considering the algorithm presented in the previous section as a starting point, we have to solve the following problem: compute the worst case response time of a set of ET tasks and DYN messages by taking into consideration:

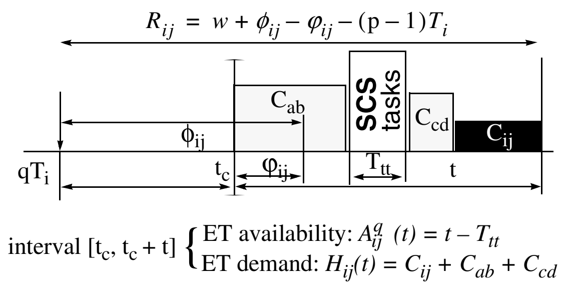

$R_{ij} = w + \phi_{ij} - \varphi_{ij} - (p-1)T_i$
C_{ab}
SCS tasks
C_{cd}
C_{ij}
ϕ_{ij}
qT_i
t_c
φ_{ij}
T_{tt}
t
interval $[t_c, t_c + t]$
ET availability: $A^q_{ij}(t) = t - T_{tt}$
ET demand: $H_{ij}(t) = C_{ij} + C_{ab} + C_{cd}$

Figure 3.4: Availability and Demand

- The interference from the set of statically scheduled tasks.
- The characteristics of the communication protocol, which influence the worst case delays induced by the messages communicated on the bus.

First, we extend the notions of *ET availability* and *demand* introduced in [Ped00] for analysing heterogeneous ST/DYN communication protocols. Our extension applies the same concepts to task execution. We start by defining the *ET demand* for an FPS or EDF activity τ_{ij} over a time interval *t* as the maximum amount of CPU time or bus time which can be *demanded* by higher or equal priority ET activities and by τ_{ij} during the time interval *t*. In Figure 3.4, the ET demand of the task τ_{ij} during the busy window *t* is denoted with $H_{ij}(t)$, and it is the sum of worst case execution times for task τ_{ij} and two other higher priority tasks τ_{ab} and τ_{cd}. During the same interval *t*, we define the *ET availability* as the processing time which is not used by statically scheduled activities. In Figure 3.4, the CPU availability for the analysed interval is obtained by substracting from *t* the amount of processing time needed for the statically scheduled activities. Figure 3.4 presents how the *availability* $A^q{}_{ij}(t)$ and the *demand* $H_{ij}(t)$ are computed for a task τ_{ij}: the busy window of τ_{ij} starts at the critical instant $q\ T_i + t_c$ initiated by task τ_{ab} and ends at moment $qT_i + t_c + t$, when both higher priority tasks (τ_{ab}, τ_{cd}), all TT tasks scheduled for execution in the analysed interval, and τ_{ij} have finished execution.

```
1 w_ij = p • C_ij + B_ij
2 do
3   Compute demand H_ij(w_ij)
4   Compute availability A_ij(w_ij)
5   if H_ij(w_ij) > A_ij(w_ij) then
6       w_ij += H_ij(w_ij) - A_ij(w_ij)
7   endif
8 while H_ij(w_ij) ≥ A_ij(w_ij)
9 return w_ij
```

Figure 3.5: Determining the Length of the Busy Window

During a time interval t, the *ET demand* H_{ij} associated with the task under analysis τ_{ij} is equal with the length of the busy window which would result when considering only ET activity on the system:

$$H_{ij}(t) = w_{ij}(t) \ . \tag{3.15}$$

During the same time interval t, the availability A_{ij} associated with task τ_{ij} is:

$$A_{ij}(t) = min\left[A_{ij}^{ab}(t)\right] , \forall\tau_{ab} \in TT \mid \mathcal{M}(\tau_{ab}) = \mathcal{M}(\tau_{ij}), \tag{3.16}$$

where $A_{ij}^{ab}(t)$ is the total available CPU-time on processor $\mathcal{M}(\tau_{ij})$ in the interval $[\phi_{ab}, \phi_{ab} + t]$, and ϕ_{ab} is the start time of task τ_{ab} as recorded in the static schedule table.

The discussion above is, in principle, valid for both types of ET tasks (i.e., FPS and EDF tasks) and messages. However, there exist two important differences. First, messages do not preempt each other, therefore, in the demand equation the blocking term will be non-zero, but equal with the largest transmission time of any ET message. Second, the availability for a message is computed by substracting from t the length of the ST slots which appear during the considered interval; moreover, because an ET message will not be sent unless there is enough time before the current dynamic phase ends, the availability is further decreased with C_A for each dynamic phase in the busy window (where C_A is the transmission time of the longest ET message).

Our schedulability analysis algorithm determines the length of a busy window w_{ij} for FPS and EDF tasks and messages by identifying the appropriate size of w_{ij} for which the ET demand is satisfied by the availability: $H_{ij}(w_{ij}) \le A_{ij}(w_{ij})$. The algorithm for computing the busy window w_{ij} for an activity τ_{ij} is shown in Figure 3.5. The algorithm consists of a loop that computes at each step the demand and availability for a given busy window (lines 3-4). If, for the analysed interval w_{ij}, the available time does not satisfy the ET demand, then the algorithm increases the length of the busy window with the needed amount of time (lines 5-7), and starts another iteration. The entire loop is ended only when the busy window w_{ij} is long enough to provide enough available time for the execution of all ET tasks that can appear during an interval of time of length w_{ij}. This procedure for the calculation of the busy window is included in the iterative process for calculation of response times presented in Section 3.3. It is important to notice that this process includes both tasks and messages and, thus, the resulted response times of the FPS and EDF tasks are computed by taking into consideration the delay induced by the bus communication.

After performing the schedulability analysis, we can check if $R_{ij} \le D_{ij}$ for all the ET tasks. If this is the case, the set of ET activities is schedulable. In order to drive the global scheduling process, as it will be explained in the next section, it is not sufficient to test if the task set is schedulable or not, but we need a metric that captures the "degree of schedulability" of the task set. For this purpose we use the function *DSch,* similar with the one described in [PopP00b]

$$DSch = \begin{cases} f_1 = \sum_{i=1}^{N} \sum_{j=1}^{N_i} max(0, R_{ij} - D_{ij}) & , \text{ if } f_1 > 0 \\ f_2 = \sum_{i=1}^{N} \sum_{j=1}^{N_i} (R_{ij} - D_{ij}) & , \text{ if } f_1 = 0 \end{cases} \tag{3.17}$$

where N is the number of ET task graphs and N_i is the number of activities in the ET task graph Γ_i.

If the task set is not schedulable, there exists at least one task for which $R_{ij} > D_{ij}$. In this case, $f_1 > 0$ and the function is a metric of how far we are from achieving schedulability. If the set of ET tasks is schedulable, $f_2 \le 0$ is used as a metric. A value $f_2 = 0$ means that the task set is "just" schedulable. A smaller value for f_2 means that the ET tasks are schedulable and a certain amount of processing capacity is still available.

Now, that we are able to perform the schedulability analysis for the ET tasks considering the influence from a given static schedule of TT tasks, we can go on to perform the global scheduling and analysis of the whole application.

3.5 Static Cyclic Scheduling of Time-Triggered Activities in a Heterogeneous TT/ET Environment

As mentioned in the beginning of 3.1, building the static cyclic schedule for the TT activities in the system has to be performed in such a way that the interference imposed on the ET activities is minimum. The holistic scheduling algorithm is presented in Figure 3.6. For the construction of the schedule table with start times for SCS tasks and ST messages, we adopted a list scheduling-based algorithm [Dav99] which iteratively selects tasks and schedules them appropriately.

A ready list contains all SCS tasks and messages which are ready to be scheduled (they have no predecessors or all their predecessors have been scheduled). From the ready list, tasks and messages are extracted one by one (Figure 3.6, line 2) to be scheduled on the processor they are mapped to, or into a static bus-slot associated to that processor on which the sender of the message is executed, respectively. The priority function which is used to select among ready tasks and messages is a critical path metric, modified for the particular goal of scheduling tasks mapped on distributed systems [Ele00a]. If the selected activity is a task, then the algorithm calls the *schedule_TT_task* procedure (line 4), that will be described later. When scheduling a ST message extracted from the ready list, we place it into the first bus-slot associated with the sender node in which there is sufficient space available (line 6). If all SCS tasks and messages have been scheduled and the schedulability analysis for the ET tasks and DYN messages indicates that all ET activities meet their deadlines, then the global system scheduling has succeeded. For the case that no correct schedule has been produced, we have implemented a backtracking mechanism in the list scheduling algorithm, which allows to turn back to previous scheduling steps and to try alternative solutions. In order to avoid excessive scheduling times, the maximum number of backtracking steps can be limited.

Let us consider a particular task τ_{ij} selected from the ready list to be scheduled. We consider that $ASAP_{ij}$ is the earliest time moment which satisfies the condition that all preceding activities (tasks or messages) of τ_{ij} are finished and processor $\mathcal{M}(\tau_{ij})$ is free. The moment $ALAP_{ij}$ is the latest time when τ_{ij} can be scheduled. With only the SCS tasks in the system, the straightforward solution would be to schedule τ_{ij} at $ASAP_{ij}$. We will call this approach Simple List Scheduling (SLS) and we will use it as a reference baseline during our experiments. However, if the system functionality contains both TT and ET activities, then such a simple solution could have negative effects on the schedulability of FPS and EDF tasks. What we have to do is to place the TT task τ_{ij} in such a position inside the interval [$ASAP_{ij}$, $ALAP_{ij}$] so that the chance to finally get a globally schedulable system is maximized.

In order to consider only a limited number of possible positions for the start time of a SCS task τ_{ij} inside the interval [$ASAP_{ij}$, $ALAP_{ij}$], we take

```
SimpleListScheduling($\mathcal{A}$, $\mathcal{M}$, $\mathcal{B}$, $\mathcal{S}$)
  1 while TT_ready_list is not empty
  2     select $\tau_{ij}$ from TT_ready_list
  3     if $\tau_{ij}$ is a task then
  4         schedule_TT_task($\tau_{ij}$, $\mathcal{M}(\tau_{ij})$)
  5     else -- $\tau_{ij}$ is a message
  6         ASAP schedule $\tau_{ij}$ in slot($\mathcal{M}(\tau_{ij})$)
  7     end if
  8 end while
  9 procedure schedule_TT_task($\tau_{ij}$, $\mathcal{M}(\tau_{ij})$)
  10    schedule $\tau_{ij}$ as soon as possible, at $ASAP_{ij}$
  11 end schedule_TT_task
end SimpleListScheduling
```

Figure 3.6: Holistic Scheduling Algorithm based on Simple List Scheduling

into account the information obtained from the schedulability analysis described in Section 3.4, which allows us to compute the response times of ET (i.e., FPS and EDF) tasks. We started from the observation that statically scheduling a SCS task τ_{ij} so that the length of busy-period of an ET activity is not modified will consequently lead to unchanged worst-case response time for that ET task. This can be achieved by providing for enough available processing time between statically scheduled tasks so that the busy period of the ET task does not increase. For example, in Figure 3.7 we can see how statically scheduling two SCS tasks τ_1 and τ_2 influences the busy period w_3 of a FPS (or EDF) task τ_3. Figure 3.7.a, presents the system with only τ_1 scheduled. In the worst case, the ET task τ_3 will become ready for execution at the same time $\phi_1$when τ_1 starts execution, thus being affected by the maximum possible interference from the task τ_1. Such a situation is unavoidable, and leads to the busy-period w_3 as depicted in Figure 3.7.a. Figures 3.7.b-c show how scheduling another SCS task τ_2 further affects the response time of the ET task. Scheduling task τ_2 too early (like in Figure 3.7.b) decreases the availability during the interval $[\phi_1, \phi_1 + w_3]$, and consequently leads to an increase of w_3 (to w'_3)

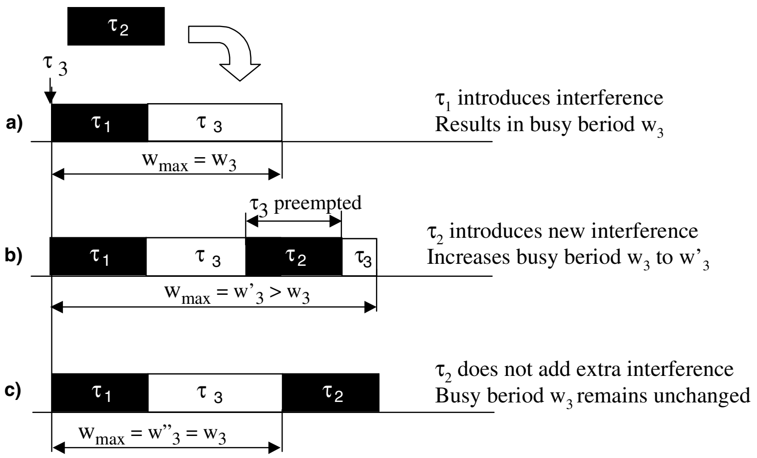
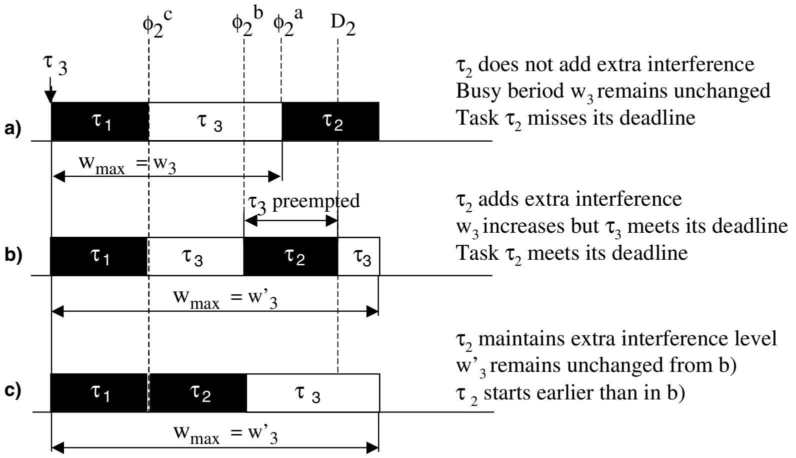

τ2
τ3
a)
τ1
τ3
τ1 introduces interference
Results in busy beriod w3
wmax = w3
τ3 preempted
b)
τ1
τ3
τ2
τ3
τ2 introduces new interference
Increases busy beriod w3 to w'3
wmax = w'3 > w3
c)
τ1
τ3
τ2
τ2 does not add extra interference
Busy beriod w3 remains unchanged
wmax = w"3 = w3

Figure 3.7: Static Scheduling, its influence over the execution of ET tasks (a-b) and minimisation of such an interference (c)

which in turn will increase the response time R_3. Such a situation is avoided if the two SCS tasks are scheduled like in Figure 3.7.c, where no extra interference is introduced in the busy period w_3. However, we notice that in such a situation, task τ_2 is scheduled later, which means that during the static scheduling we have to consider two aspects:

1. The interference with the FPS and EDF activities should be minimized;
2. The deadlines of TT activities should be satisfied.

The technique illustrated in Figure 3.7 takes care only of the first aspect, while ignoring the second one. It should be noticed that scheduling a SCS task later decreases the probability of finding feasible start times for that particular task. For example, in Figure 3.8.a, task τ_2 is scheduled like in Figure 3.7.c, so that no extra interference is added to the execution of ET activities. However, the busy period w_{max} pushes the start of τ_2 to such a late time in the static schedule, that the task misses its deadline and the resulted static schedule is not valid. This effect can further propagate to other SCS tasks that are scheduled later (for example, the successors of τ_2). In order to take care of both issues, we modified the list scheduling



ϕ_2^c
ϕ_2^b
ϕ_2^a
D_2
τ_3
a)
τ_1
τ_3
τ_2
$w_{max} = w_3$
τ_2 does not add extra interference
Busy beriod w_3 remains unchanged
Task τ_2 misses its deadline
τ_3 preempted
b)
τ_1
τ_3
τ_2
τ_3
$w_{max} = w'_3$
τ_2 adds extra interference
w_3 increases but τ_3 meets its deadline
Task τ_2 meets its deadline
c)
τ_1
τ_2
τ_3
$w_{max} = w'_3$
τ_2 maintains extra interference level
w'_3 remains unchanged from b)
τ_2 starts earlier than in b)

Figure 3.8: Optimised Static Scheduling: trade-off between minimising the TT interference over ET tasks (a) and allowing TT interference (b) while also maximising the chances of building a feasible static schedule (c)

algorithm so that at each step, whenever a TT task τ_k has to be scheduled, the following alternatives are evaluated:

1. scheduling the TT task late, so that no extra interference is added to the ET activities (consequently, though we decrease the risk of obtaining a feasible static schedule for the TT activities, there is a higher probability for ET activities to meet their deadlines);
2. scheduling the TT task earlier, at a time that will introduce a small TT amount of interference over ET activities (but we increase the probability that the resulted static schedule for TT tasks meets the deadlines).

The start time for the first case is determined by allowing enough slack time[1] between the SCS tasks so that the longest ET busy period in the system remains unchanged (like it has been illustrated in Figure 3.7.c). If the task being scheduled is τ_k, then we denote the determined value for the start time with $\phi_k{}^a$ (for example, the moment $\phi_2{}^a$ in Figure 3.8 has been determined using this kind of approach).

In order to discuss the second case, we use the example in Figure 3.8. Scheduling the TT task τ_2 at the time shown in Figure 3.8.a leads to a deadline miss, which means that we have to consider an earlier start time. In the example, we chose the latest possible time when τ_2 will meet its deadline: $\phi_2{}^b = D_2 - C_2$, where C_2 is the worst-case execution time of the task τ_2. As illustrated in Figure 3.8.b, we can notice that, while allowing task τ_2 to meet its deadline, such a schedule will introduce an additional interference over the execution of task τ_3, leading to a longer busy period w_3 and a longer response time R_3. However, considering that such an increase is acceptable (in the sense that all ET tasks will not miss their deadlines as a result of such a scheduling), we can now improve the probability of finding a valid static schedule. We achieve this by scheduling the task τ_2 even earlier in time, at a time $\phi_2{}^c = \max(ASAP_2, \phi_1 + C_1)$, where the value $\phi_1 + C_1$ represents the end time of the TT task τ_1 that has already been scheduled on the same processor during the previous steps. Our intention is to perform a transformation of the static schedule in such a way that the maximum ET busy period $w_{\max}$ does not increase. This technique is illustrated in Figure 3.7.c, where one can see that the same level of

1. We consider slack time any amount of processing time that is not reserved for the SCS tasks.

TT interference as in Figure 3.7.b is influencing the worst case response time of τ_3, but task τ_2 is scheduled much earlier, thus giving a chance for its successors to be statically scheduled at earlier moments, and consequently improving the probability of finding a valid static schedule.

However, in the more general case of distributed systems, the complex dependencies between tasks mapped on different processors do not allow us to easily transform the static schedule without affecting the interference over the execution of the ET activities. Actually, determining the values ϕ_k^b and ϕ_k^c for a given TT task τ_k is not a straightforward operation like in the example presented in Figure 3.8. In reality, for any given task τ_k there may be several pairs $\{\phi_k^b, \phi_k^c\}$ that have to be investigated. Our initial assumption was that inside the interval $S_k = [ASAP_k, ALAP_k]$ of possible start times for a TT task τ_k there are p distinct sub-intervals $S_k^i = [\phi_k^{b,i}, \phi_k^{b,i+1})$, with the following properties:

- $\bigcup_{i=1}^{p} S_k^i = S_k$ and $\bigcap_{i=1}^{p} S_k^i = \varnothing$.
- the global timing properties of at least one of the ET activities mapped on $\mathcal{M}(\tau_k)$ remain unchanged when task τ_k is scheduled to start at any $\phi_k \in S_k^i$ for a given sub-interval S_k^i .

Determining the sub-intervals with the properties described above is not an easy task. In [PopT03a], we have managed to determine such sub intervals for the restricted model when all task graphs in the system are synchronised, i.e. they all are initiated at the same time 0. In such a case, all the response times of the (in particular ET) activities being executed in the system can be referenced to the initial time 0. As a consequence, we noticed that there is no interference between a TT task and an ET task whenever the TT task is scheduled after an ET task has finished its execution. This means that, whenever we need to schedule a TT task τ_{ij}, the number p of sub intervals mentioned above is given by the number of ET tasks that finish inside the $[ASAP_{ij}, ALAP_{ij}]$ interval. For example, in Figure 3.9, we depict the alternative start times that need to be considered for a TT task τ_{ij} when there are three ET tasks (τ_{kl}, τ_{kl+1}, τ_{kl+2}) that end their execution inside the $[ASAP_{ij}, ALAP_{ij}]$ interval. The possible start times that we consider in our approach are $ASAP_{ij}$ and the response times

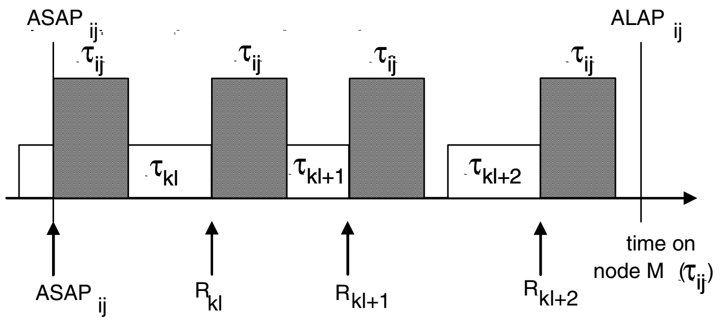


Figure 3.9: Alternative Start Times for a TT task τ_{ij} in the Model with Synchronised Task Graphs

of the three ET tasks: R_{kl}, R_{kl+1}, R_{kl+2}. One can notice that, for example, if the task τ_{ij} is scheduled at time R_{kl}, then there will be no interference added to the execution of task τ_{kl}. As shown in [PopT03a], using such an approach allowed us to explore efficiently the range of start times for any TT task and to rapidly find a valid static schedule with a reduced interference over the ET tasks.

However, for the more general application model that we consider in this thesis, each task graph has an arbitrary release time, which makes it impossible to use a common origin in time for the response times of all the activities in the system. In such a situation, when selecting the possible start times for a TT task, we decided to rely only on the length w_{max} of the longest ET busy period in the system. The first possible start time that we consider for a TT task is selected in such a way that the value of w_{max} does not change. Such a decision is depicted in Figure 3.10.a, where the longest ET busy period has the same value before (w_{max}) and after (w'_{max}) a TT task τ_2 is added to the static schedule at time ϕ_2. The static schedule contains only two tasks τ_1 and τ_2, of equal worst case execution time. An ET task will suffer a maximum of TT interference if it becomes ready at a moment in time when a TT task is scheduled for execution. If we consider that the longest ET busy period w_{max} is associated with the response time of an ET task τ_{max}, then we notice in the figure that the task τ_{max} has the same response time regardless if it becomes ready when τ_1 starts (like at

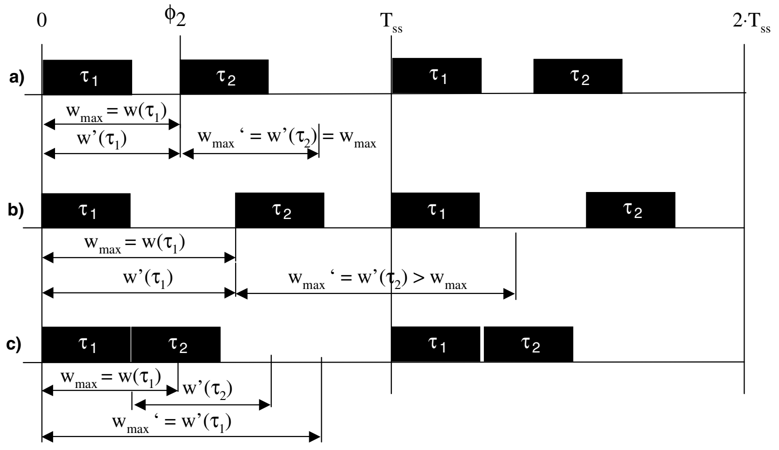


Figure 3.10: Alternative Start Times for a TT Task τ_2 in the Model with Arbitrary Offsets

time 0, with a busy window of $w'(\tau_1)$) or when τ_2 starts (like in the second round of the static schedule, at time $T_{ss}+\phi_2$, when $w'(\tau_2) = w'(\tau_1)$).

In some cases, the value of w_{max} is large enough so that adding another TT task in the static schedule cannot be performed without increasing the TT interference. Such a situation is illustrated in Figure 3.10.b, where w_{max} has a larger value than in Figure 3.10.a. Consequently, task τ_2 is scheduled much later in the schedule than in the previous case. We notice that if τ_{max} becomes ready at the same time task τ_2 starts, then the longest ET busy period will be extended due to additional interference from the TT tasks in the next round of the static schedule. In such situations, when the extra TT interference over ET activities cannot be avoided, the only thing that we can improve is the probability of finding a valid static schedule, and we do this by scheduling task τ_2 as early as possible (as shown in Figure 3.10.c).

When scheduling a TT task earlier, additional care has to be taken if the value w_{max} is larger than the period T_{ss} of the static schedule. In such a situation, we cannot schedule the TT task too early without risking an increase in the interference over ET tasks. Let us see how the same example illustrated in Figure 3.10 modifies if the longest busy period is so large that the increase in interference discussed in Figure 3.10.b cannot be

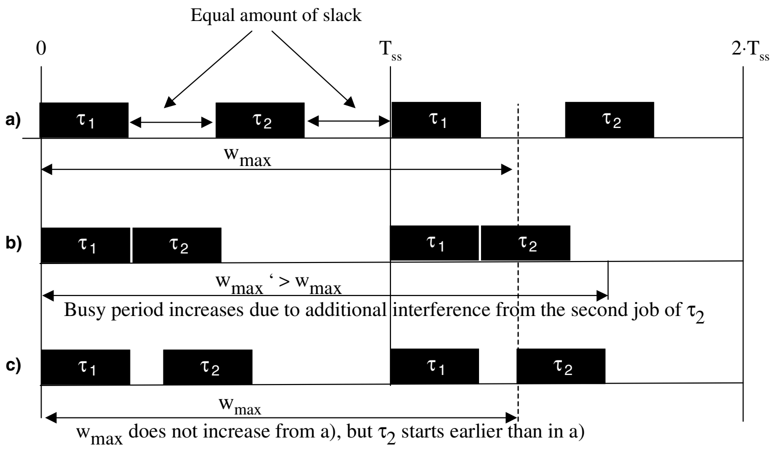


Figure 3.11: Avoiding Unnecessary TT Interference over the Execution of ET Tasks

avoided. We start by scheduling τ_2 somewhere in the middle of the slack available at the end of the static schedule, as shown in Figure 3.11.a. Such a scheduling is intended to help us determine the new value of the longest ET busy window w_{max}, and we discuss now the case when w_{max} is larger than the period T_{ss} of the static schedule. If we try now to schedule task τ_2 as early as possible (at time $\phi_2 = C_1$ as in Figure 3.11.b, where C_1 is the worst-case execution time of τ_1), then we notice that the value w_{max} will increase due to the fact that the ET busy period will contain an additional job of τ_2. However, if we push τ_2 only to the time moment

$$\phi_2 = \phi_1 + w_{max} - \left\lfloor \frac{w_{max}}{T_{ss}} \right\rfloor \cdot T_{ss}, \tag{3.18}$$

as in Figure 3.11.c, then we avoid introducing new TT interference and the response times of ET tasks remain the same as in Figure 3.11.a.

We have modified the holistic scheduling algorithm SLS presented in Figure 3.6 by including all the above observations in the procedure that schedules a TT task (*schedule_TT_task*). The optimised procedure for

```
1  procedure schedule_TT_task($\tau_{ij}$, $\mathcal{M}(\tau_{ij})$)
2      compute largest ET busy period $w_{max}$ on node $\mathcal{M}(\tau_{ij})$
3      $LS$ = determine_largest_TT_start($\mathcal{M}(\tau_{ij})$)
4      schedule $\tau_{ij}$ at $\phi_a = LS + w_{max}$ and compute cost $DSch_a$
5      $LE$ = determine_largest_TT_end($\mathcal{M}(\tau_{ij})$)
6      schedule $\tau_{ij}$ at $\phi'_a = LE$ and compute cost $DSch'_a$
7      schedule $\tau_{ij}$ at $\phi_b$ inthe middle of the last slack on $\mathcal{M}(\tau_{ij})$
8      compute the new value $w'_{max}$ and the cost $DSch_b$
9      $\phi_c = LS + w'_{max} - \lfloor w'_{max}/T_{ss} \rfloor \cdot T_{ss}$
10     schedule $\tau_{ij}$ at $\phi_c$ and compute cost $DSch_c$
11     select $\phi_{ij} \in \{\phi_a, \phi'_a, \phi_b, \phi_c\}$ with the smallest associated $DSch$
12     schedule $\tau_{ij}$ at $\phi_{ij}$
13 end schedule_task
```

Figure 3.12: Optimised TT Task Scheduling

scheduling a TT task τ_{ij} on a node $\mathcal{M}(\tau_{ij})$ can be seen in Figure 3.12. Before we actually schedule τ_{ij}, we compute the largest ET busy period (line 2), and we use it in order to explore the following alternative start times for τ_{ij}:

First, we investigate the case when the busy period is initiated by the latest TT task scheduled on node $\mathcal{M}(\tau_{ij})$ (line 3). We schedule the task τ_{ij} at the determined time ϕ_a and then verify the timing properties of the system by computing its schedulability degree (line 4).

Next, we explore the cases when we accept new TT interference. First, we analyse the case when τ_{ij} is scheduled at the end of the static schedule, immediately after the last TT task on node $\mathcal{M}(\tau_{ij})$ (lines 5-6). In line 7, we then schedule the TT task τ_{ij} in the middle of the slack time available at the end of the static schedule on node $\mathcal{M}(\tau_{ij})$ (in a manner similar to the case illustrated in Figure 3.11.a). Such a schedule will result in a new value w'_{max} for the longest ET busy period (line 8), and we use this value as an input to Equation (3.18), in order to determine an earlier possible start time ϕ_c for τ_{ij} (line 9). We schedule τ_{ij} at ϕ_c and we compute again the schedulability degree of the resulted system. In the end (lines 11-12), the algorithm compares the schedulability degree of the four explored schedules and keeps the one that produces a better cost function.

3.6 Experimental Results

For the evaluation of our scheduling and analysis algorithm we generated a set of 2970 tests representing systems of 2 to 10 nodes. The number of tasks mapped on each node varied between 10 and 30, leading to applications with a number of 20 up to 300 tasks. The tasks were grouped in task-graphs of 5, 10 or 15 tasks. Between 20% and 80% of the total number of tasks were considered as event-triggered and the rest were set as time-triggered. The execution times of the tasks were generated in such a way that the utilisation on each processor was between 20% and 80%. In a similar manner we assured that 20% and up to 60% of the total utilisation on a processor is required by the ET activities. All experiments were run on an AMD Athlon 850MHz PC.

The first set of experiments compares two versions of the holistic scheduling algorithm presented in Figure 3.6:

- the simple list scheduling (SLS), in which TT tasks are scheduled in an ASAP manner;
- the improved list scheduling (ILS), in which TT tasks are scheduled according to the optimised procedure presented in Figure 3.12.

In Figure 3.13.a we illustrate the capacity of the ILS based algorithm to produce schedulable systems, compared to that of SLS. The figure shows that ILS was able to generate between 31-55% more schedulable solutions compared to the case when a simple list scheduling was used.

In addition, we computed the quality of the identified solutions, as the percentage deviation of the schedulability degree ($DSch_{xLS}$) of the ET activities in the resulted system, relative to the schedulability degree of an ideal solution ($DSch_{ref}$) in which the static schedule does not interfere at all with the execution of the ET activities:

$$Interference = \frac{DSch_{ref} - DSch_{xLS}}{DSch_{ref}} \cdot 100 \qquad (3.19)$$

In other words, we used the function *DSch* as a measure of the interference introduced by the TT activities on the execution of ET activities. In

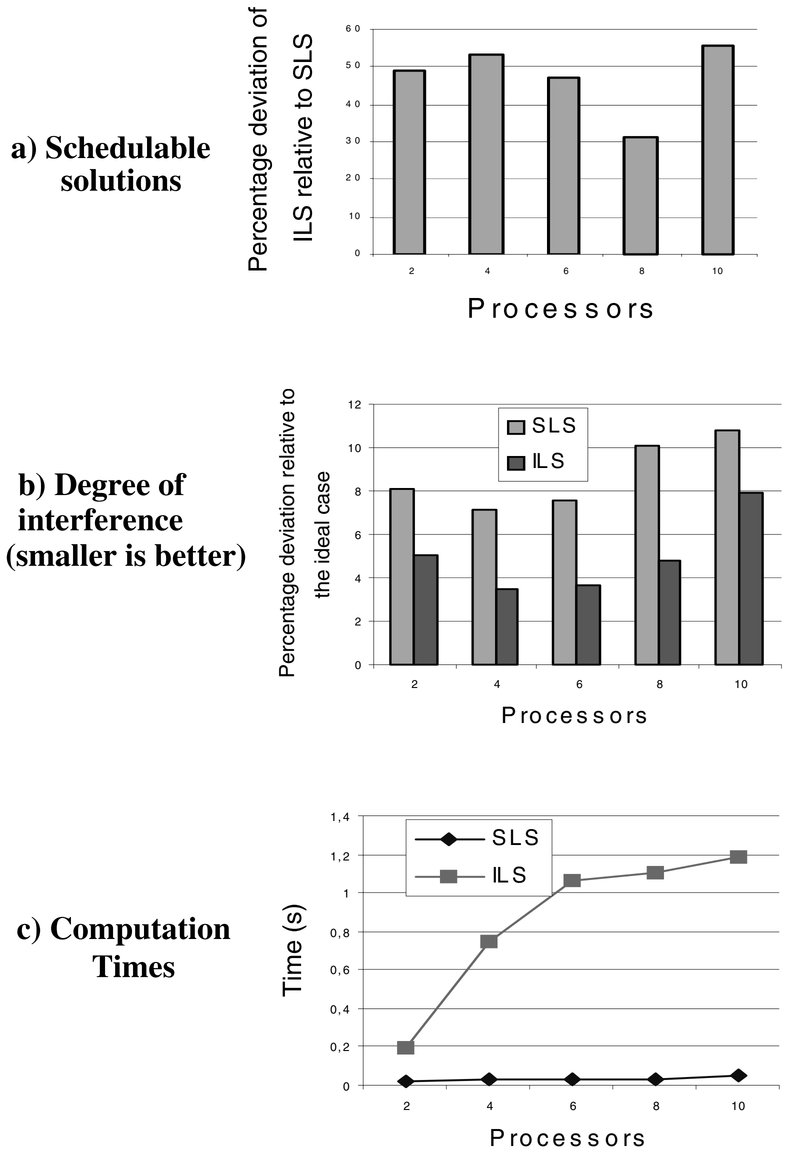


Figure 3.13: Performance of the Scheduling and Schedulability Analysis Algorithm

fig Figure 3.13.b, we present the average quality of the solutions found by the two algorithms. For this diagram, we used only those results where both algorithms managed to find a schedulable solution. It is easy to observe that the solutions obtained with ILS are constantly at a minimal level of interference, while the SLS heuristic produces solutions in which the TT interference is considerably higher, resulting in significantly larger response times of the ET activities.

In Figure 3.13.c we present the average execution times of our scheduling heuristic and compare them with the execution times of the simple list scheduling algorithm. According to expectations, the execution time for our ILS scheduling and schedulability analysis algorithm increases with the size of the application. However, even for large applications, the algorithm is still fast enough so that it could be efficiently used inside a design space exploration loop with an extremely large number of iterations (see Chapter 4).

Chapter 4
Design Optimisation

NOW THAT WE ARE ABLE to derive the schedulability degree of a heterogeneous TT/ET system that uses mixed scheduling policies, we consider in this chapter a larger design context, which involves mapping, scheduling and a couple of specific optimisation aspects which are characteristic for this type of systems. In particular, we are interested in the following issues:

- assignment of scheduling policies to tasks;
- mapping of tasks to the nodes of the architecture;
- optimisation of the access to the communication infrastructure;
- scheduling of tasks and messages (that has been already discussed in the previous chapter).

The goal is to produce an implementation that, using a given amount of resources, meets all the timing constraints of the application.

In this chapter, by scheduling policy assignment (SPA) we denote the decision whether a certain task should be scheduled with SCS, FPS or EDF. Mapping a task means assigning it to a particular hardware node. During the optimisation of the bus access we concentrate on finding that set of configuration parameters for the communication protocol that globally improves the system responsiveness.

4.1 Specific Design Optimisation Problems

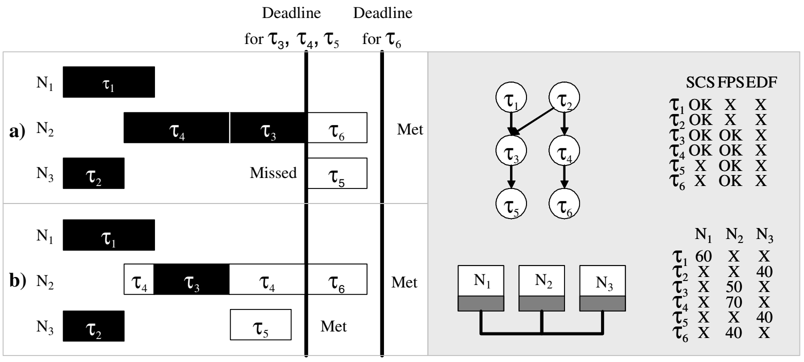
4.1.1 SCHEDULING POLICY ASSIGNMENT

Very often, the SPA and mapping decisions are taken based on the experience and preferences of the designer, considering aspects like the functionality implemented by the task, the hardness of the constraints, sensitivity to jitter, etc. Moreover, due to legacy constraints, the mapping and scheduling policy of certain processes might be fixed.

Thus, we denote with $\mathcal{P}_{SCS} \subseteq \mathcal{P}$ the subset of tasks for which the designer has assigned SCS, $\mathcal{P}_{FPS} \subseteq \mathcal{P}$ contains tasks to which FPS is assigned, while $\mathcal{P}_{EDF} \subseteq \mathcal{P}$ contains those tasks for which the designer has decided to use the EDF scheduling policy. There are tasks, however, which do not exhibit certain particular features or requirements which obviously lead to their scheduling as SCS, FPS or EDF activities. The subset $\mathcal{P}^{+} = \mathcal{P} \backslash (\mathcal{P}_{SCS} \cup \mathcal{P}_{FPS} \cup \mathcal{P}_{EDF})$ of tasks could be assigned any scheduling policy. Decisions concerning the SPA to this set of activities can lead to various trade-offs concerning, for example, the schedulability properties of the system, the size of the schedule tables, the utilization of resources, etc.

Let us illustrate some of the issues related to SPA in such a context. In the example presented in Figure 4.1 we have an application[1] with six tasks, τ_1 to τ_6, and three nodes, N_1, N_2 and N_3. The worst-case execution times on each node are given in the table labelled "Mapping". Note that an "x" in the table means that the task is not allowed to be mapped on that node (the mapping of tasks is thus fixed for this example). The scheduling policy assignment is captured by the table labelled "SPA". Thus, tasks τ_1 and τ_2 are scheduled using SCS, while tasks τ_5 and τ_6 are scheduled with FPS. Similarly, an "x" in the table means that the task cannot be scheduled with the corresponding scheduling policy. We have to decide which scheduling policy to use for tasks τ_3 and τ_4, which can be scheduled with either of the SCS or FPS scheduling policies.

1.Communications are ignored for the examples in this subsection.



	SCS	FPS	EDF
τ_1	OK	X	X
τ_2	OK	X	X
τ_3	OK	OK	X
τ_4	OK	OK	X
τ_5	X	OK	X
τ_6	X	OK	X

	N_1	N_2	N_3
τ_1	60	X	X
τ_2	X	X	40
τ_3	X	50	X
τ_4	X	70	X
τ_5	X	X	40
τ_6	X	40	X

Figure 4.1: Scheduling Policy Assignment Example #1

We can observe that the scheduling of τ_3 and τ_4 have a strong impact on their successors, τ_5 and τ_6, respectively. Thus, we would like to schedule τ_4 such that not only τ_3 can start on time, but τ_4 also starts soon enough to allow τ_6 to meet its deadline. As we can see from Figure 4.1.a, this is impossible to achieve by scheduling τ_3 and τ_4 with SCS. Although τ_3 meets its deadline, it finishes too late for τ_5 to finish on deadline. However, if we schedule τ_4 with FPS, for example, as in Figure 4.1.b, both deadlines are met. In this case, τ_3 finishes on time to allow τ_5 to meet its deadline. Moreover, although τ_4 is preempted by τ_3, it still finishes on time, meets its deadline, and allows τ_6 to meet its deadline, as well. Note that using EDF for τ_4 (if it would share the same priority level with τ_6, for example) will also meet the deadline. The idea in this example is to allow preemption for τ_4.

For a given set of preemptable tasks, the example in Figure 4.2 illustrates the optimisation of the assignment of FPS and EDF policies. In Figure 4.2 we have an application composed of four tasks running on two nodes. Tasks τ_1, τ_2 and τ_3 are mapped on node N_1, while task τ_4 runs on N_2. Tasks τ_2 and τ_3 have the same priority, while task τ_4 is data dependent of task τ_1. All tasks in the system have the same worst case-execution times (20 ms), deadlines (60 ms) and periods (80 ms). Tasks τ_2 and τ_3 are

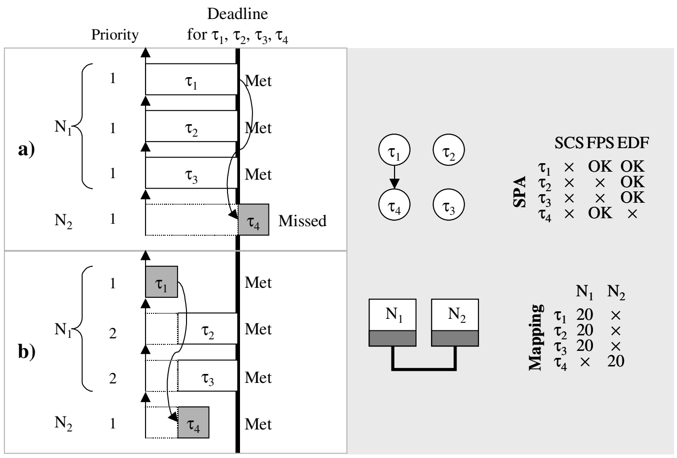


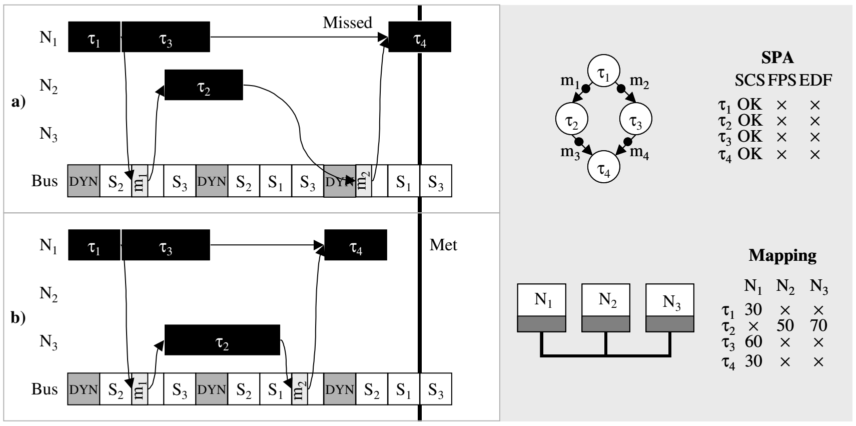
Figure 4.2: Scheduling Policy Assignment Example #2

scheduled with EDF, τ_4 with FPS, and we have to decide the scheduling policy for τ_1, between EDF and FPS.

If τ_1 is scheduled according to EDF, thus sharing the same priority level "1" with the tasks on node N_1, then task τ_4, in the worst case, misses its deadline (Figure 4.2.a). Note that in the time line for node N_1 in Figure 4.2 we depict several worst-case scenarios: each EDF task on node N_1 is depicted considering the worst-case interference from the other EDF tasks on N_1. However, the situation changes if on node N_1 we use FPS for τ_1 (i.e., changing the priority levels of τ_2 and τ_3 from "1" to "2"). Figure 4.2.b shows the response times when task τ_1 has the highest priority on N_1 (τ_1 retains priority "1") and the other tasks are running under EDF at a lower priority level (τ_2 and τ_3 share lower priority "2"). Because in this situation there is no interference from tasks τ_2 and τ_3, the worst-case response time for task τ_1 decreases considerably, allowing task τ_4 to finish before its deadline, so that the system becomes schedulable.

4.1.2 Mapping

The designer might have already decided the mapping for a part of the tasks. For example, certain tasks, due to constraints such as having to be close to sensors/actuators, have to be physically located in a particular



SPA	SCS	FPS	EDF
τ_1	OK	×	×
τ_2	OK	×	×
τ_3	OK	×	×
τ_4	OK	×	×

Mapping	N_1	N_2	N_3
τ_1	30	×	×
τ_2	×	50	70
τ_3	60	×	×
τ_4	30	×	×

Figure 4.3: Mapping Example

hardware unit. They represent the set $\mathcal{P}^M \subseteq \mathcal{P}$ of already mapped tasks. Consequently, we denote with $\mathcal{P}^* = \mathcal{P} \setminus \mathcal{P}^M$ the tasks for which the mapping has not yet been decided.

For a distributed heterogeneous system, the communication infrastructure has an important impact on the design and, in particular, on the mapping decisions. Let us consider the example in Figure 4.3 where we have an application consisting of four tasks, τ_1 to τ_4, and an architecture with three nodes, N_1 to N_3. Thus, the bus will have three static slots, S_1 to S_3 for each node, respectively. The sequence of slots on the bus is S_2 followed by S_1 and then S_3. We have decided to place a single dynamic phase within a bus cycle, labelled "DYN" and depicted in gray, preceding the three static slots (see Section 2.2 for the details about the bus protocol). We assume that τ_1, τ_3 and τ_4 are mapped on node N_1, and we are interested to map task τ_2. Task τ_2 is allowed to be mapped on node N_2 or on node N_3, and its execution times are depicted in the table labelled "mapping". Moreover, the scheduling policy is fixed for each task, such that all tasks are scheduled with SCS.

In order to meet the deadline, one would map τ_2 on the node it executes fastest, i.e., node N_2 see Figure 4.3.a. However, in spite of the shorter execution time of the task τ_2, the response time for τ_3 is extended beyond the task deadline due to the long communication delay. One can observe in the

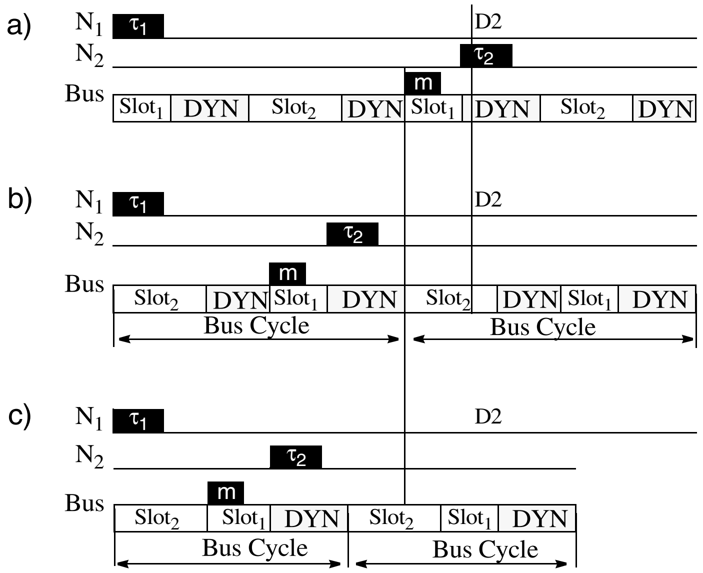


Figure 4.4: Optimization of Bus Access Cycle

Figure 4.3.a that the first slot where node N_2 is able to send message m_2 appears only in the third bus cycle. The application will meet the deadline only if τ_2 is, counter-intuitively, mapped on the slower node, i.e., node N_3, as depicted in Figure 4.3.b: in this situation, message m_2 will be sent in slot S_3 during the second bus cycle, leading to a shorter communication delay and to a response time for τ_3 that is small enough in order to meet the deadline imposed on that task.

4.1.3 Bus Access Optimisation

The configuration of the bus access cycle has a strong impact on the global performance of the system. The parameters of this cycle have to be optimised such that they fit the particular application and the timing requirements. Parameters to be optimised are the number of static and dynamic phases during a communication cycle, as well as the length and order of these phases. Considering the static phases, parameters to be fixed are the order, number, and length of slots assigned to the different nodes. Let us denote such a bus configuration with $\mathcal{B}$.

For example, consider the situation in Figure 4.4, where task τ_1 is mapped on node N_1 and sends a message m to task τ_2 which is mapped on node N_2. In case a), task τ_1 misses the start of the ST $Slot_1$ and, therefore, message m will be sent during the next bus cycle, causing the receiver task τ_2 to miss its deadline D_2. In case b), the order of ST slots inside the bus cycle is changed, the message m will be transmitted earlier and τ_2 will meet its deadline. The resulted situation can be further improved, as it can be seen in Figure 4.4.c), where task τ_2 finishes even earlier, if the first DYN phase in the bus cycle can be eliminated without producing intolerable delays of the DYN messages (which have been ignored in this example).

4.2 Exact Problem Formulation

As an input we have an application $\mathcal{A}$ given as a set of task graphs (Section 2.4) and a system architecture consisting of a set $\mathcal{N}$ of nodes (Section 2.1). As introduced previously, $\mathcal{T}_{SCS}$, $\mathcal{T}_{FPS}$ and $\mathcal{T}_{EDF}$ are the sets of tasks for which the designer has already assigned SCS, FPS or EDF scheduling policy, respectively. Also, $\mathcal{T}^M$ is the set of already mapped tasks.

As part of our problem, we are interested to:

- find a scheduling policy assignment $\mathcal{S}$ for tasks in $\mathcal{T}^+ = \mathcal{T} \setminus (\mathcal{T}_{SCS} \cup \mathcal{T}_{FPS} \cup \mathcal{T}_{EDF})$;
- decide a mapping for tasks in $\mathcal{T}^* = \mathcal{T} \setminus \mathcal{T}^M$;
- determine a bus configuration $\mathcal{B}$;
- determine the schedule table for the SCS tasks and priorities of FPS and EDF tasks;

such that imposed deadlines are guaranteed to be satisfied.

```
OptimisationStrategy($\mathcal{A}$)
1  Step 1: $\mathcal{B}^0$ = InitialBusAccess($\mathcal{A}$)
2          ($\mathcal{M}^0$, $\mathcal{S}^0$) = InitialMSPA($\mathcal{A}$, $\mathcal{B}^0$)
3          if HolisticScheduling($\mathcal{A}$, $\mathcal{M}^0$, $\mathcal{B}^0$, $\mathcal{S}^0$) returns schedulable then stop end if
4  Step 2: ($\mathcal{M}$, $\mathcal{S}$, $\mathcal{B}$) = MSPAHeuristic($\mathcal{A}$, $\mathcal{M}^0$, $\mathcal{B}^0$)
5          if HolisticScheduling($\mathcal{A}$, $\mathcal{M}$, $\mathcal{S}$, $\mathcal{B}$) returns schedulable then stop end if
6  Step 3: $\mathcal{B}$ = BusAccessOptimisation($\mathcal{A}$, $\mathcal{M}$, $\mathcal{S}$, $\mathcal{B}$)
7          HolisticScheduling($\mathcal{A}$, $\mathcal{M}$, $\mathcal{B}$, $\mathcal{S}$)
end OptimisationStrategy
```

Figure 4.5: The General Strategy

4.3 Design Optimisation Strategy

The design problem formulated in the previous section is NP-complete (the scheduling sub-problem, in a simpler context, is already NP-complete [Ull75]). Therefore, our strategy, outlined in Figure 4.5, is to divide the problem into several, more manageable, sub-problems. Our OptimisationStrategy has three steps:

1. In the first step (lines 1–3) we decide on an initial bus access configuration $\mathcal{B}^0$(function InitialBusAccess), and an initial policy assignment $\mathcal{S}^0$ and mapping $\mathcal{M}^0$(function InitialMSPA). The initial bus access configuration, scheduling policy assignment and mapping algorithm (lines 1-2 in Figure 4.5) are presented in Section 4.3.1. Once an initial mapping, scheduling policy assignment and bus configuration are obtained, the application is scheduled using the HolisticScheduling algorithm (line 3) described in Section 3.5, Figure 3.6.
2. If the application is schedulable, the optimisation strategy stops. Otherwise, it continues with the second step by using an iterative improvement mapping and policy assignment heuristic, MSPAHeuristic (line 4), presented in Section 4.3.2, to improve the partitioning and mapping obtained in the first step.
3. If the application is still not schedulable, we use, in the third step, the algorithm BusAccessOptimisation, which optimises the access to the communication infrastructure (line 6). If the application is still unschedulable, we conclude that no satisfactory implementation could be

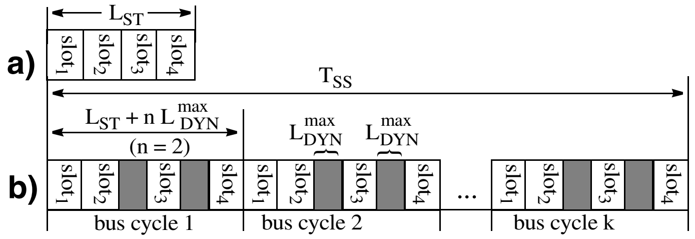


Figure 4.6: Initial Bus Configuration

found with the available amount of resources.

4.3.1 Building An Initial Configuration

The first step starts with generating a mapping and partitioning of the tasks, as well as a bus cycle (lines 1-2 in Figure 4.5). Such an initial system configuration is based on a very simple and fast heuristic that relies on the following strategies:

- The scheduling policy assignment is performed with the only constraint to evenly distribute the load between the SCS and the FPS/EDF domains.
- The mapping is based on a very fast heuristic aimed at minimising inter-processor communication while keeping a balanced processor load.
- The initial bus cycle is constructed in the following two steps:

1. We consider that each node can transmit messages during only one ST slot inside a bus cycle. The ST slots are assigned in order to the nodes such that $Node_i$ transmits during $Slot_i$ (Figure 2.1). The length of $Slot_i$ is set to a value which is equal to the length of the largest ST message generated by a task mapped on $Node_i$. Considering an architecture of 4 nodes, a structure like the one in Figure 4.6.(a) is produced after this step.
2. Dynamic phases are introduced in order to generate a mixed ST/DYN bus cycle. We start from the rough assumption that the total length of the dynamic phases over a period T_{SS} (T_{SS} is the length of the static

schedule, see 3.5) is equal to the total length of the DYN messages transmitted over the same period, which is:

$$\sum_{m_i \in DYNdomain} \frac{T_{SS}}{T_i} \cdot L_i \qquad (4.1)$$

where T_i and L_i are the period and the length (expressed in time units) of the DYN message m_i. We set the length of each DYN phase to the length L_{DYN}^{max} of the largest DYN message. The number n of dynamic phases in each cycle can be determined from the following equation:

$$\frac{T_{SS}}{L_{ST} + n \cdot L_{DYN}^{max}} \cdot n \cdot L_{DYN}^{max} = \sum_{m_i \in DYNdomain} \frac{T_{SS}}{T_i} \cdot L_i \qquad (4.2)$$

where L_{ST} is the total length of the static slots in a bus cycle and $L_{ST} + n \cdot L_{DYN}^{max}$ is the length of the bus cycle. Finally, the dynamic phases are evenly distributed inside the bus cycle. Figure 4.6.b illustrates such an initial bus configuration.

4.3.2 MAPPING AND SCHEDULING POLICY ASSIGNMENT HEURISTIC

In Step 2 of our optimisation strategy (Figure 4.5), the following design transformations are performed with the goal to produce a schedulable system implementation:

- change the scheduling policy of a task;
- change the mapping of a task;
- change the priority level of a FPS of EDF task.

Our optimisation algorithm is presented in Figure 4.7 and it implements a greedy approach in which every task in the system is iteratively mapped on each node (line 4) and assigned to each scheduling policy (line 8), under the constraints imposed by the designer. The next step involves adjustments to the bus access cycle (line 10), which are needed for the case when the bus cycle configuration cannot handle the minimum requirements of the current inter-node communication. Such adjustments are mainly based on enlargement of the static slots or dynamic phases in the bus cycle, and are required in the case the bus has to support larger messages than before. New messages may appear on the bus due to, for example, re-mapping of tasks; consequently, there may be new ST messages that are larger than the current static slot for the sender node (or similarly the bus will face the situation where new DYN messages are larger than the largest DYN phase in the bus cycle).

```
MSPAHeuristic($\mathcal{A}$, $\mathcal{M}$, $\mathcal{B}$, $\mathcal{S}$)
 1  for each activity $\tau_{ij}$ in the system do
 2     for each processor $N_i \in \mathcal{N}$ in the system do
 3        if $\tau_{ij}$ in $\mathcal{P}^*$ then -- can be remapped
 4           $\mathcal{M}(\tau_{ij}) = N_i$
 5        end if
 6        for policy = SCS, FPS do
 7           if $\tau_{ij}$ in $\mathcal{P}^+$ then -- the scheduling policy can be changed
 8              $\mathcal{S}(\tau_{ij})$ = policy
 9           end if
 10          adjust bus cycle($\mathcal{A}$, $\mathcal{M}$, $\mathcal{B}$, $\mathcal{S}$)
 11          recompute FPS priority levels
 12          for all FPS tasks $\tau_{ab}$ sharing identical priority levels do
 13             $\mathcal{S}(\tau_{ab})$ = EDF
 14          end for
 15          HolisticScheduling($\mathcal{A}$, $\mathcal{M}$, $\mathcal{B}$, $\mathcal{S}$)
 16          if $\delta_{\mathcal{A}} <$ best_$\delta_{\mathcal{A}}$ then
 17             best_policy$_{ij}$ = $\mathcal{S}(\tau_{ij})$; best_processor$_{ij}$ = $\mathcal{M}(\tau_{ij})$
 18             best_$\delta_{\mathcal{A}} = \delta_{\mathcal{A}}$
 19          end if
 20          if $\delta_{\mathcal{A}} < 0$ then
 21             return best ($\mathcal{M}$, $\mathcal{B}$, $\mathcal{S}$)
 22          end if
 23       end for
 24    end for
 25 end for
end MSPAHeuristic
```

Figure 4.7: Policy Assignment and Mapping

Such an adjustment of the bus access cycle is illustrated in Figure 4.8, where 4 TT tasks are mapped on 3 nodes (N_1, N_2 and N_3). The number at the side of each message represents its length. Tasks mapped on different nodes communicate through ST messages and an ST slot should be able to accommodate the longest message transmitted by the associated node. The figure shows how the lengths of the slots associated with N_1 and N_2 are modified after a task has been re-mapped. In one case, task τ_2 is moved from N_2 to $N_1$and therefore, the message $m_{1,2}$ will disappear (τ_1 and τ_2 are both mapped on N_1), while message $m_{2,4}$ will be transmitted in $Slot_1$ instead of $Slot_2$. In the second case, τ_3 is moved from N_2 to N_1, which means that $m_{1,3}$ disappears, while $m_{3,4}$ is transmitted in $Slot_1$.

Before the system is analysed for its timing properties, our heuristic also tries to optimise the priority assignment of tasks running under FPS (line 11). The state of the art approach for such a task is the HOPA algorithm for assigning priority levels to tasks in multiprocessor systems [Gut95]. However, due to the fact that HOPA was computationally expensive to be run inside our design optimisation loop, we use a scaled down greedy algorithm, in which we drastically reduce the number of iterations needed for determining an optimised priority assignment.

Finally, the resulted system configuration is analysed (line 15) using the scheduling and schedulability analysis algorithm presented in Section 3.5,

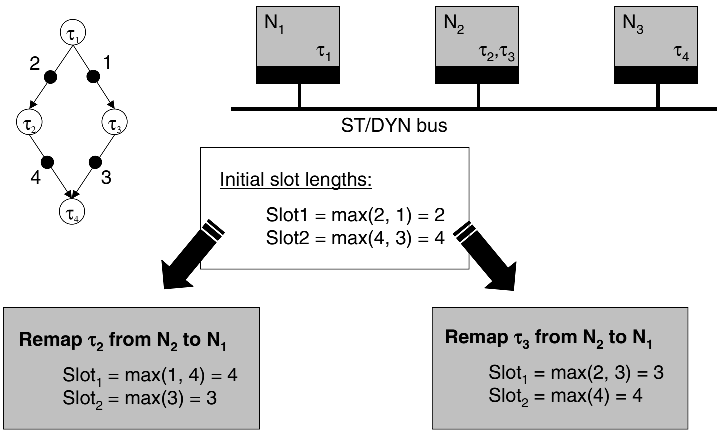


Figure 4.8: Adjustment of the Bus Access Cycle

Figure 3.6. The resulted cost function will decide whether the current configuration is better than the current best one (lines 16–19). Moreover, if all activities meet their deadlines (*DSch* < 0), the optimisation heuristic stops the exploration process and returns the current best-so-far configuration (lines 20-22).

4.3.3 BUS ACCESS OPTIMISATION

It may be the case that even after the mapping and partitioning step, some ET activities are still not schedulable. In the third step (line 6, Figure 4.5), our algorithm tries to remedy this problem by changing the parameters of the bus cycle, like ST slot lengths and order, as well as the number, length and order of the ST and DYN phases. The goal is to generate a bus access scheme which is better adapted to the particular task configuration. The heuristic is illustrated in Figure 4.9. The algorithm iteratively looks for the right place and size of $Slot_i$ used for transmission of ST messages from $Node_i$ (outermost loops). The position of $Slot_i$ is swapped with all the positions of slots of higher order (line 03). Also, all alternative lengths (lines 04-05) of $Slot_i$ larger than its minimal allowed length (which is equal to

```
1  for i = 1 to NrNodes
2     for j = i to NrNodes
3        swap $Slot_i$ with $Slot_j$
4        for all slot lengths $\lambda$ > min_len($Slot_i$)
5           len($Slot_i$) = $\lambda$
6           for all DYN phase lengths $\pi$
7              len($Ph_i$) = $\pi$
8              if DSch $\leq$ 0 then stop endif
9              keep solution with lowest DSch
10          end for
11       end for
12       swap back $Slot_i$ and $Slot_j$
13    end for
14    bind best position and length of $Slot_i$
15    bind length of $Ph_i$
16 end for
```

Figure 4.9: Bus Access Optimisation

the length of the largest ST message generated by a task mapped on $Node_i$) are considered. For any particular length and position of $Slot_i$, alternative lengths of the adjacent ET phase Ph_i are considered (innermost loop). For each alternative, the schedulability analysis evaluates cost *DSch*, and the solution with the lowest cost is selected. If $DSch \leq 0$, the system is schedulable and the heuristic is stopped.

It is important to notice that the possible length π of an ET phase (line 06) includes also the value 0. Therefore, in the final bus cycle, it is not needed that each static slot is followed by a dynamic phase (see also Figure 2.1). Dynamic phases introduced as result of the previous steps can be eliminated by setting the length to $\pi = 0$ (such a transformation is illustrated in Figure 4.4.c). It should be also mentioned that enlarging a slot/phase can increase the schedulability by allowing several ST/DYN messages to be transmitted quickly immediately one after another. At the same time, the following slots are delayed, which means that ST messages transmitted by nodes assigned to upcoming slots will arrive later. Therefore, the optimal schedulability will be obtained for slot and phase lengths which are not tending towards the maximum. The number of alternative slot and phase lengths to be considered by the heuristic in Figure 4.9 is limited by the following two factors:

1. The maximum length of a static slot or dynamic phase is fixed by the technology (e.g. 32 or 64 bits).
2. Only frames consisting of entire messages can be transmitted, which excludes several alternatives.

4.4 Experimental Results

For the evaluation of our design optimisation heuristic we have used synthetic applications as well as a real-life example consisting of a vehicle cruise controller.

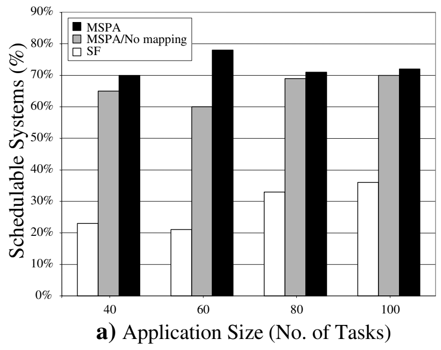
We have randomly generated applications of 40, 60, 80 and 100 tasks on systems with 4 processors. 56 applications were generated for each dimension, thus a total of 224 applications were used for experimental evaluation. An equal number of applications with processor utilisation of 20%, 40%, 60% and 80% were generated for each application dimension. All experiments were run on an AMD AthlonXP 2400+ processor, with 512 MB RAM.

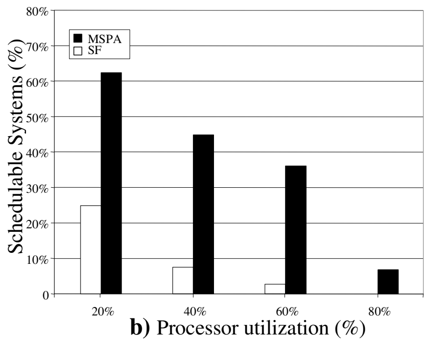
We were first interested to determine the quality of our design optimisation approach for hierarchically scheduled systems, the MSPAHeuristic (MSPA, see Figure 4.7). We have compared the percentage of schedulable implementations found by MSPA with the number of schedulable solutions obtained by the InitialMSPA algorithm described in Section 4.3 (see Figure 4.5, line 2), which derives a straight-forward system implementation, denoted with SF. The results are depicted in Figure 4.10.a. We can see that our MSPA heuristic (the black bars) performs very well, and finds a number of schedulable systems that is considerably and consistently higher than the number of schedulable systems obtained with the SF approach (the white bars). On average, MSPA finds 44.5% more schedulable solutions than SF.

Second, we were interested to determine the impact of the scheduling policy assignment (SPA) decisions on the number of schedulable applications obtained. Thus, for the same applications, we considered that the task mapping is fixed by the SF approach, and only the SPA is optimised. Figure 4.10.a presents this approach, labelled "MSPA/No mapping", corresponding to the gray bars. We can see that most of the improvement over the SF approach is obtained by carefully optimising the SPA in our MSPA heuristic.

We were also interested to find out what is the impact of the processor utilization of an application on the quality of the implementations produced by our optimisation heuristic. Figure 4.10.b presents the percentage of schedulable solutions found by MSPA and SF as we ranged the utilization from 20% to 80%. We can see that SF degrades very quickly with the increased utilization, with under 10% schedulable solutions for applications with 40% utilization and without finding any schedulable solution for applications with 80% utilization, while MSPA is able to find a significant number of schedulable solutions even for high processor utilisation.

In Figure 4.10.c we show the average run times obtained by applying our MSPA heuristic on the examples presented in Figure 4.10.a. The upper curve illustrates the average execution times for those applications which were not found schedulable by our heuristic. This curve can be considered





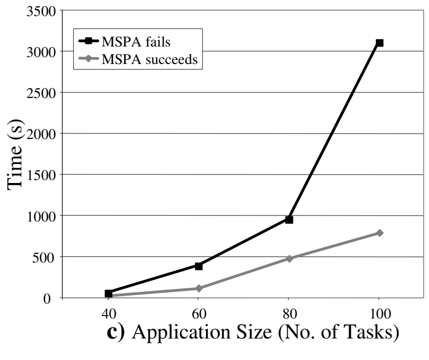


Figure 4.10: Performance of the Design Optimisation Heuristic

as an upper bound for the computation time of our algorithm. For the examples that were found schedulable, our heuristic stops the exploration process earlier, thus leading to smaller computation times, as shown in the lower curve in Figure 4.10.c. We can see that, considering the complex optimisation steps performed, our design optimisation heuristic produces good quality results in a reasonable amount of time (for example, the heuristic will finish on average in less than 500 seconds for applications with 80 tasks that were found schedulable).

In the case when Step 2 of the general design optimisation strategy (line 4 in Figure 4.5) does not yield a system that meets its deadlines, an additional bus access optimisation is performed at the end of the MSPA. In our experiments we have found that after such a bus access optimisation (Step 3 in Figure 4.5) the number of schedulable applications found will increase with 4-6%. While such an increase may seem insignificant, we have to take into consideration that the second step of the general strategy is already aggressive enough in finding schedulable systems. The full power of the bus access optimisation will be more visible in Chapter 7, where we exclusively investigate the influence of the bus access structure on the schedulability degree of the system.

Finally, we have considered the real-life example implementing a vehicle cruise controller described in Appendix C. For this example, SF failed to produce a schedulable implementation. We applied our design optimisation heuristic MSPA first in a context in which the mapping is fixed by SF, and we only allowed the reassignment of scheduling policies. After 29.5 seconds, the best scheduling policy allocation that was found still resulted in an unschedulable system, but with a "degree of schedulability" three times higher than the one obtained by SF. When optimisation of task mapping was also allowed, MSPA succeeded in finding a schedulable system configuration after 28.49 seconds.

Chapter 5 The FlexRay Communication Protocol

FlexRay is a communication protocol heavily promoted by a large group of car manufacturers and automotive electronics suppliers. However, before it can be successfully used for safety-critical applications that require predictability, timing analysis techniques are necessary for providing bounds for the message communication times.

In this part of the thesis we present an approach to timing analysis of applications communicating over a FlexRay bus, taking into consideration the specific aspects of this protocol, including the DYN segment. More exactly, we propose techniques for determining the timing properties of messages transmitted in the static and the dynamic segments of a FlexRay communication cycle. We first develop a worst-case response time analysis for ET messages sent using the DYN segment, thus providing predictability for messages transmitted in this segment. The analysis techniques for messages are integrated in the context of a holistic schedulability analysis algorithm that computes the worst-case response times of all the tasks and messages in the system.

Such an analysis, while being able to bound the message transmission times on both the ST and DYN segments, represents the first step towards enabling the use of this protocol in a systematic way for time critical applications. The second step towards an efficient use of FlexRay is taken in the following chapter, where we propose several optimisation techniques that consider the particular features of an application during the process of finding a FlexRay bus configuration that can guarantee that all time constraints are satisfied.

The second part of the thesis is organised as follows. The remaining part of the current chapter presents the FlexRay media access control. In Chapter 6, we present our timing analysis for distributed real-time systems that use the FlexRay protocol, together with the experimental results we have run in order to determine the efficiency of our approaches. Chapter 7 shows how system schedulability is improved as a result of careful bus access optimisation. We will present and evaluate three optimisation algorithms that can be used to improve the schedulability of a system that uses FlexRay. We will evaluate the proposed analysis and optimisation techniques using extensive experiments.

5.1 The Media Access Control for FlexRay

In this section we will describe how messages generated by the CPU reach the communication controller and how they are transmitted on the bus. Let us consider the example in Figure 5.1 where we have an architecture consisting of three nodes, N_1 to N_3 sending messages m_a, m_b,... m_h using a FlexRay bus.

Figure 5.1 depicts the main components of a node in a FlexRay-based system:

- the *communication controller* that connects the node to the FlexRay bus
- the *host*, that contains a CPU and local memories
- the *controller-host-interface* (CHI) which is mainly represented in the figure as a set of buffers used for sending data between the host and the communication controller

In FlexRay, the communication takes place in periodic cycles (Figure 5.1.b depicts two cycles of length T_{bus}). Each cycle contains two time intervals with different bus access policies: an ST segment and a DYN segment[1]. The ST and DYN segment lengths can differ, but are fixed over the cycles. We denote with ST_{bus} and DYN_{bus} the length of these segments. Both the ST and DYN segments are composed of several slots. In the ST segment, the slots number is fixed, and the slots have constant and equal length, regardless of whether ST messages are sent or not over the bus in that cycle. The length of an ST slot is specified by the FlexRay global configuration parameter *gdStaticSlot* [Fle07]. In Figure 5.1 there are three static slots for the ST segment.

The length of the DYN segment is specified in number of "minislots", and is equal to *gNumberOfMinislots.* Thus, during the DYN segment, if no message is to be sent during a certain slot, then that slot will have a very small length (equal to the length *gdMinislot* of a so called minislot), otherwise the DYN slot will have a length equal with the number of minislots needed for transmitting the whole message [Fle07]. This can be seen in Figure 5.1.b, where DYN slot 2 has 3 minislots (4, 5, and 6) in the first bus cycle, when message m_e is transmitted, and one minislot (denoted with "MS" and corresponding to the minislot counter 2) in the second bus cycle when no message is sent.

During any slot (ST or DYN), only one node is allowed to send a frame on the bus, and that is the node which holds the message with the frame identifier (*FrameID*) equal to the current value of the slot counter. There are two slot counters, corresponding to the ST and DYN segments, respectively. The assignment of frame identifiers to nodes is static and decided off-line, during the design phase. Each node that sends messages has one or more ST and/or DYN slots associated to it. The bus conflicts are solved by allocating off-line one slot to at most one node, thus making it impossible for two nodes to send during the same ST or DYN slot.

In Figure 5.1, node N_1 has been allocated ST slot 2 and DYN slot 3, N_2 transmits through ST slots 1 and 3 and DYN slots 2 and 4, while node N_3 has DYN slots 1 and 5. For each of these slots, the CHI reserves a buffer that can be written by the CPU and read by the communication controller

1. The FlexRay bus cycle contains also a *symbol window* and a *network idle time*, but their size does not affect the equations in our analysis. For simplicity, they will be ignored during the examples throughout the thesis.

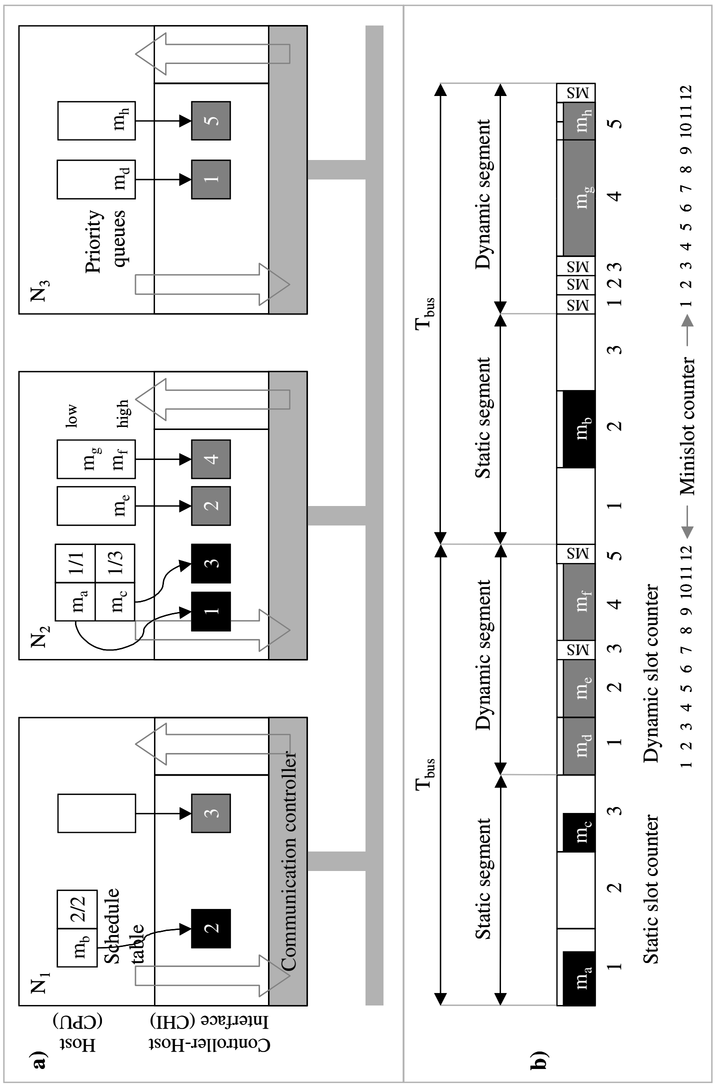


Figure 5.1: FlexRay Communication Cycle Example

(these buffers are read by the communication controller *at the beginning* of each slot, in order to prepare the transmission of frames). The associated buffers in the CHI are depicted in Figure 5.1.a. We denote with $DYNSlots_{N_p}$ the number of dynamic slots associated to a node N_p (this means that for N_2 in Figure 5.1, $DYNSlots_{N_2}$ has value 2).

We use different approaches for ST and DYN messages to decide which messages are transmitted during the allocated slots. For ST messages, we consider that the CPU in each node holds a schedule table with the transmission times. Each transmission time is expressed in the table as a pair of numbers representing the ST slot and the number of the bus cycle inside the hyper-period that is covered by the static cyclic schedule. When the time comes for an ST message to be transmitted, the CPU will place that message in its associated ST buffer of the CHI. For example, ST message m_b sent from node N_1 has an entry "2/2" in the schedule table specifying that it should be sent in the second ST slot of the second bus cycle.

For the DYN messages, the designer specifies their *FrameID*. For example, DYN message m_e has the frame identifier "2". We assume that there can be several messages sharing the same DYN *FrameID*[1]. For example, messages m_g and m_f have both *FrameID* 4. If two messages with the same frame identifier are ready to be sent in the same bus cycle, a priority scheme is used to decide which message will be sent first. Each DYN message m_i has associated a priority $priority_{m_i}$. Messages with the same *FrameID* will be placed in an output queue ordered based on their priorities. The message form the head of the priority queue is sent in the current bus cycle. For example, message m_f will be sent before m_g because it has a higher priority.

At the beginning of each communication cycle, the communication controller of a node resets the slot and minislot counters. At the beginning of each communication slot, the controller verifies if there are messages ready for transmission (present in the CHI send buffers) and packs them

1. If messages are not sharing FrameIDs, this is handled implicitly as a particular case of our analysis.

into frames[1]. In the example in Figure 5.1 we assume that all messages are ready for transmission before the first bus cycle.

Messages selected and packed into ST frames will be transmitted during the bus cycle that is about to start according to the schedule table. For example, in Figure 5.1, messages m_a and m_c are placed into the associated ST buffers in the CHI in order to be transmitted in the first bus cycle. However, messages selected and packed into DYN frames will be transmitted during the DYN segment of the bus cycle only if there is enough time until the end of the DYN segment. Such a situation is verified by comparing if, in the moment the DYN slot counter reaches the value of the *FrameID* for that message, the value of the minislot counter is smaller than a given value *pLatestTx*. The value *pLatestTx* is fixed for each node during the design phase, depending on the size of the largest DYN frame that node will have to send during run-time. For example, in Figure 5.1, message m_h is ready for transmission before the first bus cycle starts, but, after message m_f is transmitted, there is not enough room left in the DYN segment. This will delay the transmission of m_h for the next bus cycle.

1. In this thesis we do not address frame-packing [PopP05], and thus assume that one message is sent per frame.

Chapter 6
Timing Analysis of FlexRay Messages

Given a distributed system based on FlexRay, as described in the previous two sections, the tasks and messages have to be scheduled. As presented in Chapter 3, this means that for the SCS tasks and ST messages we need to build the schedule tables, while for the FPS tasks and DYN messages we have to determine their worst-case response times.

The global scheduling and analysis algorithm presented in Chapter 3 is, in principle, valid for all distributed embedded systems that rely on heterogeneous communication protocols, which makes it applicable for the FlexRay based systems, too. More exactly, this means that for the ST messages transmitted over FlexRay we can build the static cyclic schedule using the same algorithm presented in Figure 3.6. However, for the DYN messages we will have to modify the schedulability analysis, since the FlexRay DYN segment relies on FTDMA and not on CSMA/BA like in Chapter 3.

The next subsection presents our solution for computing the worst case response times of DYN messages, while in Section 6.2 we will integrate this solution into a holistic schedulability analysis that determines the timing properties of both FPS tasks and DYN messages (which is called in line 11, of the *schedule_TT_task* presented in Figure 3.6).

6.1 Schedulability Analysis of DYN Messages

We have captured in the following equation the worst case response time R_m of a FlexRay DYN message m:

$$R_m(t) = \sigma_m + w_m(t) + C_m, \quad (6.1)$$

where C_m is the message communication time (see Section 2.4), σ_m is the longest delay suffered during one bus cycle if the message is generated by its sender task after its slot has passed, and w_m is the worst case delay caused by the transmission of ST frames and higher priority DYN messages during a given time interval t. For example, in Figure 6.1, we consider that a message m is supposed to be transmitted in the 3rd DYN slot of the bus cycle. The figure presents the case when message m appears during the first bus cycle after the 3rd DYN slot has passed, therefore the message has to wait σ_m until the next bus cycle starts. In the second bus cycle, the message has to wait for the ST segment and for the first two DYN slots to finish, delay denoted with w_m (that also contains the transmission of a message *m*' that uses the second DYN slot).

The communication controller decides what message is to be sent on the bus in a certain communication slot *at the beginning* of that slot. As a consequence, in the worst case, a DYN message m is generated by its sender task immediately after the slot with the $FrameID_m$ has started, forcing message m to wait until the next bus cycle starts in order to really start

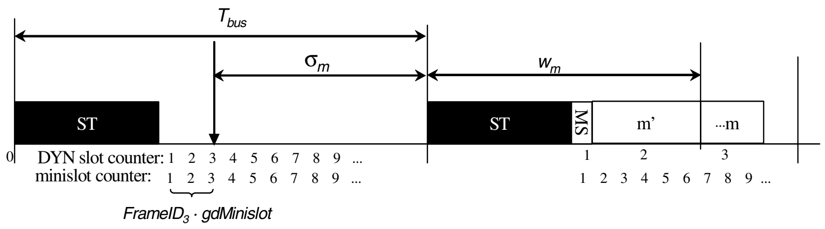

T_{bus}
σ_m
w_m
ST
ST
MS
m'
...m
0
DYN slot counter: 1 2 3 4 5 6 7 8 9 ...
minislot counter: 1 2 3 4 5 6 7 8 9 ...
1 2 3
1 2 3 4 5 6 7 8 9 ...
$FrameID_3 \cdot gdMinislot$

Figure 6.1: Response Time of a DYN Message

competing for the bus. In conclusion, in the worst case, the delay σ_m has the value:

$$\sigma_m = T_{bus} - (ST_{bus} + FrameID_m \cdot gdMinislot), \qquad (6.2)$$

where ST_{bus} is the length of the ST segment.

What is now left to be determined is the value w_m corresponding to the maximum amount of delay that can be produced by interference from ST frames and DYN messages. We start from the observations that the transmission of a ready DYN message m during the DYN slot $FrameID_m$ can be delayed because of the following causes:

- local messages with higher priority, that are generated by the same node and use the same frame identifier as m. We will denote this set of *higher priority local messages* with $hp(m)$. For example, in Figure 5.1.a, messages m_g and m_f share *FrameID* 4, thus $hp(m_g) = \{m_f\}$.
- any messages in the system that can use DYN slots with lower frame identifiers than the one used by m. We will denote this set of messages having *lower frame identifiers* with $lf(m)$. In Figure 5.1.a, $lf(m_g) = \{m_d, m_e\}$.
- unused DYN slots with frame identifiers lower than the one used for sending m (though such slots are unused, each of them still delays the transmission of m for an interval of time equal with the length *gdMinislot* of one minislot); we will denote the set of such minislots with $ms(m)$. Thus, in the example in Figure 5.1.a, $ms(m_g) = \{1, 2, 3\}$, and $ms(m_f)=\{3\}$.

Determining the interference of DYN messages in FlexRay is complicated by several factors. Let us consider the example in Figure 6.2, where we have two nodes, N_1 (with *FrameIDs* 1 and 3) and N_2 (with *FrameID* 2), and three messages m_1 to m_3. N_1 sends m_1 and m_3, and N_2 sends message m_2. Messages m_1 and m_2 have *FrameIDs* 1 and 2, respectively. We consider two situations: Figure 6.2.a, where m_3 has a separate *FrameID* 3, and Figure 6.2.b, where m_3 shares the same *FrameID* 1 with m_1. The values of *pLatestTx* for each node are depicted in the figure[1].

1. We use $pLatestTx_m$ to denote $pLatestTx_N$ of the node N sending message m.

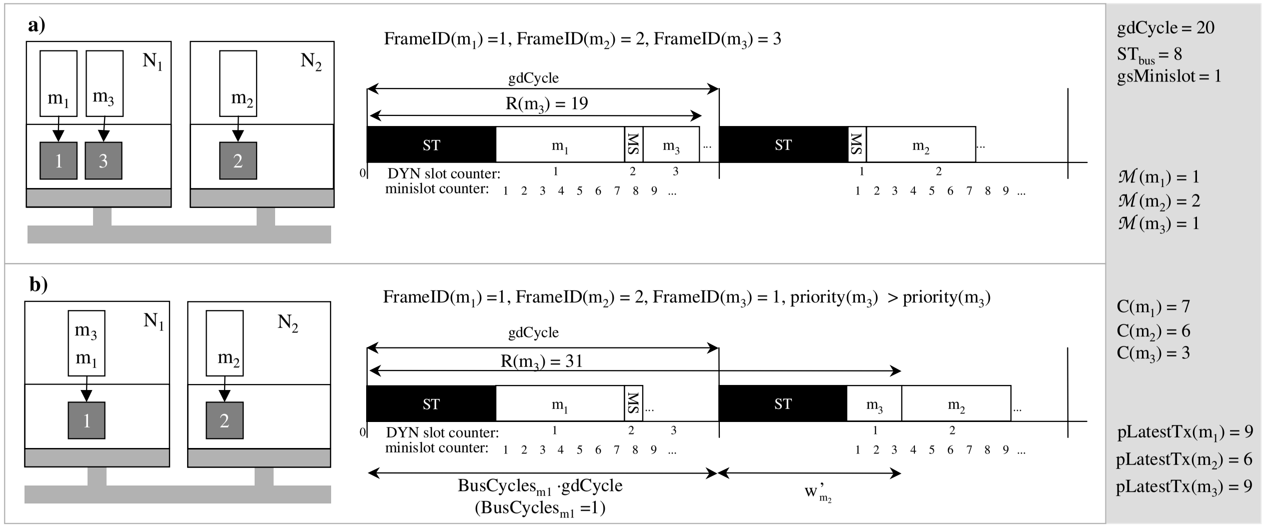

a)
N₁
m₁
m₃
1
3
N₂
m₂
2
FrameID(m₁) =1, FrameID(m₂) = 2, FrameID(m₃) = 3
gdCycle
R(m₃) = 19
ST
m₁
MS
m₃
...
ST
MS
m₂
...
0
DYN slot counter: 1 2 3 1 2
minislot counter: 1 2 3 4 5 6 7 8 9 ... 1 2 3 4 5 6 7 8 9 ...
b)
N₁
m₃
m₁
1
N₂
m₂
2
FrameID(m₁) =1, FrameID(m₂) = 2, FrameID(m₃) = 1, priority(m₃) > priority(m₃)
gdCycle
R(m₃) = 31
ST
m₁
MS
...
ST
m₃
m₂
...
0
DYN slot counter: 1 2 3 1 2
minislot counter: 1 2 3 4 5 6 7 8 9 ... 1 2 3 4 5 6 7 8 9 ...
BusCycles_m1 ·gdCycle
(BusCycles_m1 =1)
w'_m₂
gdCycle = 20
ST_bus = 8
gsMinislot = 1
ℳ(m₁) = 1
ℳ(m₂) = 2
ℳ(m₃) = 1
C(m₁) = 7
C(m₂) = 6
C(m₃) = 3
pLatestTx(m₁) = 9
pLatestTx(m₂) = 6
pLatestTx(m₃) = 9

Figure 6.2: Transmission Scenarios for DYN Messages

In Figure 6.2.a, message m_2, that has a lower FrameID than m_3, cannot be sent immediately after message m_1, because the value of the minislot counter has exceeded the value $pLatestTx_{m_2}$ when the value of the DYN slot counter becomes equal to 2 (hence, m_2 does not fit in this DYN cycle). As a consequence, the transmission of m_2 will be delayed for the next bus cycle. However, since in the moment when the DYN slot counter becomes 3 the minislot counter does not exceed the value $pLatestTx_{m_3}$, message m_3 will fit in the first bus cycle. Thus, a message (m_3 in our case) can be sent before another message with a lower *FrameID* (m_2). Such situations must be accounted for when building the worst-case scenario.

In Figure 6.2.b, message m_3 shares the same *FrameID* 1 with m_1 but we consider that it has a lower priority, thus $hp(m_3) = \{m_1\}$. In this case, m_3 is sent in the first DYN slot of the second bus cycle (the first slot of the first cycle is occupied with m_1) and thus will delay the transmission of m_2. In this scenario, we notice that assigning a lower frame identifier to a message does not necessarily reduce the worst-case response time of that message (compare to the situation in Figure 6.2.a, where m_3 has *FrameID* = 3).

We next focus on determining the delay $w_m(t)$ in Equation (6.1). The delay produced by all the elements in $hp(m)$, $lf(m)$ and $ms(m)$ can extend to one or more bus cycles. As a consequence, Equation (6.1) for finding the worst case response time R_m can be rewritten as:

$$R_m(t) = \sigma_m + BusCycles_m(t) \times T_{bus} + w'_m(t) + C_m \qquad (6.3)$$

where $BusCycles_m(t)$ is the number of bus periods for which the transmission of m is not possible because transmission of messages from $hp(m)$ and $lf(m)$ and because of minislots in $ms(m)$. The delay $w'_m(t)$ denotes now the time, in the last bus cycle, until m is sent, and is measured from the beginning of the bus cycle in which message m is sent until the actual transmission of m starts. For example, in Figure 6.2.b, $BusCycles_{m_2} = 1$ and $w'_{m_2}(t) = ST_{bus} + C_{m_3}$. Note that both these terms are functions of time, computed over an analysed interval t. This means that when computing them we have to take into consideration all the elements in $hp(m)$, $lp(m)$ and $ms(m)$ that can appear during such a given time interval t. Thus, we will consider the multi-set $hp(m, t)$ containing all the occurrences over t of

```
Compute_DYN_message_response_time($m$)
1  $t = C_m$
2  $R_{k\text{-}1}$ = t
3  do
4     compute $BusCycles_m(t)$
5     compute $w'_m(t)$
6     compute $R_k(t)$ using Equation (6.3)
7     set $t = R_k(t)$
8  until $Rk(t) = R_{k\text{-}1}(t)$
9  set $R_m = t$
```

Figure 6.3: Iterative process for solving Equation (6.3)

elements in *hp*(*m*). The number of such occurrences for a message $l \in hp(m)$ is equal to: $\lceil (J_l + t)/T_l \rceil$, where T_l is the period of the message l and J_l is its worst-case jitter (such a jitter is computed as the difference between the worst-case and best-case response times of its sender task *s*: $J_l = R_s - R_s^b$ [Pal98]). Similarly, *lf*(*m*, *t*) and *ms*(*m*, *t*) consider all the occurrences over *t* of elements in *lf*(*m*) and *ms*(*m*) respectively.

The next two sections (6.1.1 and 6.1.2) present the optimal (i.e., exact) solutions for determining the values for $BusCycles_m(t)$ and $w'_m(t)$, respectively. These, however, can be intractable for larger problem sizes. Hence, in Sections 6.1.3 and 6.1.4 we propose heuristics that quickly compute upper bounds (i.e., pessimistic) values for these terms.

Once for any given *t* we know how to obtain the values *BusCycles*(*t*) and $w'_m(t)$, we can use this knowledge to determine the worst case response time for a message *m* using equation Equation (6.3). This equation is solved using an iterative process that is depicted in Figure 6.3. The algorithm Compute_DYN_message_response_time starts by considering an initial value $t = C_m$ and an identical initial message response time $R_m = t$ (lines 1-2). However, the initial time interval *t* does not consider possible delays or interferences produced by other messages in the system. Therefore, the iteration that follows in lines 3-8 is adjusting the message response time by computing in each step *k* the values for $BusCycles_m(t)$, $w'_m(t)$ and *R*(*t*) respectively. At the end of each iteration, the analysed time interval is adjusted, according to the new value of the response time for message *m* (line 7). The entire process ends when the response times computed over two successive iterations does not change (line 8), meaning that the analysed time interval t is large enough to allow the transmission of all the messages in $hp(m)$ and $lf(m)$, of all the minislots in $ms(m)$, and of the message m itself, while accounting at the same time for all the ST segments that can appear during t. This means that the value t produced after the iteration is completed represents actually the computed worst-case response time of message m, hence it has to be saved accordingly (line 9).

6.1.1 OPTIMAL SOLUTION FOR $BusCycles_m$

We start with the observation that a message m with $FrameID_m$ cannot be sent by a node N_p during a bus cycle b if at least one of the following conditions is fulfilled:

1. There is too much interference from elements in $lf(m)$ and $ms(m)$, so that the minislot counter exceeds the value $pLatestTx_{N_p}$, making it impossible for N_p to start the transmission of m during b. For example in Figure 6.2.a, message m_2 cannot be sent during the first bus cycle because the transmission of a higher priority message m_1 pushes the minislot counter over the value $pLatestTx_{N_2}$.
2. The DYN slot $FrameID_m$ in b is used by another local higher priority message from $hp(m)$. For example, in Figure 6.2.b, messages m_1 and m_3 share the same frame identifier and $hp(m_3) = \{m_1\}$. Therefore, the transmission of m_3 in the first bus cycle is not possible.

Whenever a bus cycle satisfies at least one of these two conditions, it will be called "filled", since it is unusable for the transmission of the message m under analysis. In the worst case, the value $BusCycles_m(t)$ is then the maximum number of bus cycles that can be filled using elements from $hp(m)$, $lf(m)$ and $ms(m)$.

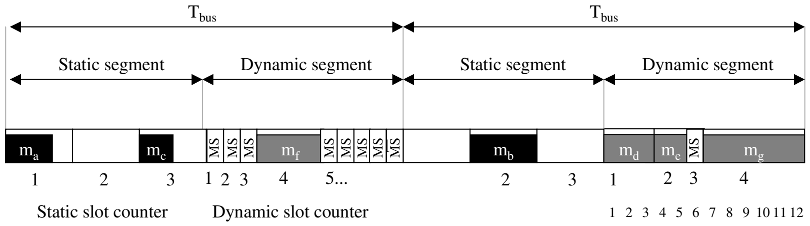


Figure 6.3: Worst Case Scenario for DYN message m_g

Since messages in *hp*(*m*, *t*) and *lf*(*m*, *t*) can become ready at any point during the analysed interval *t*, one can notice that, in the worst case, each bus cycle which is filled with an element from *hp*(*m*, *t*) will contain no messages from *lf*(*m*, *t*). This means that in the worst case, each filled bus cycle will contain either only messages from *lf*(*m*, *t*), or only one message from *hp*(*m*, *t*). For example, considering the same setup presented in Figure 5.1, the worst-case scenario for message m_g is when message m_f is ready at the beginning of the first bus cycle and messages m_d and m_e become ready just before the start of their slots in the second bus cycle (see Figure 6.3 for the worst-case scenario of m_g).

This means that, in the worst case, the delay produced by elements in *lf*(*m*, *t*) and *ms*(*m*, *t*) adds up to that produced by messages in *hp*(*m*, *t*):

$$\begin{aligned} BusCycles_m(t) = {} & BusCycles_m(hp(m,t)) + \\ & BusCycles_m(lf(m,t), ms(m,t)) \end{aligned} , \tag{6.4}$$

where we denote with *BusCycles*$_m$(*hp*(*m*, *t*)) the number of bus cycles in which the delay of the message *m* under analysis is produced by messages in *hp*(*m*, *t*) (corresponding to the second condition presented above); similarly, *BusCycles*$_m$(*lf*(*m*, *t*), *ms*(*m*, *t*)) is the number of "filled" bus cycles in which the transmission of message *m* is delayed by elements in *lf*(*m*, *t*) and *ms*(*m*, *t*) (corresponding to the first condition presented above).

Since each message in *hp*(*m*, *t*) delays the transmission of *m* with one bus cycle, the occurrences over *t* of messages in *hp*(*m*) will produce a delay equal to the total number of elements in *hp*(*m*, *t*):

$$BusCycles_m(hp(m,t)) = |hp(m,t)| \ . \tag{6.5}$$

The problem that remains to be solved is to determine how many bus cycles can be "filled" according to the first condition presented above using only elements in *lf*(*m*, *t*) and *ms*(*m*, *t*). As we will discuss later, a simplified version of this problem is equivalent to bin covering, which belongs to the family of NP-hard problems [Lab95]. To obtain the optimal solution, we have modelled the problem of computing *BusCycles*$_m$(*lf*(*m*, *t*), *ms*(*m*, *t*)) as an integer linear program (ILP). The model starts from the observation that, considering we have *n* elements in *lf*(*m*, *t*), there are at most *n* bus cycles that can be filled. For each such bus cycle we create a binary variable $y_{i=1..n}$ that is set to 1 when the *i*-th bus cycle is filled with elements from *lf*(*m*, *t*) and *ms*(*m*, *t*), and to 0 if it is not filled (i.e., it can allow the transmission of message *m* under analysis).

The goal of the ILP problem is to maximize the number of filled bus cycles (i.e., to calculate the worst-case):

$$BusCycles_m(lf(m,t), ms(m,t)) = \sum_{i=1..n} y_i \quad , \tag{6.6}$$

subject to a set of conditions that set the variables y_i to 1 or 0. Bellow we describe these conditions, which capture how messages in *lf*(*m*, *t*) and the minislots in *ms*(*m*, *t*) are sent by FlexRay in these bus cycles.

We allocate a binary variable x_{ijk} that is set to 1 if a message $m_k \in lf(m,t)$ ($k = 1..n$) is sent during the *i*-th bus cycle, using the *FrameID* $j = 1..FrameID_m$. The load transmitted in each bus cycle can be expressed as:

$$Load_i = \sum_{\substack{m_k \in lf(m,t) \\ j = 1...FrameID_m}} x_{ijk} \times C_k + \sum_{j = 1...FrameID_m} \left(1 - \sum_{m_k \in lf(m,t)} x_{ijk}\right) \times gdMinislot \quad , \tag{6.7}$$

where C_k are the communication times (Equation (2.1)) of the messages $m_k \in lf(m,t)$. Each term of the sum in Equation (6.7) captures the particularities of FlexRay DYN frames: if a message *k* is transmitted in cycle *i* with frame identifier *j*, then $x_{ijk} = 1$ and the length of the frame being transmitted is equal with the length of the message *k*, (thus the term $x_{ijk} \times C_k$); if x_{ijk} is 0 for all *j* and *k*, then there is no actual transmission on the bus in that DYN slot, but there is still some delay due to the empty minislot of length *gdMinislot* that has to pass in order to increase the value of the DYN slot counter (thus the second term).

The condition that sets each variable y_i to 1 whenever possible is:

$$Load_i > pLatestTx_{N_p} \times gdMinislot \times y_i \quad , \tag{6.8}$$

where $pLatestTx_{N_p}$ is the last minislot which allows the start of transmission from node N_p which generates the message *m* under analysis. Such a condition enforces that a variable y_i cannot be set to 1 unless the total amount of interference from *lf*(m, t) and *ms*(m, t) in cycle i exceeds $pLatestTx_{N_p}$ minislots (only then message m is not allowed to be transmitted and, thus, bus cycle i is "filled").

In addition to this condition we have to make sure that

- each message $m_k \in lf(m, t)$ is sent in only one cycle i:

$$\sum_{\substack{i = 1 \dots n \\ j = 1 \dots FrameID_m}} x_{ijk} \le 1, \forall\, m_k \in lf(m, t)\,; \tag{6.9}$$

- each frame identifier is used only once in a bus cycle:

$$\sum_{k = 1 \dots n} x_{ijk} \le 1, \forall\, i, j \;\; ; \tag{6.10}$$

- each message $m_k \in lf(m, t)$ is transmitted using its frame identifier:

$$x_{ijk} \le Frame_{jk}, \;\; \forall\, i, j, k \;\; , \tag{6.11}$$

where $Frame_{jk}$ is a binary constant with value 1 if message $m_k \in lf(m, t)$ has a frame identifier $FrameID_{m_k} = j$ (otherwise, $Frame_{jk}$ is 0).

Finally, we have to enforce that in every cycle i no message m_k will start transmission after its associated $pLatestTx_{m_k}$. If we have $x_{ijk} = 1$, then we have to add the condition that the total amount of transmission that takes place before DYN slot j has to finish no later than $pLatestTx_k$:

$$\sum_{\substack{m_q \in lf(m, t) \\ p = 1..j-1}} x_{ipq} \times C_q + \sum_{p = 1..j-1} \left(1 - \sum_{m_q \in lf(m, t)} x_{ipq}\right) \times gdMinislot \le pLatestTx_k \times gdMinislot \tag{6.12}$$

The conditions (6.7)–(6.12) together with the maximisation goal expressed in Equation (6.6) define the ILP program that will determine the maximum worst-case number of bus cycles that can be filled with elements in *lf*(m, t) and *ms*(m, t). By adding this result to the value determined in Equation (6.5), we obtain the total number $BusCycles_m(t)$ (Equation (6.4)).

6.1.2 OPTIMAL SOLUTION FOR w_m'

Coming back to Equation (6.3), all we have left to determine is the value of w_m'. In the worst case, the elements in *lf*(*m*,*t*) and *ms*(*m*,*t*) will delay the message under analysis for $BusCycles_m$ (*lf*(*m*,*t*), *ms*(*m*,*t*)) bus periods. In addition, they will delay the actual transmission of *m* during the DYN segment of the bus period $BusCycles_m$ + 1, by an amount w_m'.

The problem of determining the value for w_m' is defined as follows: given the multi-sets *lf*(*m*,*t*) and *ms*(*m*,*t*) and the maximum number $BusCycles_m$(*lf*(*m*,*t*), *ms*(*m*,*t*)) that they can fill, what is the maximum possible load (Equation (6.7)) in the first unfilled bus cycle (i.e. the bus cycle that does not satisfy the condition in Equation (6.8)).

In order to determine the exact value of w_m' in the worst case, one can use the same ILP formulation defined in the previous section for computing $BusCycles_m$(*lf*(*m*,*t*), *ms*(*m*,*t*)), with the following modifications:

- since we know the value $BusCycles_m$ (which is determined solving the ILP formulation presented in the previous section), we add conditions that force the values $y_i = 1$ for all $i=1..BusCycles_m$, and $y_i = 0$ for all $i = BusCycles_m + 1..n$; in this way, the messages will be packed so that the bus cycles from 1 to $BusCycles_m$ will be filled (i.e they satisfy the condition expressed in Equation (6.8)), while the remaining bus cycles will be unfilled.
- using the same set of conditions (6.7)–(6.12) for filling the first $BusCycles_m$ cycles, the goal described in Equation (6.6) is replaced with the following one, expressing that the load of the cycle number $BusCycles_m$ + 1 has to be maximized ($Load_L$ is expressed as in Equation (6.7)):

$$maximize\ Load_L,\ for\ L = BusCycles_m + 1 \tag{6.13}$$

6.1.3 HEURISTIC SOLUTION FOR $BUSCYCLES_M$

We first make the observation that in a bus cycle where a message *m* is sent by a node N_p during DYN slot $FrameID_m$, in the worst case there will be at most $FrameID_m - 1$ unused minislots before *m* is transmitted (in

Figure 6.2.a, the transmission of m_2 can be preceded by at most one unused minislot).

Instead of considering the multiset $ms(m, t)$ to calculate the actual number of unused minislots before message m, as we did for the exact solution, we will consider the worst-case number of minislots. The delay produced by the minislots will be considered as part of the message communication time as follows (see also Equation (2.1)):

$$C_m' = (FrameID_m - 1) \times gdMinislot + C_m \,. \tag{6.14}$$

Since the duration of one minislot (*gdMinislot*) is an order of magnitude smaller compared to the length of a cycle, this approximation will not introduce any significant pessimism.

The problem left to solve now is how many bus cycles can be filled with the elements from a multiset $lf'(m, t)$, that consists of all the messages in $lf(m, t)$ for which we consider the communication times computed using Equation (6.14).

If we ignore the conditions expressed in equations (6.10)-(6.12), then determining $BusCycles_m(lf'(m, t))$ becomes a *bin covering* problem [Ass84]. Bin covering tries to maximize the number of bins that can be filled to a fixed minimum capacity using a given set of items with specified weights. In our scenario, the messages in $lf'(m, t)$ are the items, the dynamic segments of the bus cycles are bins, and $pLatestTx_{N_p} \times gdMinislot$ is the minimum capacity required to fill a bin. The bin-covering problem is NP-hard in the strong sense [Lab95], and our heuristic solution is to determine an upper bound, using the approach presented in [Lab95], on the number of maximum bins that can be covered. The upper bounds proposed in [Lab95] are of polynomial complexity and lead to very good quality results (see Appendix B).

Note that, ignoring the conditions from equations (6.10)-(6.12) and determining an upper bound for bin-covering can only lead to a possible increase in the number of bus cycles compared to the exact solution. Experiments will show the impact of the heuristic on the pessimism of the analysis.

6.1.4 HEURISTIC SOLUTION FOR w_m'

A straightforward heuristic to the computation of w_m' stems from the observation that, in a hypothetical worst-case scenario, message *m* could be sent in the last possible moment of the current bus cycle, which means that

$$w'_m = ST_{bus} + pLatestTx_{N_p} \times gdMinislot \ , \qquad (6.15)$$

where ST_{bus} is the length of the ST segment of a bus cycle.

6.2 Holistic Schedulability Analysis of FPS Tasks and DYN Messages

The schedulability analysis algorithm that is depicted in Section 3.3, Figure 3.3 considers that the messages are transmitted according to a fixed-priority policy. For this reason, the algorithm uses the same procedures for computing worst-case response times for both the DYN messages and for the FPS tasks (the only difference is that the messages must consider a blocking time needed to model the non-preemptivity of the transmission). If we replace the communication protocol with FlexRay, then the analysis algorithm has to implement the technique presented in the previous section in order to capture the worst-case response times of DYN messages. The modified algorithm for the schedulability analysis is presented in Figure 6.4, where the code added in lines 5-7 is responsible for taking care of the situations when the analysed system activity is a DYN message. In such a situation, the algorithm has to call the procedure that computes the worst-case response time of that message (the procedure has been presented in Figure 6.3).

What is important to mention is that in a distributed system, the worst-case response time R_{ij} depends on the jitters of the higher priority tasks and predecessors of τ_{ij}. This means that for all the activities in the system we must be able to compute their jitter. According to the analysis of multiprocessor and distributed systems presented in [Pal98], the jitter for a system activity *a* that is data dependant on another system activity *b* can be

```
1  do
2   Done = true
3   for each transaction $\Gamma_i$ do
4      for each activity $\tau_{ij}$ in $\Gamma_i$ do
5         if $\tau_{ij}$ is a DYN message then
6            $R_{ij}^{max}$ = Compute_DYN_message_response_time($\tau_{ij}$)
7         else
8            for each task $\tau_{ik}$ in $\Gamma_i$ do
9               if $Prio_{ik} \geq Prio_{ij}$ and $\mathcal{M}(\tau_{ik}) = \mathcal{M}(\tau_{ij})$ then
10                 for each job p of $\tau_{ij}$ do
11                    Consider that $\tau_{ik}$ initiates $t_c$
12                    Compute $R_{ij}^p$
13                    if $R_{ij}^p > R_{ij}^{max}$ then
14                       $R_{ij}^{max} = R_{ij}^p$
15                    endif
16                 endfor
17              endif
18           endfor
19        endif
20        if $R_{ij}^{max} > R_{ij}$ then -- larger $R_{ij}$ found
21           $R_{ij} = R_{ij}^{max}$
22           Done = false
23           for each successor $\tau_{ik}$ of $\tau_{ij}$ do
24              $J_{ik} = R_{ij} - R_{ij}^b$ -- update jitters
25           endfor
26        endif
27     endfor
28  endfor
29 while (Done!= true)
```

Figure 6.4: Schedulability Analysis Algorithm for FlexRay-based Systems

computed as the difference between the worst-case response time R_b and best-case response time $R^b{}_b$ of the predecessor. In the algorithm in Figure 6.4, this computation is performed on line 24, where each time the worst-case response time of a system activity is updated, the jitters of all its successors are also recomputed. For all the tasks in the system we already know how to compute the worst-case and best-case response times, which means that we are able to compute the jitter of any activity that is preceded by a task. What we have to investigate now is the case when a task is preceded by a DYN message and see how the analysis of the message timing properties influences the jitters of such a task.

Let us consider a task τ_r that starts execution only after it receives a message m. Its jitter depends on the values of the best-case and worst-case transmission times of that message:

$$J_{\tau_r} = R_m - R_m^b. \tag{6.16}$$

The calculation of the worst-case transmission time R_m of a DYN message m was presented in Section 6.1. For computing R_m^b we have to identify the best-case scenario of transmitting message m. Such a situation appears when the message becomes ready immediately before the DYN slot with $FrameID_m$ starts, and it is sent during that bus cycle without experiencing any delay from higher priority messages. Thus, the equation for the best-case transmission time of a message is:

$$R_m^b = C_m, \tag{6.17}$$

where C_m is the time needed to send the message m.

Let us make a final remark. According to [Ped00], the worst-case response time calculation of FPS tasks is of exponential complexity. The approximation approach proposed in [Ped00] and also used in the algorithm presented in Figure 6.4 is a heuristic with a certain degree of pessimism. The pessimism of the response times calculated by our holistic analysis will, of course, also depend on the quality of the solution for the delay induced by the DYN messages transmitted over FlexRay. The calculation of this delay is our main concern in this chapter. Therefore, when we speak about optimal and heuristic solutions in this chapter we refer to the approach used for calculating the $BusCycles_m$ and w_m' (used in the worst-case response times calculation for DYN messages) and not the holistic response time analysis which is based on the heuristics in [Ped00].

6.3 Analysis for Dual-channel FlexRay Bus

The specification of the FlexRay protocol mentions that the bus has two communication channels [Fle07]. The analysis presented above is appropriate for systems where the two channels of the FlexRay bus are used in a redundant manner, transporting the same information simultaneously in order to support fault-tolerance.

In order to increase the bandwidth of the bus, one can use the two channels independently, so that different sets of messages are sent over each of the channels during a bus cycle. In this section we outline the extension of our previous analysis in order to compute the worst case response times for messages transmitted in such systems.

First, we extend our system model presented in Chapter 2 and consider that all nodes in the system have access to a dual-channel FlexRay bus. As a consequence, in the application model each message *m* is associated a pair <$FrameID_m$, $Channel_m$>, with the meaning that message *m* is sent during $FrameID_m$ on $Channel_m$ (where $Channel_m = \{A, B\}$).

Second, we notice that the transmission of a message can be delayed only by messages that are transmitted on the same channel. As a consequence, the only modification in the analysis presented in Section 6.1 is the definition of the sets *lf*(*m*) and *hp*(*m*), which contain only those messages that are transmitted on $Channel_m$:

- *hp*(*m*) becomes now the set of local messages with higher priority, that use the same frame identifier *and* the same channel as *m*.
- *lf*(*m*) contains any messages in the system that can use $Channel_m$ and DYN slots with lower frame identifiers than the one used by *m*.

6.4 Evaluation of Analysis Algorithms

We are interested to determine the quality of the proposed analysis approaches, and how well they scale with the number of FlexRay messages that have to be analysed. All the experiments were run on P4 machines using 2GB RAM. The ILP-based solutions have been implemented using the CPLEX 9.1.2 ILP solver[1].

We have generated synthetic applications of 20, 30, 40 and 50 tasks mapped on architectures consisting of 2, 3, 4, and 5 nodes, respectively.

1. http://www.ilog.com/products/cplex

Fifteen applications were generated for each of these four cases. The number of time-critical FlexRay messages were 30, 60, 90, and 120 for each case, respectively. Out of these, 10, 20, 30, and 40 messages were time-critical DYN messages that were analysed using the approaches presented in Section 5. Each application has been analysed using four holistic analysis approaches, depending on the approach used for the calculation of the components $BusCycles_m$ and w_m' of the worst-case response time R_m for a DYN message:

Holistic Analysis	$BusCycles_m$	w_m'
OO	**O**ptimal solution (6.1.1)	**O**ptimal solution (6.1.2)
OO$^-$	**O**ptimal solution (6.1.1)	ILP from 6.1.2 with 1 min. time-out (**O**$^-$)
OH	**O**ptimal solution (6.1.1)	**H**euristic solution (6.1.4)
HH	**H**euristic solution (6.1.3)	**H**euristic solution (6.1.4)

OO will always provide the tightest worst-case response times. However, it is only able to produce results for up to 20 DYN messages in a reasonable time. We have noticed that the bottleneck for OO is the exact calculation of w_m' (which is a value smaller than a bus cycle), and that running the ILP from Section 6.1.2 using a time-out of one minute we are able to obtain near-optimal results for w_m'. We have denoted with OO$^-$ such an analysis. Since the near-optimal result for w_m', obtained after this time-out, is a lower bound, OO$^-$ can lead to an incorrect (optimistic) result (i.e., the system is reported as schedulable, but in reality it might not be). Although OO$^-$ is, thus, of no practical use, it is very useful in determining, by comparison, the quality of our proposed FlexRay analysis heuristics, OH and HH.

In order to evaluate the approaches for FlexRay analysis, we have determined for an analysis approach A the average ratio:

$$\text{ratio} = \frac{1}{n} \cdot \sum_{m \in DYN} \frac{R_m^A}{R_m^{OO^-}} \tag{6.18}$$

where A is one of the OO, OH or HH approaches and n is the number of DYN messages in the analysed application.

This ratio captures the degree of pessimism for approach *A* compared to OO$^-$; the smaller the ratio, the less pessimistic the analysis. The results obtained with OO, OH and HH are presented in Table 6.1. For each application dimension, Table 6.1 presents the average ratio and the average execution times of the complete analysis (including all tasks and messages) in seconds. It is important to notice that, while the execution time is for the whole analysis, including all tasks and messages, the ratio is calculated only for the DYN messages, since their response time calculation is directly affected by the degree of pessimism of the various approaches proposed in this chapter. The ratio calculated over all tasks and messages in the system is smaller than the ones shown in Table 6.1.

We can see that OO is very close to OO$^-$, which means that OO$^-$ is a good comparison baseline (it is only slightly optimistic). Due to the very large execution times, we were not able to run OO for more than 20 DYN messages.

Table 6.1 shows that OH produces very good quality results, in a reasonable time. For example, for 40 DYN messages, the analysis has finished in 367.87 seconds on average, and the average ratio is only 1.005.

Another result from Table 6.1 concerns the HH heuristic. Although HH is slightly more pessimistic than OH (for example, the DYN response times determined with HH were 1.012 times larger, on average, than those of OO$^-$ for applications with 30 messages, compared to 1.005 for OH), it is also significantly faster. We have successfully analysed with HH large applications, with over 100 DYN messages in 0.16 seconds on average. Thus, HH is also suitable for design space exploration, where a potentially huge number of design alternatives have to be analysed in a very short time.

We have run a set of experiments with 15 applications of 40 tasks and 25 dynamic messages mapped on an architecture consisting of two nodes, and varied the number of frame identifiers per processor. Figure 6.5 presents the ratio for HH calculated according to Equation (6.18) as we vary the number of frame identifiers per processor from 2 to 6. We can see that the quality of the heuristic improves as the number of frame IDs increases (and, consequently, the number of messages sharing the same *FrameID* decreases). The more messages are sharing a *FrameID*, the more

No of msgs.	30 (10 DYN)		60 (20 DYN)		90 (30 DYN)		120 (40 DYN)	
	Ratio	Exec. (s)	Ratio	Exec. (s)	Ratio	Exec. (s)	Ratio	Exec. (s)
OO	1.009	3.1 s	1.009	42.3 s	–	–	–	–
OH	1.013	1.29 s	1.012	14.42 s	1.005	57.32 s	1.005	367.87 s
HH	1.016	0.012 s	1.018	0.019 s	1.012	0.036 s	1.012	0.04 s

Table 6.1: Comparison of FlexRay Analysis Approaches

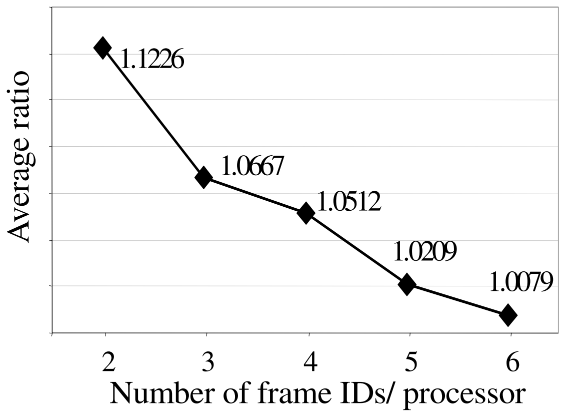


Figure 6.5: Quality of HH

important conditions (6.10)-(6.12) are to the quality of the result, because they restrict the way bins can be covered (e.g., messages sharing the same *FrameID* should not be packed in the same bin). However, even for a small number of frame IDs HH produces good quality results (e.g., for two frame IDs, HH's ratio is 1.1226).

We also considered the real-life example implementing a vehicle cruise controller that is described in Appendix C. We considered the entire functionality is implemented using fixed priority scheduling and DYN communication. However, we have allocated only 10 percent of the FlexRay communication cycle to the DYN segment communication. Scheduling the system using the OO approach took 0.19 seconds. Using the OH approach took 0.08 s, while the HH alternative was the fastest, finishing the analysis in 0.002 s. The average ratio of OH relative to OO is 1.003, while the average ratio of HH relative to OO is 1.004, which means that the heuristics obtained results almost identical to the optimal approach OO.

6.5 Conclusions

In this chapter, we have presented a schedulability analysis for the FlexRay communication protocol. Since we considered that for ST messages we can build a static cyclic schedule just like in the approach discussed in Chapter 3, we have concentrated only on determining the worst-case response time analysis of the DYN messages. We have also shown how the schedulability analysis of DYN messages can be integrated with the holistic schedulability analysis that determines the timing properties for all the ET tasks and DYN messages in the system.

We have proposed three approaches for the derivation of worst-case response times of DYN messages. OO uses an ILP formulation to derive the optimal solution for the communication delay. HH uses heuristic-based upper-bounds for a bin-covering problem in order to quickly determine good quality response times. OH is able to further reduce the pessimism of HH by using an ILP formulation for one part of the solution.

Chapter 7 Optimisation of the FlexRay Bus Access Scheme

IN THIS CHAPTER we consider the problem of optimising the bus access parameters for an application implemented on a FlexRay-based distributed system.

7.1 Introduction

The design of a FlexRay bus configuration for a given system consists of a collection of solutions for the following subproblems:

1.determine the length of an ST slot;
2.the number of STslots
3.the assignment of ST slots to nodes;
4.determine the length of the DYN segment
5.assign DYN slots to nodes
6.assign *FrameID*s to DYN messages.

The choice of a particular bus configuration is extremely important when designing a specific system, since its characteristics heavily influence the global timing properties of the application.

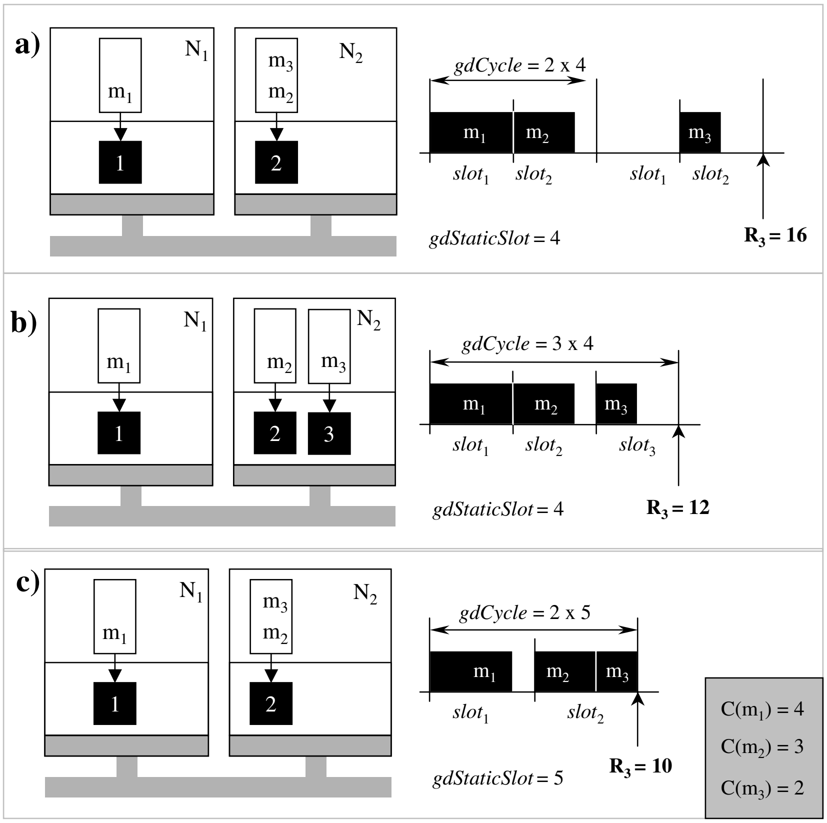


Figure 7.1: Influence of the Bus Cycle on the Response Times of ST Messages

Let us see first an example similar to the one presented in Figure 4.4, that presents such an influence in the particular context of FlexRay buses. Notice in Figure 7.1 how the structure of the ST segment affects the response time of message m_3 (for this example we ignored the DYN segment and task execution on each node). The figure considers a system with two nodes, N_1 that sends message m_1 and N_2 that sends messages m_2 and m_3. The message sizes are depicted in the figure. In the first scenario (Figure 7.1.a), the ST segment consists of two slots, $slot_1$ used by N_1 and $slot_2$ used by N_2. In this situation, message m_3 can be scheduled only dur-

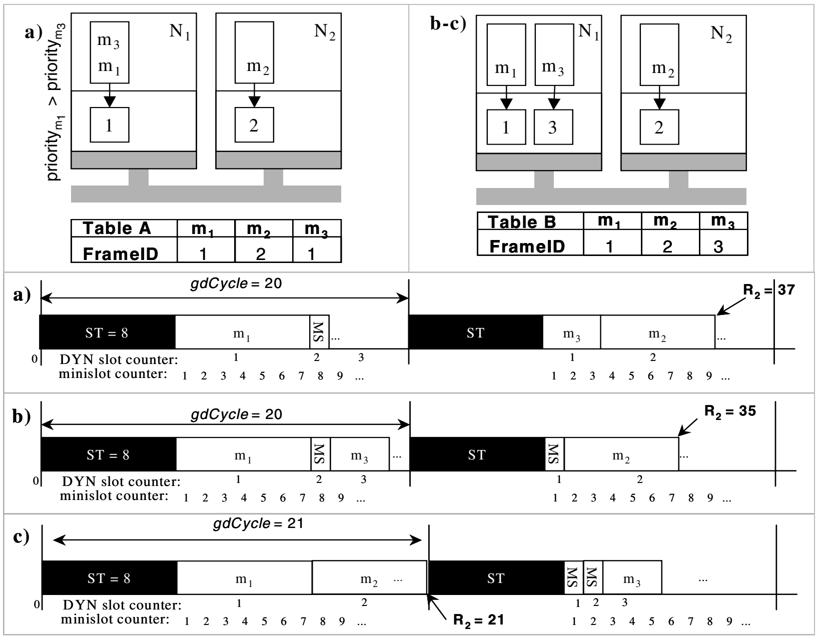


Figure 7.2: Influence of the Bus Cycle on the Response Times of the DYN Messages

ing the second bus cycle, with a response time of 16. If the ST segment consists of 3 slots (Figure 7.1.b), with N_2 being allocated $slot_2$ and $slot_3$, then N_2 is able to send both its messages during the first bus cycle. The configuration in Figure 7.1.c consists of only two slots, like in Figure 7.1.a. However, in this case the slots are longer, such that several messages can be transmitted during the same frame, producing a faster response time for m_3 (one should notice, however, that by extending the size of the ST slots we delay the reception of messages m_1 and m_2).[1]

Similar optimisations can be performed with regard to the DYN segment. Let us consider the example in Figure 7.2, where we have two nodes

1. It is important to underline here the fact that in FlexRay:
   a) all ST slots have the same length
   b) each node can have several ST slots during the bus cycle.

N_1 and N_2. Node N_1 is transmitting messages m_1 and m_3, while N_2 sends m_2. Figure 7.2 depicts three configuration scenarios, a-c. Table A depicts the frame identifiers for the scenario in Figure 7.2.a, while Table B corresponds to Figure 7.2.b-c. The length of the ST slot has been set to 8. In Figure 7.2.a, the length of the DYN segment is not able to accommodate both m_1 and m_2, thus m_2 will be sent during the second bus cycle, after the transmission of m_3 ends. Figure 7.2.b and Figure 7.2.c depict the same system but with a different allocation of DYN slots to messages (Table B). In Figure 7.2.b we notice that m_3, which now does not share the same frame identifier with m_1, can be sent during the first bus cycle, thus m_2 will be transmitted earlier during the second cycle. Moreover, if we enlarge the size of the DYN segment as in Figure 7.2.c, then the worst-case response time of m_2 will considerably decrease since it will be sent during the first bus cycle (notice that in this case m_3, having a greater frame identifier than that of m_2, will be sent only during the second cycle).

We present three approaches for optimising the bus access such that the schedulability of the system is improved. The first approach builds a relatively straightforward, basic, bus configuration. The other two approaches perform optimisation over the basic configuration.

7.2 The Basic Bus Configuration (BBC)

In this section we construct a basic bus configuration which results from analysing the minimal bandwidth requirements of the application. The BBC algorithm is presented in Figure 7.3 and it starts by assigning a *FrameID* to each of the DYN messages (implicitly DYN slots are assigned to the nodes that generate the message). This assignment (line 1) is performed under the following guidelines:

- Each DYN message receives an unique *FrameID*; this is recommended in order to avoid delays due to messages in the set *hp*(m), as discussed in 6.1. For example, in Figure 6.2, we notice that message m_3 has to wait for an entire *gdCycle* when it shares a frame identifier with the higher priority message m_1 (Figure 6.2.a), which is not the case when it has its own *FrameID* (Figure 6.2.b).

- DYN messages with a higher criticality receive smaller *FrameID*s.; this is required in order to reduce, for a given message, the delay produced by *lpf*(*m*) and *ms*(*m*) (see 6.1). We capture the criticality of a message *m* as:

$$CP_m = D_m - LP_m, \tag{7.1}$$

where D_m is the deadline of the message and LP_m is the longest path in the task graph from the root to the node representing the communication of message *m*. A small value of CP_m (higher criticality) indicates that the message should be assigned a smaller *FrameID*.

In the next step, the algorithm sets the number of ST slots in a bus cycle (line 2). Since each node that generates ST messages needs at least one ST slot, the minimum number of ST slots is $nodes_{ST}$, the number of nodes that send ST messages. Next, the size of an ST slot is set so that it can accommodate the largest ST message in the system (line 3). In line 4, the configuration of the ST segment is completed by assigning in a round robin fashion one ST slot to each node that requires one (i.e. in a system with four nodes, the ST segment will contain four slots: node 1 will use slot 1, node 2 will use ST slot 2, etc.).

In order to determine the size of the DYN segment, we have to consider the fact that such a size is restricted by the protocol specifications (there can be at most 7994 minislots in a DYN segment) and by the application characteristics (the DYN segment should be large enough in order to accommodate the transmission of the largest DYN message; in addition, since we assumed that each DYN message has an unique *FrameID*, the DYN segment should have a number of minislots greater or equal than the number of DYN messages in the system). We denote with DYN_{bus}^{min} and DYN_{bus}^{max} the limits of this interval (line 5).

Since the sizes of the ST and DYN segments are now fixed, the bus period can be easily computed (line 6). Line 7 introduces a restriction

```
1  Assign FrameIDs to DYN messages
2  gdNumberOfStaticSlots = nodes_ST
3  gdStaticSlot = max ($C_m$), $m$ is an ST message
4  Assign one ST slot to each node (round robin)
5  for $DYN_{bus} = DYN_{bus}^{min}$ to $DYN_{bus}^{max}$ step gdMinislot do
6     gdCycle = $ST_{bus} + DYN_{bus}$
7     if gdCycle < 16000 μs then
8        GlobalSchedulingAlgorithm()
9        Compute cost function DSch
10       if DSch < BestDSch then save current solution
11    endif
12 end for
```

Figure 7.3: Basic Bus Configuration

imposed by the FlexRay specification, which limits the maximum bus cycle length to 16 ms.

Once we have defined the structure of the bus cycle, we can analyse the entire system (line 8) by performing the global static scheduling and schedulability analysis described in Section 6. The resulted system is then evaluated using a cost function that captures the schedulability degree of the system (line 9). This function has been already defined in Equation (3.17):

$$\mathrm{DSch} = \begin{cases} f_1 = \sum_{i=1}^{N} \sum_{j=1}^{N_i} max(0, R_{ij} - D_{ij}) & \text{, if } f_1 > 0 \\ f_2 = \sum_{i=1}^{N} \sum_{j=1}^{N_i} (R_{ij} - D_{ij}) & \text{, if } f_1 = 0 \end{cases}$$

where R_{ij} and D_{ij} are the worst case response times and respectively the deadlines for all the activities τ_{ij} in the system. The function is strict positive if at least one task or message in the system misses its deadline, and negative if the whole system is schedulable. Its value is used in line 10, when deciding whether the current configuration is the best so far.

```
1 Assign FrameIDs to DYN messages
2 for $gdNumberOfStaticSlots$ =
      $gdNumberOfStaticSlots_{min}$ to $gdNumberOfStaticSlots_{max}$ do
3   for $gdStaticSlot = gdStaticSlot_{min}$ to $gdStaticSlot_{max}$
                                           step 20 * gdBit do
4     Assign ST slots to nodes in round-robin fashion
5     $DYN_{bus}$ = Determine_DYN_segment_length()
6     End optimisation if feasible $DYN_{bus}$ and $DSch \leq 0$
7   end for
8 end for
```

Figure 7.4: OBC Heuristic

7.3 Heuristic for Optimised Bus Configuration (OBC)

The Basic Bus Configuration (BBC) generated as in the previous section can result in an unschedulable system (the cost function in Equation (3.17) is positive). In this case, additional points in the solution space have to be explored. In Figure 7.4 we present the OBC heuristic that further explores the design space in order to find a schedulable solution.

While for the BBC the number and size of ST slots has been set to the minimum ($gdNumberOfStaticSlots_{min}$ = $nodes_{ST}$, $gdStaticSlot_{min}$ = $\max(C_m)$), the proposed heuristic explores different alternatives between these minimal values and the maxima imposed by the protocol specification (the for loops over lines 2-8 and 4-7). Thus, during a bus cycle there can be at most $gdNumberOfStaticSlots_{max}$ = 1023 ST slots (line 3), while the size of an ST slot can take at most $gdStaticSlot_{max}$ = 661 macroticks. In addition, the payload for a FlexRay frame can increase only in 2-byte increments, which according to the FlexRay specification translates into 20 *gdBit*, where *gdBit* is the time needed for transmitting one bit over the bus (line 3).

The assignment of ST slots (line 5) to nodes is performed, like for the BBC, in a round robin fashion, with the difference that each node can have not only one but a quota of ST slots, determined by the ratio of ST mes-

```
Determine_DYN_segment_length()
1 for $DYN_{bus}$ = $DYN_{bus}^{min}$ to $DYN_{bus}^{max}$ step gdMinislot do
2    gdCycle = $ST_{bus}$ + $DYN_{bus}$
3    if gdCycle < 16000 μs then
4       GlobalSchedulingAlgorithm()
5       Compute schedulability degree DSch
6       if DSch < BestDSch then $BestDYN_{bus}$ = $DYN_{bus}$
7    endif
8  end for
9 return $BestDYN_{bus}$
```

Figure 7.5: Exhaustive Search for the length of the DYN segment

sages that it transmits (i.e. a node that sends more ST messages will be allocated more ST slots).

For each alternative configuration of the ST segment, the algorithm searches for that size of the DYN segment that allows the DYN messages to meet their deadlines and the cost function in Equation (3.17) to be minimised (line 6). A straight forward alternative to perform this would be to evaluate all possible sizes of the DYN segment inside a for loop, like in the BBC algorithm (lines 5-12, Figure 7.3). Such an exhaustive implementation is presented in Figure 7.5.

However, as opposed to the BBC, in the proposed heuristic the selection of the DYN segment length is nested inside two for loops (lines 2 and 3, Figure 7.4). Moreover, the estimation of each individual solution alternative implies a complete scheduling and schedulability analysis of the systems (like in line 4, Figure 7.5). Therefore, in the context of the heuristic in Figure 7.4, such a straight forward approach is not affordable, due to excessively long run times. This is important, since, in the context of a system-level design framework, the bus access optimisation heuristic can be placed inside other optimisation loops, e.g. for task mapping. Thus, instead of running the scheduling and schedulability analysis and evaluating the cost function in Equation (3.17) for all possible lengths of the DYN segment (like in line 5, Figure 7.5), the evaluation should be performed for only a reduced number of points while, at the same time, obtaining a close to optimal result. The proposed solution is presented in the next subsection.

7.3.1 CURVE-FITTING BASED HEURISTIC FOR DYN SEGMENT LENGTH

Let us go back to the schedulability analysis in 6.1. One can notice in Equation (6.3) that the dominant part of the message delay is represented by the product between $BusCycles_m$ (number of bus cycles that the message under analysis has to wait) and *gdCycle* (length of the bus cycle). If we consider a time interval t on which a fixed set of DYN messages S is generated, then a shorter size for the DYN segment means that fewer messages will be served during each bus cycle; consequently, several such bus cycles are needed to transmit all the messages in S (considering a fixed size ST_{bus} for the static segment, then a shorter DYN_{bus} results in a shorter *gdCycle* but in a larger value for $BusCycles_m$). A longer DYN segment generally means that more DYN messages can be sent during the same bus period, resulting in a lower number of bus cycles required for the transmission of all messages (a larger DYN_{bus} increases the length *gdCycle*, but results in smaller value for $BusCycles_m$). The resulted trade-off is illustrated in Figure 7.6, where we consider a system composed of 45 tasks which communicate through 10 static and 20 dynamic messages. We have performed the response time analysis for this system, assuming the length of the dynamic segment between 2285.4 and 13000 µs. The static segment size is fixed at 1286 and, consequently, the total size of the bus cycle is varying between 3571.4 and 14286µs. Figure 7.6 shows the response time for several dynamic messages in this system. The curves confirm the trade-off outlined above. Large sizes of the bus cycle lead to increased response times. However, very short bus cycles will also lead to large response times due to the fact that the number of cycles to wait ($BusCycles_m$ in Equation (6.3)) increases. This phenomenon has been confirmed by a large number of experiments similar to those illustrated in Figure 7.6. This regularity of the dependence *response time* vs. *size of the dynamic segment* is at the foundation of our heuristic presented in Figure 7.7 (which is invoked in line 5 of the OBC algorithm in Figure 7.4). Instead of

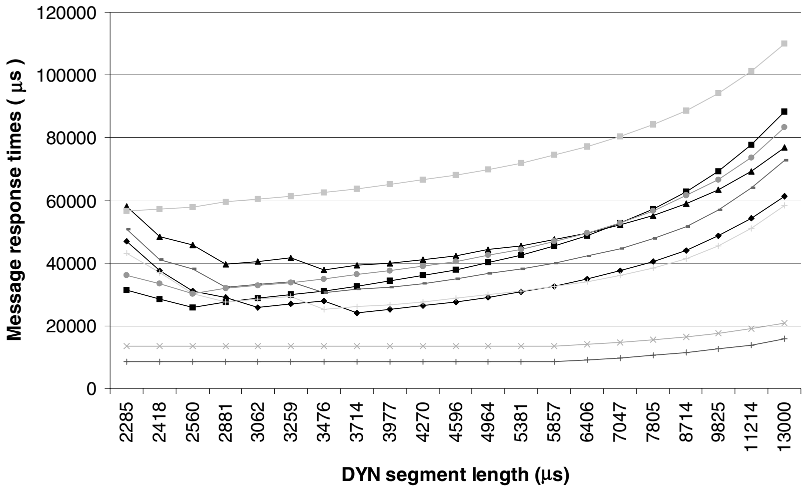


Figure 7.6: Influence of DYN Segment Length on Message Response Times

exhaustively perform the scheduling and schedulability analysis for all possible values of the DYN segment length, we will evaluate response times for only a small number of points and use a curve fitting approach to extrapolate the response time corresponding to all other points.

The algorithm in Figure 7.7 stores, in the set *Points,* the characteristics (DYN segment length, message response times, cost function) for a reduced set of bus configurations, The set initially contains only a small number (in our experiments we used five) of DYN segment sizes in the interval $[DYN_{bus}^{min}, DYN_{bus}^{max}]$ (line 1). For each alternative configuration in this initial set, the system is completely specified, allowing us to run our global scheduling and analysis algorithm, in order to determine the worst-case response times of all tasks and messages and the corresponding values of the cost function (line 4).

The algorithm maintains a data structure similar to the one presented in Table 7.1. Each cell (*i*,*j*) in the table stores the response time of the DYN message *i* when the DYN segment has value *j*. If the value *j* is in the set *Points*, then all the values on that particular column are computed exactly,

```
Determine_DYN_segment_length()
1  Points = {$DYN_{bus}^{min}$, $DYN_{bus}^{1}$, $DYN_{bus}^{2}$,..., $DYN_{bus}^{max}$}
2  for $DYN_{bus}$ in Points do
3      set current DYN segment length to $DYN_{bus}$
4      GlobalSchedulingAlgorithm()
5      Store message response times $R_m$( $DYN_{bus}$ )
6      Compute and store schedulability degree DSch($DYN_{bus}$)
7  end for
8  do
9      for $DYN_{bus}$ = $DYN_{bus}^{min}$ to $DYN_{bus}^{max}$ step gdMinislot do
10         if $DYN_{bus}$ is not in Points then
11             foreach DYN msg m do
12                 interpolate $R_m$($DYN_{bus}$) based on $R_m$(Points)
13             endfor
14             Compute and store MsgDSch($DYN_{bus}$)
15         endif
16     end for
17     select $DYN_{bus} \in Points$ with minimum stored $DSch_{min}$
18     if $DSch_{min} \leq 0$ then
19         return $DYN_{bus}$ -- system is schedulable
20     endif
21     select $DYN_{bus} \notin Points$ with minimum $MsgDSch_{min}$($DYN_{bus}$
22     set current DYN segment length to $DYN_{bus}$
23     GlobalSchedulingAlgorithm()
24     Compute and store DSch($DYN_{bus}$)
25     Add $DYN_{bus}$ to Points
26     if $DSch \leq 0$ then
27         return $DYN_{bus}$ -- system is schedulable
28     endif
29 while not termination condition
30 return infeasible $DYN_{bus}$
```

Figure 7.7: Determining the Size of the DYN segment using interpolation

DYN_{bus}	DYN_{bus}^{min}		...		DYN_{bus}^{max}
R_{m1}	C	I	C	I	C
R_{m2}	C	I	C	I	C
...	C	I	C	I	C
R_{mp}	C	I	C	I	C
DSch	C	-	C	-	C
MsgDSch	C	I	C	I	C

Table 7.1: Data structure used for interpolation

using the global scheduling and analysis algorithm (these values are marked with *C* in the table). For the DYN segment values that are not in the set *Points*, these values are interpolated and are marked with an *I* in the table. The last two lines of the table contain the degree of schedulability *DSch* and a variation of the same cost function (Equation (3.17)), called *MsgDSch* in which only the DYN messages are considered:

$$MsgDSch = \begin{cases} f_1 = \sum\limits_{\tau_{ij} \in DYNmsgs} max(R_{ij} - D_{ij}, 0), if f_1 > 0 \\ f_2 = \sum\limits_{\tau_{ij} \in DYNmsgs} (R_{ij} - D_{ij}), if f_1 = 0 \end{cases} \tag{7.2}$$

The values *DSch* and *MsgDSch* are computed using the message worst-case response time in each column, regardless of the fact that those response times are computed with the global scheduling and analysis algorithm. However, the table denotes with *C* those values of *MsgDSch* that are relying on message response times that are determined using the schedulability analysis, and with *I* the values of *MsgDSch* that are based on message response times that are determined using the interpolation algorithm (*I*). The global cost function *DSch* can be computed only for the situations when the schedulability analysis has been run (*C*), otherwise the table stores no value since there is no information about the worst-case response times of the tasks that are executed on each node in the system.

The algorithm presented in Figure 7.7 tries to find that length of the DYN segment for which the system is schedulable (i.e. find that DYN_{bus} for which $DSch(DYN_{bus}) \leq 0$). For all possible values of the size of the DYN segment (line 9) that have not been evaluated yet (line 10), the response times of messages in the system are computed using an interpolation algorithm based on a Newton polynomial. Considering a given DYN message m and a value DYN_{bus} then we denote with $R_m(DYN_{bus})$ the value of the worst-case response time of message m when the length of the dynamic segment of the bus is DYN_{bus}. Since for the values $DYN_{bus} \in Points$ we can compute accurately the values of the response times, then we will use these values to rapidly interpolate the values $R_m(DYN_{bus})$ for any $DYN_{bus} \notin Points$. Each time a new point is added to the set *Points*, the interpolated values of R_m are recomputed (line 12).

In each step of the algorithm, after the Table 7.1 has been filled up with computed or interpolated values, the algorithm selects that point $DYN_{bus} \in Points$ that leads to the system configuration with the best schedulability degree $DSch_{min}$ (i.e. minimum cost function) (line 17). If the associated $DSch_{min} \leq 0$ then the system is schedulable and the cost function is based on exact schedulability analysis, meaning that we have found the desired result (line 19).

If none of the investigated configurations captured by the set *Points* leads to a schedulable system, then the algorithm selects that value $DYN_{bus} \notin Points$ with the smallest $MsgDSch_{min}$ (line 21). The selection process uses the function expressed in Equation (7.2) which favours bus configurations that lead to smaller response times for the DYN messages in the system. The selected configuration is added to the set *Points* and evaluated (lines 22-25). If the evaluation of the resulted architecture leads to a cost function $DSch_{min} \leq 0$ then the algorithm returns the DYN_{bus} value as the one that produces a schedulable system (line 26). Otherwise, the algorithm repeats the above steps (approximation of response times, selection and evaluation of another DYN bus size) until either a schedulable solution is found, or when a certain *termination condition* is met. This condition is that a certain number N_{max} of iterations have been performed without finding a schedulable solution and without any improvement of the cost function (in our experiments we had N_{max}=10).

7.4 Simulated Annealing Based Approach

We have also implemented an approach based on simulated annealing [Kir83] for bus access optimisation. The SA heuristic explores the design space performing the following set of moves:

- *gdNumberOfStaticSlots* is incremented or decremented, inside the allowed limits (when an ST slot is added, it is allocated randomly to a node; when an ST slot is removed, the operation has to be done in such a way that each node that transmits ST messages will still have a ST slot allocated to it);
- *gdStaticSlot* is increased or decreased with 20 x *gdBit*, inside the allowed limits;
- the assignment of ST slots to nodes is changed by re-assigning a randomly selected ST slot from a node N_1 to another node N_2. We also use in this context a similar transformation that switches the allocation of two ST slots, $FrameID_1$ and $FrameID_2$, used by two nodes N_1 and N_2 respectively;
- the assignment of DYN slots to messages is modified by switching the slots used by two DYN messages.

In Section 7.5 we used extensive, time consuming runs with the Simulated Annealing approach, in order to produce a reference point for the evaluation of our greedy heuristic.

7.5 Evaluation of FlexRay Bus Optimisation Heuristics

In order to evaluate our optimisation algorithms we generated 7 sets of 25 applications representing systems of 2 to 5 nodes respectively. We considered 10 tasks mapped on each node, leading to applications with a number of 20 to 50 tasks. Depending on the mapping of tasks, each such system had up to 50 additional nodes in the application task graph due to the communication tasks. The tasks were grouped in task graphs of 5 tasks each. Half of the tasks in each system were time triggered and half were event triggered. The execution times were generated in such a way that the utilisation on each node was between 30% and 60% (similarly, the message transmission times were generated so that the bus utilisation was between 10% and 70%). All experiments were run on an AMD Athlon 2400+ PC.

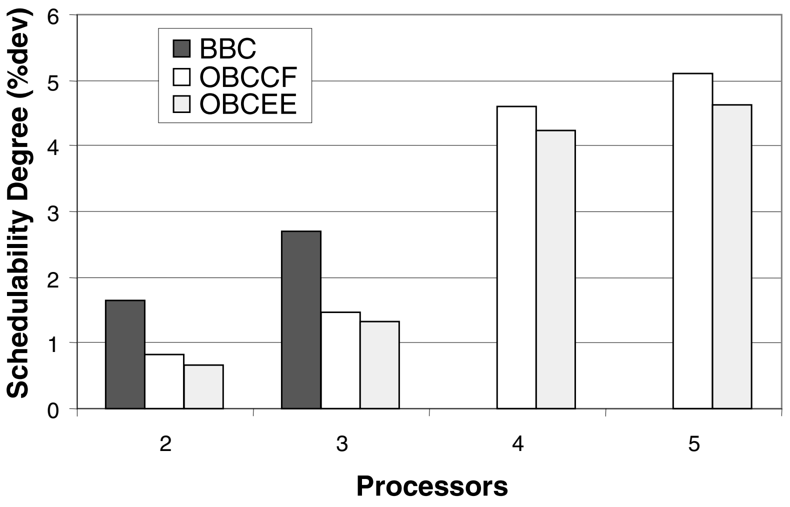
We have performed the bus optimisation using four approaches:

1. Basic Bus Configuration BBC (Section 7.2);
2. OBCCF - the OBC heuristic with the curve fitting procedure (Section 7.3);
3. OBCEE - the OBC heuristic with an exhaustive exploration of the sizes for the DYN segment; and
4. SA - the Simulated Annealing based heuristic.

Figure 7.8 shows the results obtained after running our algorithms on the generated applications. On the upper side of the figure one can see the average percentage deviation for the cost function obtained with BBC, OBCCF and OBCEE respectively, relative to the cost function obtained with SA. On the diagram in the lower part of the figure we present the computation times required by each algorithm (except the running times for SA, which are inherently large and were left out in order to provide a better view of the results).

One can notice that the BBC approach runs in almost zero time, but it fails to find any schedulable configurations for systems with more than 3 processors. On the other hand, the other approaches continue to find schedulable solutions even for larger systems. Comparing OCCF and OBCEE, we can observe that both produce results which are very close to the reference values produced by SA (max. 4-5% deviation). OBCCF generates results which are less than 0.5% away from those produced by OBCEE, but with a run time that is up to 2 orders of magnitude smaller. This confirms the high efficiency of our curve fitting based optimisation.

Finally, we considered the real-life example implementing a vehicle cruise controller described in Appendix C. We have partitioned the functionality in such a way that two of the task graphs were time triggered and the other two were event triggered. Configuring the system using the BBC approach took less than 5 seconds but resulted in an unschedulable system. Using the OBCCF approach took 137 seconds, while the OBCEE required 29 minutes. The cost function obtained by OBCCF was 1.2% larger in the solution obtained with OBCEE. In both cases the selected bus configuration resulted in a schedulable system.



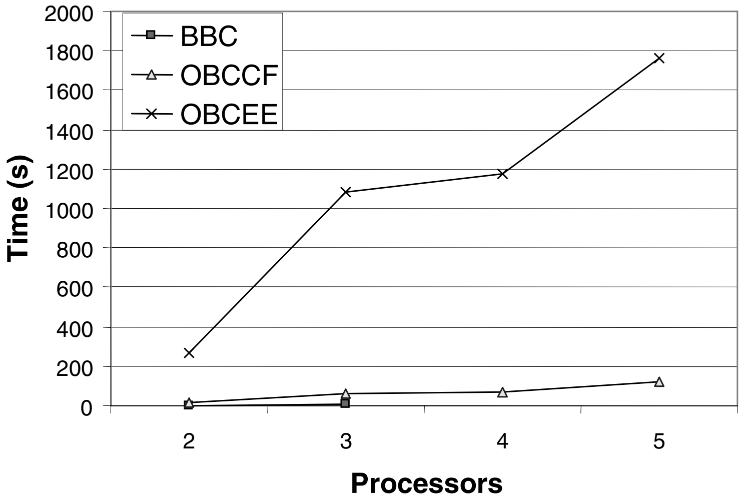


Figure 7.8: Evaluation of Bus Optimisation Algorithms

7.6 Conclusions

In this chapter, we have shown the importance of finding a FlexRay bus configuration that is customised according to the particular needs of the application. We have also proposed and evaluated three different heuristics that are able to generate such a configuration.

Chapter 8 Conclusions and Future Work

8.1 Conclusions

In this thesis we have studied several problems that appear in distributed embedded systems that support heterogeneous scheduling policies and communication protocols.

We have first defined and solved the problem of scheduling heterogeneous ET/TT distributed embedded systems that rely on a combination of static cyclic scheduling and EDF-within-priorities scheduling. For the communication we have assumed a bus modelled using the Universal Communication Model, and we have shown how the communication delays can be accounted for during the scheduling and schedulability analysis of the entire system.

Once we have solved the scheduling and schedulability analysis problem, we addressed several design problems which we identified as specific for the type of heterogeneous systems we are studying: the assignment of scheduling policies to the activities in the system, and the synthesis of parameters of the heterogeneous ST/DYN communication protocol. These two aspects have been approached in a larger design context, which also involved mapping and scheduling of the functionality. Our experiments have shown the importance of each of the optimisation aspects taken into consideration and the efficiency of the proposed optimisation heuristic.

The second part of the thesis concentrates on similar problems, by taking the analysis and optimisation methodology described in the first part of the thesis and applying it on a particular case of systems that use FlexRay as a communication protocol.

For each problem we have run extensive experiments, based on synthetic applications and a real-life example, that were used to evaluate the efficiency of our solutions.

8.2 Future Work

While the work presented in this thesis may seem complete, we consider it only as the foundation that has to be considered in the future, when looking for solutions to more complex problems. There are several possible extensions to our work that a careful reader can already imagine:

First, in this thesis we have concentrated on distributed architectures interconnected by a single bus. While the main difficulty regarding the communication aspects in our study was represented by the heterogeneity of the communication protocol, the system architecture that has been considered is somewhat simplistic. The growing complexity of nowadays embedded applications and the limitations on the number of nodes that can be connected to one bus are two main reasons causing the current distributed architectures to actually rely not only on one single bus, but on a large set of buses, interconnected by gateways in a hierarchical manner (see for example in Figure 8.1 an architecture consisting of four node clusters). Such architectures represent complex extensions of our studied problems, and their investigation is worth being considered. Similar bus interconnections have been studied in [PopP06], but the communication protocols inside each cluster were not heterogeneous in the sense defined in our work. While our timing analysis for heterogeneous protocols can determine worst-case response times for messages transmitted over one bus, it does not provide yet the generalisations needed to cover hierarchical bus infrastructures. The work presented in [PopP06] and in this thesis can

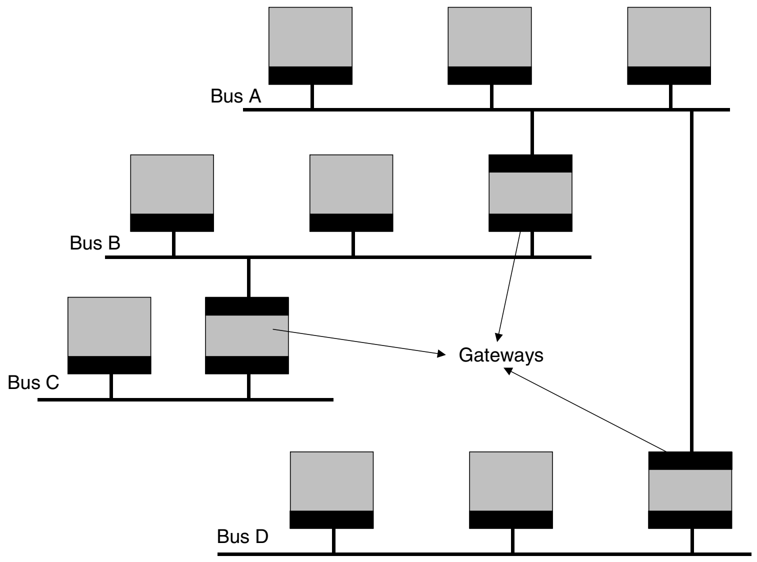


Figure 8.1: Hierarchical Bus-based Infrastructure

therefore represent a good starting point for studying the worst-case timing scenarios in the more general context of a system architecture based on a hierarchy of buses.

A second possible direction of study is arising from the fact that the hierarchy of schedulers that has been considered on each node in the system is fixed. While the hierarchy that we chose may serve well for implementing a wide range of hard real-time applications, it may also be considered a restricted model. Consequently, system models that allow for a flexible specification of the scheduler hierarchies while permitting for a straightforward timing analysis could represent an important generalisation of our work.

Finally, another direction of study could explore the case when our optimisation approach is not able to produce a schedulable system under the limited set of resources specified in the initial architecture. In such situations, the most common solution for improving the timing properties of the system is to add new hardware resources to the existing configuration. Allocating more nodes and buses to the system architecture is in itself a complex problem, since it will bring up new questions, like:

- how many new buses need to be added to the communication infrastructure?
- how many processing nodes should be added to the system?
- to what bus should the new nodes be connected?

Increasing the number of buses usually adds more bandwidth to the entire communication support. Similarly, adding more nodes in the system will increase its processing power. However, such solutions have also drawbacks that need to be taken into consideration during the design process. Additional resources almost always increase the cost of the application, and one has to take care when extending the hardware architecture, in order to avoid unnecessary or significant increases of the final cost.

One particular drawback of the growing amount of buses inside a vehicle is represented by the large weight produced by such an increased number of wires. This represents a more and more important problem in application areas like automotive electronics, where the efficiency of the product is heavily influenced by the physical properties of the system. As a consequence, future systems may be forced to consider also solutions based on wireless communication protocols, meaning that investigating the real-time properties of such protocols could represent a viable research topic.

Appendix A
List of Notations

Application Model notations

Notation	**Description**
Γ_i	Task-graph i
τ_{ij}	Activity j belonging to task-graph Γ_i
T_{ij}	Period of activity τ_{ij}
C_{ij}	Worst-case execution time of activity τ_{ij}
D_{ij}	Deadline for activity τ_{ij}
Φ_{ij}	Offset for activity τ_{ij} measured from the release of the event that initiates the task-graph Γ_i
J_{ij}	Jitter for activity τ_{ij}
B_{ij}	Blocking time for activity τ_{ij}
R^b_{ij}	Best-case response time for activity τ_{ij}
R_{ij}	Worst-case response time for activity τ_{ij}
$Prio_{ij}$	Priority for activity τ_{ij}
$\mathcal{M}(\tau_{ij})$	The node on which task τ_{ij} is mapped
T_{ss}	LCM of the periods of all the task-graphs in the application

Design Optimisation notations

Notation	**Description**
Ψ^{P}	The set of task-graphs which can be repartitioned between time-triggered and event-triggered domains
Ψ^{M}	The set of tasks which can be remapped
$\mathcal{P}$	The set of tasks in the system
$\mathcal{P}_{SCS}$	The set of tasks in the system that are assigned to SCS by the designer
$\mathcal{P}_{FPS}$	The set of tasks in the system that are assigned to FPS by the designer
$\mathcal{P}_{EDF}$	The set of tasks in the system that are assigned to EDF by the designer
$\mathcal{P}^{M}$	The set of tasks in the system that are mapped by the designer
$\mathcal{P}^{*}$	The set of tasks in the system that can be remapped during design optimisation
$\mathcal{A}$	System application
$\mathcal{N}$	Number of nodes in the distributed architecture
$\mathcal{M}$	Mapping function that assigns each task in the system to one of the nodes in the architecture
$\mathcal{B}$	Bus configuration
$\mathcal{S}$	Scheduling policy assignment
$DSch$	Degree of schedulability for all system activities
$MsgDSch$	Degree of schedulability for DYN messages

Scheduling and Schedulability Analysis notations

Notation	**Description**
$R_{abc}(p)$	Response time of the p-th job of a FPS task τ_{ab}, in a busy window initiated by task τ_{ac}
$R^{A}{}_{abc}(p)$	Response time of the p-th job of an EDF task τ_{ab}, in a busy window starting at time A, with a critical instant initiated by task τ_{ac}
φ_{ijk}	Phase between event arivals and the critical instant of task τ_{ij}
p	A job of task under analysis τ_{ab}
$W_{ik}(\tau_{ab}, t)$ $W^{FP}{}_{ik}(\tau_{ab}, t)$	Worst-case interference produced by strictly higher priority activities in task graph Γ_i over the execution of the FPS task τ_{ab}, when the critical instant is initiated by τ_{ik}
$W_i(\tau_{ab},t)$ $W^{FP}{}_i(\tau_{ab},t)$	Worst-case interference produced by strictly higher priority activities in task graph Γ_i over the execution of the FPS task τ_{ab}
$W^{EDF}{}_{ik}(\tau_{ab}, t)$	Worst-case interference produced by equal priority activities in task graph Γ_i over the execution of the EDF task τ_{ab}, when the critical instant is initiated by τ_{ik}
$W^{EDF}{}_i(\tau_{ab},t)$	Worst-case interference produced by equal priority activities in task graph Γ_i over the execution of the EDF task τ_{ab}
$A_{ij}(t)$	ET availability over an interval of time t, associated with the busy period of activity τ_{ij}
$H_{ij}(t)$	ET demand associated with the busy period of length t of activity τ_{ij}

FlexRay notations

Notation	Description
gdCycle	Length of the bus cycle (T_{bus})
gNumber-OfMinislots	Length of DYN segment, expressed in minislots
gdMinislot	Length of the minislot
STb_{us}	Length of the ST segment in time units
DYN_{bus}	Length of the DYN segment in time units
gdStaticSlot	Length of a ST slot in time units
pLatestTx	The largest value of the minislot counter that allows the start of a DYN transmission
$FrameID_m$	The frame identifier for a message m
$hp(m)$	Set of DYN messages sent by the same node that transmits m, having the same $FrameID_m$ but a higher local priority than m
$lf(m)$	Set of DYN messages that have lower frame identifiers than $FrameID_m$
R_m	Worst-case response time for a message m

Appendix B
Bin-Covering Heuristics

In this appendix we briefly present the bin covering heuristics that we used in our timing analysis. These heuristics are presented in more detail in [Lab95].

The bin covering problem considers a set of n items of weights $w_1..w_n$ and an unlimited number of bins of infinite capacity. The target is to fill as many bins as possible with a minimum capacity C_{min}, using the n given items.

The heuristics presented below determine upper bounds for a given instance of the bin covering problem. All the following operations assume that the items are sorted in decreasing order of their weights:

$$w_1 \geq w_2 \geq \ldots \geq w_n .$$

Before computing the upper bounds, the list of items is first processed based on the following reduction criteria:

1. Any item with $w_j > C_{min}$ can be assigned alone to a bin. We denote with R_1 the number of such items and we eliminate them from the list of items.
2. If two items k and l satisfy the condition $w_k + w_l = C_{min}$ then there exists an optimal solution in which a bin contains only items k and l. We denote with R_2 the number of bins that can be filled in this way and we remove from the list the items that satisfy this reduction criterion.

3. Let k be the maximum index such that:

$$w_1 + \sum_{j=k}^{n} w_j \geq C_{min} .$$

If $w_1 + w_k \geq C_{min}$ then there exists an optimal solution in which a bin contains only items 1 and k. The number of bins that can be filled in this way is denoted with R_3 and the items that fill these bins according to the 3rd criterion are removed from the list.

In a second step, we use the n' remaining items to fill bins with at least C_{min}. The heuristics presented bellow will determine upper bounds for the maximum value we are looking for.

1. Since no two remaining items can fill a bin up to C_{min}, then $U_0 = \left\lfloor \frac{n'}{2} \right\rfloor$.

2. The continuous relaxation of the problem gives another bound:

$$U_1 = \left\lfloor \sum_{j=1}^{n'} \frac{w_j}{C_{min}} \right\rfloor$$

3. Let $t = \min\{s\colon \sum_{h=s}^{n'} w_h < C_{min}\}$ and define $p(j) = \min\{p\colon \sum_{h=j}^{j+p} w_h \geq C_{min}\}$ for $j = 1,\ldots,\ \tau = min\left(\left\lfloor \frac{n'}{2} \right\rfloor, t-1\right)$.

Then, for $k = 0, 1,\ldots\tau$, $U_2(k) = k + \left\lfloor \sum_{j=k+1}^{n'-\alpha(k)} \frac{w_j}{C_{min}} \right\rfloor$ where $\alpha(0) = 0$ and $\alpha(k) = \sum_{j=1}^{k} p(j)$ for $k > 0$, is a valid upper bound for the bin covering problem that considers the reduced set of items. We denote with U_2 the minimum of all these values: $U_2 = min(U_2(k)), k = 1\ldots\tau$.

4. Let $\beta(j)$ be the smallest index k such that $w_j + \sum_{k=1, k\neq j}^{\beta(j)} w_k \geq C_{min}$ and define $q(j) = \beta(j)$ if $\beta(j) > j$, $q(j) = \beta(j) + 1$ otherwise. Then $U_3 = \left\lfloor \sum_{j=1}^{n'} \frac{1}{q(j)} \right\rfloor$ is a valid upper bound for the bin covering problem using the reduced set of items.

The upper bound we are looking for is determined as the minimum of the four values U_0, U_1, U_2, U_3:

$$U = \min(U_0, U_1, U_2, U_3).$$

This result can be further improved as follows: given an upper bound value U, if

$$U > \left\lfloor \frac{\sum_{j=1}^{n'} w_j - (3U - n')}{C_{min}} \right\rfloor$$

then U - 1 is a valid upper bound value.

Considering now the initial set of items, before the reductions in the first step, the upper bound *UB* of the maximum number of bins that can be filled with C_{min} is: $UB = R_1 + R_2 + R_3 + U$.

Appendix C
Real-Life Example

During our experiments we have also used a safety critical application, with hard real-time constraints, implementing a vehicle cruise controller (CC). As described in [PopP04a], such an application has to deliver the following functionality:

- "*It maintains a constant speed for speeds over 35 km/h and under 200 km/h.*
- *offers an interface (buttons) to increase/decrease the reference speed, and*
- *is able to resume its operation at the previous reference speed.*
- *The CC operation is suspended when the driver presses the brake pedal.*"

The process graph that models the CC has 32 processes and 22 messages grouped in 4 task graphs.

Figure C.1 shows the structure of the four task graphs that compose the real life example. In the task graphs, the activities are marked with big circles for tasks and small black circles for messages. All tasks have periods and deadlines equal to 460 ms. For each task we show the worst-case execution time, and for each message we show its worst case transmission time.

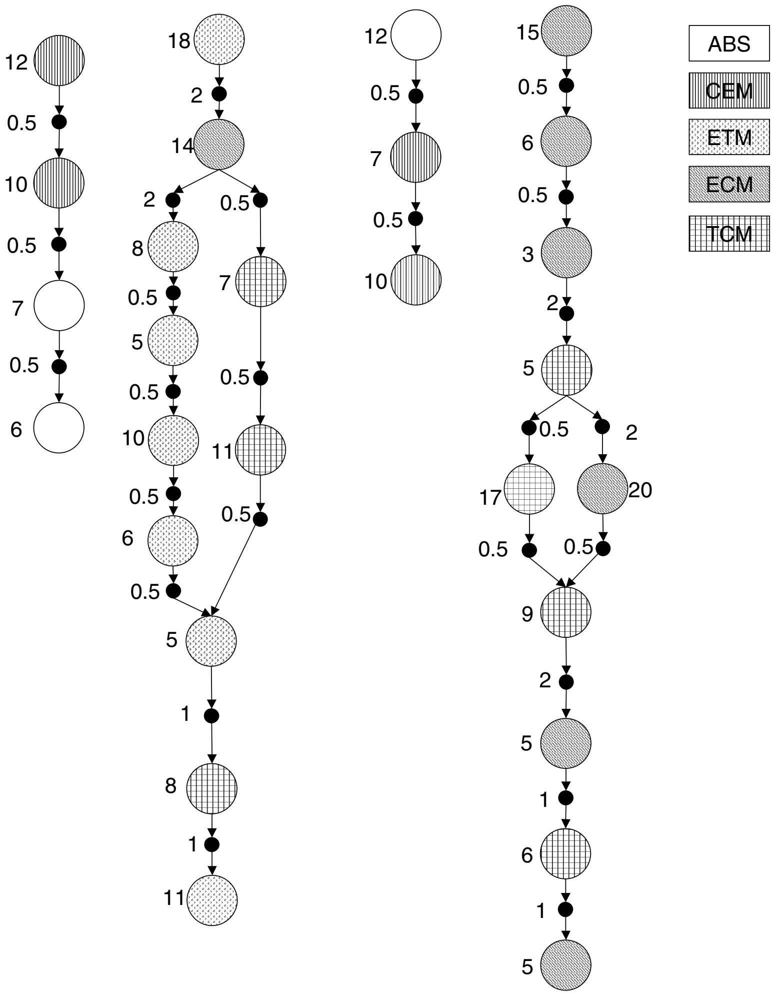


Figure C.1: Real-Life Example

The CC has a functionality grouped in five modules -- Electronic Throttle Module (ETM), Anti-lock Breaking System (ABS), Engine Control Module (ECM), Central Electronic Module (CEM) and Transmission Control Module (TCM) --, that are mapped on an architecture consisting of five nodes. Figure C.1 also shows the mapping of tasks on the five nodes in the architecture.

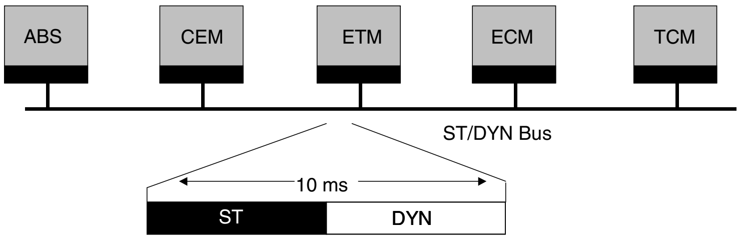


Figure C.2: Architecture of the Cruise Controller

As shown in Figure C.2, the nodes in the architecture are interconnected by a heterogeneous ST/DYN bus, with 50% ST and 50% DYN bandwidth. The period for the bus access cycle is 10 ms.

Appendix D
List of Figures

References

[ARI629] ARINC, “Multi-Transmitter Data Bus, Part 1, Technical Description”, ARINC Document 629P1-4, Aeronautical Radio, Inc., Annapolis, MD, USA, 1995.

[Abd99] T. F. Abdelhazer, K. G. Shin, “Combined Task and Message Scheduling in Distributed Real-Time Systems”, IEEE Transactions on Parallel and Distributed Systems, 10(11), pages 1179-1191, 1999.

[Agr94] G. Agrawal, B. Chen, W. Zhao, S. Davari, “Guaranteeing Synchronous Message Deadlines with the Timed Token medium Access Control Protocol”, IEEE Transactions on Computers, 43(3), pages 327-339, 1994.

[Alm99] L. Almeida, “Flexibility and Timeliness in Fieldbus-based Real-Time Systems“, Ph. D. Thesis, University of Aveiro, Portugal, 1999.

[Alm02] L. Almeida, P. Pedreiras, J. A. G. Fonseca, “The FTT-CAN Protocol: Why and How”, IEEE Transactions on Industrial Electronics, 49(6), pages 1189-1201, 2002.

[And01] B. Andersson, S. Baruah, J. Jonsson, “Static-Priority Scheduling on Multiprocessors”, Proceedings of the IEEE Real-Time Systems Symposium, pages 193-20, 2001.

[Ass84] S.F. Assman, D.S. Johnson, D.J. Kleitman, J.Y.-T. Leung, “On a Dual Version of the One-Dimensional Bin Packing Problem”, *Journal of Algorithms*, 5, pages 502–525, 1984.

[Aud93] N. Audsley, K. Tindell, A. et. al., “The End of Line for Static Cyclic Scheduling?”, 5th Euromicro Workshop on Real-Time Systems, 1993.

[Aud95] N. Audsley, A. Burns, et. al., “Fixed Priority Preemptive Scheduling: An Historical Perspective”, Real-Time Systems, 8(2/3), 1995.

[Bal97] F. Balarin, editor, Hardware-Software Co-Design of Embedded Systems: The Polis Approach, Kluwer Academic Publishers, 1997.

[Bal98] F. Balarin, L. Lavagno, et. al., “Scheduling for Embedded Real-Time Systems”, IEEE Design and Test of Computers, January-March,1998.

[Bar98] S. Baruah, “A General Model for Recurring Real-Time Tasks”, Proceedings of the IEEE Rel-Time Systems Symposium, pages 114-122, 1998.

[Bec98] J. E. Beck, D. P. Siewiorek, “Automatic Configuration of Embedded Multicomputer Systems”, IEEE Transactions on CAD, 17(2), pages 84-95, 1998.

[Ben02] L. Benini, G. De Micheli, “Networks on Chips: A New SoC Paradigm”, IEEE Computer, 35(1), pages 70-78, 2002.

[Ben05] L. Benini, D. Bertozzi, “Network-on-Chip Architectures and Design Methods”, IEEE Proceedings of Computers and Digital Techniques, 152(2), pages 261-272, 2005.

[Ber03] J. Berwanger, M. Peller, R. Griessbach, “A New High-Performance Data Bus System for Safety Related Applications”, http://www.byteflight.com, 2003.

[Bin01] Enrico Bini, Giorgio Butazzo, Giuseppe Butazzo, “A Hyperbolic Bound for the Rate Monotonic Algorithm”, Proceedings of the 13th Euromicro Conference on Real-Time Systems, pages 59-66, 2001.

[Bli98] T. Blicke, J. Teich, L. Thiele, “System-Level Synthesis using Evolutionary Algorithms”, Design Automation for Embedded Systems, 4(1), pages 23-58, 1998.

[Bos91] R. Bosch GmbH, “CAN Specification Version 2.0”, 1991.

[But97] Giorgio C. Butazzo, “Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications”, Kluwer Academic Publishers, 1997.

[But05] G. Butazzo, “Rate Monotonic vs EDF: Judgement Day”, Real-Time Systems, 29(1), 2005.

[Cen04] G. Cena, A. Valenzano, “Performance analysis of Byteflight networks”, *Proceedings of the IEEE International Workshop on Factory Communication Systems*, pages 157–166, 2004.

[Cof72] E.G. Coffman Jr., R.L. Graham, “Optimal Scheduling for two Processor Systems”, *Acta Informatica*, 1, 1972.

[Dav99] B. P. Dave, G. Lakshminarayana, N. K. Jha, “COSYN: Hardware-Software Co-Synthesis of Heterogeneous Distributed Embedded Systems”, IEEE Transactions on VLSI Systems, pages 92-104, 1999.

[Dav07] R.I. Davis, A. Burns, R.J. Bril, J.J. Lukkien, "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised", Real-Time Systems Journal, 35(3), pages 239-272, April 2007.

[Dem01] T. Demmeler, P. Giusto, "A Universal Communication Model for an Automotive System Integration Platform", Design, Automation and Test in Europe (DATE'01) Conference, Munich, pages 47-54, 2001.

[Din05] S. Ding, N. Murakami, H. Tomiyama, H. Takada, "A GA-Based Scheduling Method for FlexRay Systems", *Proceedings of EMSOFT*, 2005.

[Dob01a] R. Dobrin, G. Fohler, "Implementing Off-Line Message Scheduling on Controller Area Network (CAN)", Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation, 1, 2001.

[Dob01b] R. Dobrin, G. Fohler, P. Puschner, "Translating Offline Schedules into Task Attributes for Fixed Priority Scheduling", Proceedings of Real-Time Systems Symposium, 2001.

[Eche07] Echelon, *LonWorks: The LonTalk Protocol Specification*, http://www.echelon.com, 2007.

[Eke00] C. Ekelin, J. Jonsson, "Solving Embedded System Scheduling Problems using Constraint Programming", Chalmers University of Technology, Sweden, Report number TR 00-12, 2000.

[Ele00a] P. Eles, A. Doboli, P. Pop, Z. Peng, "Scheduling with Bus Access Optimization for Distributed Embedded Systems", IEEE Transactions on VLSI Systems, 8(5), pages 472-491, 2000.

[Ele02] P. Eles, Lecture Notes for System Design and Methodology, http://www.ida.liu.se/~TDTS30, 2002.

[Erm97] H. Ermedahl, H. Hansson, M. Sjödin, “Response Time Guarantees in ATM Networks”, Proceedings of Real-Time Systems Symposium, 1997.

[Ern98] R. Ernst, “Codesign of Embedded Systems: Status and Trends”, IEEE Design&Test of Comp., April-June, 1998.

[Edw97] S. Edwards, L. Lavagno, E. A. Lee, A. Sangiovanni-Vincentelli, “Design of Embedded Systems: Formal Models, Validation and Synthesis”, Proceedings of the IEEE, 85(3), pages 366-390, 1997.

[Edw00] S. Edwards, “Languages for Digital Embedded Systems”, Kluwer Academic Publishers, 2000.

[Fan05] X. Fan, M. Jonsson, “Guaranteed Real-Time Services oveer Standard Switched Ethernet”, Proceedings of the 30th IEEE Conference on Local Computer Networks, 2005.

[Fle07] FlexRay homepage: http://www.flexray-group.com, 2007.

[Fuh00] T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, M. Walther, Robert Bosch GmbH, “Time-Triggered Communication on CAN (Time-Triggered CAN- TTCAN)”, Proceedings of International CAN Conference, Amsterdam, The Netherlands, 2000.

[Gon91] M. González Harbour, M. H. Klein, J. P. Lehoczky, “Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority, Proceedings of 12th IEEE Real-Time Systems Symposium, pages 116 - 128, 1991.

[Gon03] M. Gonzaléz Harbour, J. C. Palencia, “Response Time Analysis for Tasks Scheduled under EDF within Fixed Priorities”, Proceedings of Real-Time Systems Symposium, pages 200–209, 2003.

[Gut95] J. J. Gutiérrez García, M. González Harbour, “Optimized Priority Assignment for Tasks and Messages in Distributed Hard Real-Time Systems”, Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems, Santa Barbara, pages 124-132, 1995.

[Ham05] A. Hamann, R. Ernst, “TDMA Time Slot and Turn Optimization with Evolutionary Search Techniques”, *Proceedings of the Design, Automation and Test in Europe Conference*, Volume 1, pages 312–317, 2005.

[Han97] H. Hansson, M. Sjödin, K. Tindell, “Guaranteeing Real-Time Traffic Through an ATM Network”, Proceedings of the 30th Hawaii International Conference on System Sciences, 5, 1997.

[Hoy92] K. Hoyme, K. Driscoll, “SAFEbus”, *IEEE Aerospace and Electronic Systems Magazine*, 8(3), pages 34–39, 1992.

[Hu03] J. Hu, R. Marculescu, “Exploiting the Routing Flexibility for Energy/Performance-Aware Mapping of Regular Architectures”, Proceedings of Design, Automation and Test in Europe, pages 10688, 2003.

[ISO02] International Organization for Standardization, “Road vehicles-Controller Area Network (CAN)—Part 4: Time-triggered communication”, ISO/DIS 11898–4, 2002.

[Jer04] A. Jerraya, W. Wolf (editors), “Multiprocessor Systems-on-Chips“, Morgan Kaufman, 2004.

[Jon97] J. Jonsson, K. J. Shin, “A Parametrized Branch-and-Bound Strategy for Scheduling Precedence-Constrained Tasks on a Multiprocessor System“, Proceedings of the International Conference on Parallel Processing (ICPP), pages 158-165, 1997.

[Jor97] P. B. Jorgensen, J. Madsen, “Critical Path Driven Cosynthesis for Heterogeneous Target Architectures”, Proceedings of the 5th International Workshop on Hardware-Software Co-design, pages 15-19, 1997.

[Keu00] K. Keutzer, S. Malik, A. R. Newton, “System-Level Design: Orthogonalization of Concerns and Platform-Based Design”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19(12), 2000.

[Kir83] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, “Optimisation by simulated annealing”, *Science*, 220, pages 671-680, 1983.

[Koo02] P. Koopman, “Critical Embedded Automotive Networks”, IEEE Micro, 22(4), pages 14-18, 2002.

[Kop92] H. Kopetz, G. Fohler, G. Grünsteidl, H. Kantz, G. Pospischil, P. Puschner, J. Reisinger, R. Schlatterbeck, W. Schütz, A. Vrchoticky, R. Zainlinger, “The Programmer’s View of MARS”, Proceedings of 13th IEEE Real-Time Systems Symposium, pages 223-226, 1992.

[Kop97] H. Kopetz, “Real-Time Systems - Design Principles for Distributed Embedded Applications”, Kluwer Academic Publisher, 1997.

[Kop03] H. Kopetz, G. Bauer, “The time-triggered architecture”, *Proceedings of the IEEE*, 91(1), pages 112–126, 2003.

[Kuc97] K. Kuchcinski, “Embedded System Synthesis by Timing Constraint Solving”, Proceedings of the International Symposium on System Synthesis, pages 50-57, 1997.

[Lab95] M. Labbe, G. Laporte, S. Martello, "An exact algorithm for the dual bin packing problem", *Operations Research Letters 17*, pages 9–18, 1995.

[Lav99] L. Lavagno, A. Sangiovanni-Vincentelli, E. Sentovich, "Models of Computation for Embedded System Design", A. A. Jerraya and J. Mermet eds: System Level Synthesis, Kluwer, 1999.

[Lee99] C. Lee, M. Potkonjak, W. Wolf, "Synthesis of Hard Real-Time Application Specific Systems", Design Automation for Embedded Systems, 4(4), pages 215-241, 1999.

[Leh89] J. Lehoczky, L. Sha, Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour", Proceedings of 11th Real-Time Systems Symposium, pages 166-171, 1989.

[Leh90] J. P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines", Proceedings of 11th IEEE Real-Time Systems Symposium, pages 201 -209, 1990.

[Leu82] J. Y. T. Leung, J. Whitehead, "On the Complexity of Fixed Priority Scheduling of Periodic, Real-Time Tasks", Performance Evaluation, 2(4), pages 237-250, 1989.

[LIN07] *Local Interconnect Network Protocol Specification*, http://www.lin-subbus.org, 2007.

[Lip04] G. Lipari, E. Bini, "A Methodology for Designing Hierarchical Scheduling Systems", Journal of Embedded Computing, 1(2), pages 257-269, 2004.

[Liu73] C. L. Liu, J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", Journal of the ACM, 20(1), pages 46-61, 1973.

[Liv98] M. A. Livani, J. Kaiser, “EDF Consensus on CAN Bus Access for Dynamic Real-Time Applications”, *Proceedings of the IPPS/SPDP Workshops*, pages 1088–1097, 1998.

[Loc92] C. Douglas Locke, “Software Architecture for Hard-Real Time Applications: Cyclic Executives vs. Fixed Priority Executives”, Journal of Real-Time Systems, 4, pages 37-53, 1992.

[Lön99] H. Lönn, J. Axelsson, “A Comparison of Fixed-Priority and Static Cyclic Scheduling for Distributed Automotive Control Applications”, Proceedings of the 11th Euromicro Conference on Real-Time Systems, pages 142-149, 1999.

[Mar00] G. Martin, “The Future of High-Level Modelling and System Level Design: Some Possible Methodology Scenarios”, 9th IEEE/DATC Electronic Design Processes Workshop, 2000.

[Mar03] P. Marwedel, “Embedded System Design”, Kluwer Academic Publishers, 2003.

[Mey03] T. Meyerowitz, C. Pinello, A. Sangiovanni-Vincentelli, “A tool for describing and evaluating hierarchical real-time bus scheduling policies”, *Proceedings of the Design Automation Conference*, pages 312–317, 2003.

[Mil99] D. Millinger, P. Gansterer, “The Time-Triggered Operating System Technical Manual”, http://www.vmars.tuwien.ac.at/~fstue/manuals/ttos/doku.html, March, 2003.

[Mic97] G. de Micheli, R. K. Gupta, “Hardware/Software Co-Design”, Proceedings of the IEEE, 85(3), pages 349-365, 1997.

[Min00] P. S. Miner, “Analysis of the SPIDER Fault-Tolerance Protocols”, *Proceedings of the 5th NASA Langley Formal Methods Workshop*, 2000.

[Mul04] K. D. Muller-Glaser et al., “Multiparadigm Modelling in Embedded Systems Design”, IEEE Transactions on Control Systems Technology, 12(2), pages 279-292, 2004.

[Nav05] N. Navet, Y. Song, F. Simont-Lion, C. Wilwert, “Trends in Automotive Communication Systems”, *Proceedings of the IEEE*, 93(6), pages 1204–1223, 2005.

[Nur04] J. Nurmi, H. Tenhunen, J. Isoaho, A. Jantsch (editors), “Interconnect-Centric Design for Advanced SoCs and NoCs”, Kluwer Academic Publishers, 2004.

[Pal98] J. C. Palencia, M. González Harbour, “Schedulability Analysis for Tasks with Static and Dynamic Offsets”, Proceedings of the 19th IEEE Real-Time Systems Symposium, pages 26-37, 1998.

[Pal99] J. C. Palencia, M. González Harbour, “Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems”, Proceedings of the 20th IEEE Real-Time Systems Symposium, 1999.

[Pal03] J. C. Palencia, M. Gonzaléz Harbour, “Offset-Based Response Time Analysis of Distributed Systems Scheduled under EDF”, Proceedings of the Euromicro Conference on Real-Time Systems, pages 3–12, 2003.

[Ple92] P. Pleinevaux, “An Improved Hard Real-Time Scheduling for IEEE 802.5”, Journal of Real-Time Systems, 4(2), 1992.

[Ped00] P. Pedreiras, L. Almeida, "Combining Event-Triggered and Time-Triggered Traffic in FTT-CAN: Analysis of the Asynchronous Messaging System", *Proceedings of the Workshop on Factory Communication Systems*, pages 67–75, 2000.

[Ped03] P. Pedreiras, L. Almeida, "The Flexible Time-Triggered(FTT) Paradigm: An Approach to QoS Management in Distributed Real-Time Systems", International Parallel and Distributed Processing Symposium, pages 123a, 2003.

[Ped05] P. Pedreiras, P. Gai, L. Almeida, G.C. Buttazzo, "FTT-Ethernet: A Flexible Real-Time Communication Protocol That Supports Dynamic QoS Management on Ethernet-Based Systems", IEEE Transactions on Industrial Informatics, 1(3), August 2005.

[PopP00a] Paul Pop, Petru Eles, Zebo Peng, Alexa Doboli, "Scheduling with Bus Access Optimization for Distributed Embedded Systems", *IEEE Transactions on VLSI Systems*, 8(5), pages 472–491, 2000.

[PopP00b] Paul Pop, Petru Eles, Zebo Peng, "Bus Access Optimization for Distributed Embedded Systems based on Schedulability Analysis", Design, Automation and Test in Europe (DATE'00), pages 567-574, 2000.

[PopP01a] Paul Pop, Petru Eles, Traian Pop, Zebo Peng, "An Approach to Incremental Design of Distributed Embedded Systems", Proceedings of the 38th Design Automation Conference (DAC), Las Vegas, USA, pages 450-455, 2001.

[PopP01b] Paul Pop, Petru Eles, Traian Pop, Zebo Peng, "Minimizing System Modification in an Incremental Design Approach", Proceedings of the 9th International Workshop on Hardware/Software Codesign (CODES 2001), Copenhagen, Denmark, pages 183-188, 2001.

[PopP03a] Paul Pop, Petru Eles, Zebo Peng, "Schedulability-Driven Communication Synthesis for Time-Triggered Embedded Systems", Real-Time Systems Journal, 24, pages 297-325, 2004.

[PopP03b] Paul Pop, Petru Eles, Zebo Peng, "Schedulability Analysis and Optimization for the Synthesis of Multi-Cluster Distributed Embedded Systems", Design, Automation and Test in Europe (DATE'03) Conference, Munich, Germany, pages 184-189, 2003.

[PopP04a] Paul Pop, Petru Eles, Zebo Peng, "Schedulability-Driven Communication Synthesis for Time-Triggered Embedded Systems", *Real-Time Systems Journal*, 24, pages 297–325, 2004

[PopP04b] Paul Pop, Petru Eles, Zebo Peng, Traian Pop, "Scheduling and Mapping in an Incremental Design Methodology for Distributed Real-Time Embedded Systems", IEEE Transactions on VLSI Systems, 12(8), August 2004.

[PopP05] Paul Pop, Petru Eles, Zebo Peng, "Schedulability-Driven Frame Packing for Multi-Cluster Distributed Embedded Systems", *ACM Transactions on Embedded Computing Systems*, 4(1), pages 112–140, 2005.

[PopP06] Paul Pop, Petru Eles, Zebo Peng, Traian Pop, "Analysis and Optimization of Distributed Real-Time Embedded Systems", ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 11, Issue 3, pages 593-625, July 2006.

[PopT02] Traian Pop, Petru Eles, Zebo Peng, “Holistic Scheduling and Analysis of Mixed Time/Event-Triggered Distributed Embedded Systems”, Proceedings of the 10th International Symposium on Hardware/Software Codesign (CODES 2002), Estes Park, Colorado, USA, pages 187-192, 2002.

[PopT03a] Traian Pop, Petru Eles, Zebo Peng, “Schedulability Analysis for Distributed Heterogeneous Time/Event-Triggered Real-Time Systems”, Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS 2003), Porto, Portugal, pages 257-266, July 2-4, 2003.

[PopT03b] Traian Pop. Petru Eles, Zebo Peng, “Design Optimization of Mixed Time/Event Triggered Distributed Embedded Systems”, CODES+ISSS 2003 (merged conference), Newport Beach, California, USA, October 1-3, pages 83-89., 2003.

[PopT05] Traian Pop, Paul Pop, Petru Eles, Zebo Peng, “Optimization of Hierarchically Scheduled Heterogeneous Embedded Systems”, Proceedings of 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pages 67–71, 2005.

[PopT06] Traian Pop, Paul Pop, Petru Eles, Zebo Peng, Alexandru Andrei, “Timing Analysis of the FlexRay Communication Protocol”, 18th Euromicro Conference on Real-Time Systems (ECRTS 06), Dresden, Germany, July 5-7, pages 203-213, 2006.

[PopT07a] Traian Pop, Paul Pop, Petru Eles, Zebo Peng, “Bus Access Optimisation for FlexRay-based Distributed Embedded Systems”, Design, Automation, and Test in Europe Conference (DATE'07), Nice, France, pages 51-62, 2007.

[PopT07b] Traian Pop, Paul Pop, Petru Eles, Zebo Peng, “Analysis and Optimisation of Hierarchically Scheduled Multiprocessor Embedded Systems”, International Journal on Parallel Programming (to appear).

[PopT07c] Traian Pop, Paul Pop, Petru Eles, Zebo Peng, Alexandru Andrei, “Timing Analysis of the FlexRay Communication Protocol”, Real-Time Systems Journal (to appear).

[Pra92] S. Prakash, A. Parker, “SOS: Synthesis of Application Specific Heterogeneous Multiprocessor Systems”, Journal of Parallel and Distributed Computers, 16, pages 338-351, 1992.

[Pro01] Profibus International, *PROFIBUS DP Specification*, http://www.profibus.com, 2007.

[Raj93] P. Raja, G. Noubir, “Static and Dynamic Polling Mechanisms for Fieldbus Networks”, ACM Operating Systems Review, 27(3), 1993.

[Ric02] K. Richter, R. Ernst, “Event Model Interfaces for Heterogeneous System Analysis”, Proceedings of Design, Automation and Test in Europe Conference (DATE’02), Paris, France, 2002.

[Ric03] K. Richter, M. Jersak, R. Ernst, “A Formal Approach to MpSoC Performance Verification”, IEEE Computer, 36(4), pages 60-67, 2003.

[Rus01] J. Rushby, “Bus Architectures for Safety-Critical Embedded Systems”, *Springer-Verlag Lecture Notes in Computer Science*, 2211, pages 306–323, 2001.

[SAE94] SAE Vehicle Network for Multiplexing and Data Communications Standards Committee, *SAE J1850 Standard*, 1994.

[San03] A. Sangiovanni-Vincentelli, "Electronic-System Design in the Automobile Industry", IEEE Micro, 23(3), pages 8-18, 2003.

[Sch94] M. Schwehm, T. Walter, "Mapping and Scheduling by Genetic Algorithms“, Conference on Algorithms and Hardware for Parallel Processing, pages 832-841, 1994.

[Sha90] L. Sha, R. Rajkumar, J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real Time Synchronization", IEEE Transactions on Computers, 39(9), pages 1175 -1185, 1990.

[Sta94] J. A. Stankovic, M. Spuri, M. di Natale, and G. C. Butazzo, "Implications of Classical Scheduling Results for Real-Time Systems", Technical Report UM-CS-94-089, Computer Science Department, University of Massachusetts, 1994.

[Str89] J. K. Strosnider, T. E. Marchok, "Responsive, Deterministic IEEE 802.5 Token Ring Scheduling", *Journal of Real-Time Systems*, 1(2), pages 133–158, 1989.

[Tab00] B. Tabbara, A. Tabbara, A. Sangiovanni-Vincentelli, "Function/Architecture Optimization and Co-Design of Embedded Systems", Kluwer Academic Publishers, 2000.

[Thi04] L. Thiele, R. Wilhelm, "Design for Timing Predictability", Real-Time Systems Journal, 28(2/3), pages 157-177, 2004.

[Tin92] K. Tindell, A. Burns, A.J. Wellings, "Allocating Hard Real-Time Tasks (An NP-Hard Problem Made Easy)", Journal of Real-Time Systems, 4(2), pages 145-165, 1992.

[Tin94a] K. Tindell, J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems", Microprocessing & Microprogramming, Vol. 50, No. 2-3, 1994.

[Tin94b] K. Tindell, H. Hansson, A. J. Wellings, "Analysing Real-Time Communications: Controller Area Network (CAN)", Proceedings of Real-Time Systems Symposium, 1994.

[Tin94c] K. Tindell, "Adding Time-Offsets to Schedulability Analysis", Department of Computer Science, University of York, Report Number YCS-94-221, 1994.

[Tin95] K. Tindell, A. Burns, A. Wellings, "Calculating CAN Message Response Times", Control Engineering Practice, 3(8), pages 1163-1169, 1995.

[Tov99] E. Tovar, F. Vasques, "Analysis of the Worst-Case Real Token Rotation Time in PROFIBUS Networks", Proceedings of the Fieldbus Conference, pages 359-366, 1999.

[TTP01C] TTP/C Specification, WebSite of Time-Triggered Technology, http://www.tttech.com/, 2007.

[Tur99] J. Turley, "Embedded Processors by the Numbers", Embedded Systems Programming, 1999.

[Ull75] D. Ullman, "NP-Complete Scheduling Problems", Journal of Computer Systems Science, 10(3), pages 384-393, 1975.

[Wan05] E. Wandeler, L. Thiele, "Real-Time Interfaces for Interface-Based Design of Real-Time Systems with Fixed Priority Scheduling", ACM Conference on Embedded Software (EMSOFT), pages 80-89, 2005.

[Wor03] WorldFIP: Digital data communications for measurement and control - Fieldbus standard for use in industrial control systems. parts 1 to 6, IEC Standard 61158, 2003.

[Wol94] W. Wolf, “Hardware-Software Co-Design of Embedded Systems”, *Proceedings of the IEEE*, V82, N7, 1994.

[Wol97] W. Wolf, “An Architectural Co-Synthesis Algorithm for Distributed Embedded Computing Systems”, IEEE Transactions on VLSI Systems, pages 218-229, 1997.

[Wol03] W. Wolf, “A Decade of Hardware/Software Codesign”, IEEE Computer, 36(4), pages 38-43, 2003.

[Wol06] W. Wolf, “High Performance Embedded Computing: Architectures, Applications and Methodologies“, Morgan Kaufman, 2006.

[Xu93] J. Xu, D.L. Parnas, “On satisfying timing constraints in hard-real-time systems”, IEEE Transactions on Software Engineering, 19(1), 1993.

[Yen97] T. Y. Yen, W. Wolf, “Hardware-Software Co-Synthesis of Distributed Embedded Systems”, Kluwer Academic Publishers, 1997.

of Reasoning about Plans, 1992, ISBN 91-7870-979-2.

No 292 **Mats Wirén:** Studies in Incremental Natural Language Analysis, 1992, ISBN 91-7871-027-8.

No 297 **Mariam Kamkar:** Interprocedural Dynamic Slicing with Applications to Debugging and Testing, 1993, ISBN 91-7871-065-0.

No 302 **Tingting Zhang:** A Study in Diagnosis Using Classification and Defaults, 1993, ISBN 91-7871-078-2.

No 312 **Arne Jönsson:** Dialogue Management for Natural Language Interfaces - An Empirical Approach, 1993, ISBN 91-7871-110-X.

No 338 **Simin Nadjm-Tehrani**: Reactive Systems in Physical Environments: Compositional Modelling and Framework for Verification, 1994, ISBN 91-7871-237-8.

No 371 **Bengt Savén:** Business Models for Decision Support and Learning. A Study of Discrete-Event Manufacturing Simulation at Asea/ABB 1968-1993, 1995, ISBN 91-7871-494-X.

No 375 **Ulf Söderman:** Conceptual Modelling of Mode Switching Physical Systems, 1995, ISBN 91-7871-516-4.

No 383 **Andreas Kågedal:** Exploiting Groundness in Logic Programs, 1995, ISBN 91-7871-538-5.

No 396 **George Fodor:** Ontological Control, Description, Identification and Recovery from Problematic Control Situations, 1995, ISBN 91-7871-603-9.

No 413 **Mikael Pettersson:** Compiling Natural Semantics, 1995, ISBN 91-7871-641-1.

No 414 **Xinli Gu:** RT Level Testability Improvement by Testability Analysis and Transformations, 1996, ISBN 91-7871-654-3.

No 416 **Hua Shu:** Distributed Default Reasoning, 1996, ISBN 91-7871-665-9.

No 429 **Jaime Villegas:** Simulation Supported Industrial Training from an Organisational Learning Perspective - Development and Evaluation of the SSIT Method, 1996, ISBN 91-7871-700-0.

No 431 **Peter Jonsson:** Studies in Action Planning: Algorithms and Complexity, 1996, ISBN 91-7871-704-3.

No 437 **Johan Boye:** Directional Types in Logic Programming, 1996, ISBN 91-7871-725-6.

No 439 **Cecilia Sjöberg:** Activities, Voices and Arenas: Participatory Design in Practice, 1996, ISBN 91-7871-728-0.

No 448 **Patrick Lambrix:** Part-Whole Reasoning in Description Logics, 1996, ISBN 91-7871-820-1.

No 452 **Kjell Orsborn:** On Extensible and Object-Relational Database Technology for Finite Element Analysis Applications, 1996, ISBN 91-7871-827-9.

No 459 **Olof Johansson:** Development Environments for Complex Product Models, 1996, ISBN 91-7871-855-4.

No 461 **Lena Strömbäck:** User-Defined Constructions in Unification-Based Formalisms,1997, ISBN 91-7871-857-0.

No 462 **Lars Degerstedt:** Tabulation-based Logic Programming: A Multi-Level View of Query Answering, 1996, ISBN 91-7871-858-9.

No 475 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av hur ekonomiska styrsystem utformas och används efter företagsförvärv, 1997, ISBN 91-7871-914-3.

No 480 **Mikael Lindvall:** An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Software Evolution, 1997, ISBN 91-7871-927-5.

No 485 **Göran Forslund**: Opinion-Based Systems: The Cooperative Perspective on Knowledge-Based Decision Support, 1997, ISBN 91-7871-938-0.

No 494 **Martin Sköld**: Active Database Management Systems for Monitoring and Control, 1997, ISBN 91-7219-002-7.

No 495 **Hans Olsén**: Automatic Verification of Petri Nets in a CLP framework, 1997, ISBN 91-7219-011-6.

No 498 **Thomas Drakengren:** Algorithms and Complexity for Temporal and Spatial Formalisms, 1997, ISBN 91-7219-019-1.

No 502 **Jakob Axelsson:** Analysis and Synthesis of Heterogeneous Real-Time Systems, 1997, ISBN 91-7219-035-3.

No 503 **Johan Ringström:** Compiler Generation for Data-Parallel Programming Langugaes from Two-Level Semantics Specifications, 1997, ISBN 91-7219-045-0.

No 512 **Anna Moberg:** Närhet och distans - Studier av kommunikationsmmönster i satellitkontor och flexibla kontor, 1997, ISBN 91-7219-119-8.

No 520 **Mikael Ronström:** Design and Modelling of a Parallel Data Server for Telecom Applications, 1998, ISBN 91-7219-169-4.

No 522 **Niclas Ohlsson:** Towards Effective Fault Prevention - An Empirical Study in Software Engineering, 1998, ISBN 91-7219-176-7.

No 526 **Joachim Karlsson:** A Systematic Approach for Prioritizing Software Requirements, 1998, ISBN 91-7219-184-8.

No 530 **Henrik Nilsson:** Declarative Debugging for Lazy Functional Languages, 1998, ISBN 91-7219-197-x.

No 555 **Jonas Hallberg:** Timing Issues in High-Level Synthesis,1998, ISBN 91-7219-369-7.

No 561 **Ling Lin:** Management of 1-D Sequence Data - From Discrete to Continuous, 1999, ISBN 91-7219-402-2.

No 563 **Eva L Ragnemalm:** Student Modelling based on Collaborative Dialogue with a Learning Companion, 1999, ISBN 91-7219-412-X.

No 567 **Jörgen Lindström:** Does Distance matter? On geographical dispersion in organisations, 1999, ISBN 91-7219-439-1.

No 582 **Vanja Josifovski:** Design, Implementation and

Evaluation of a Distributed Mediator System for Data Integration, 1999, ISBN 91-7219-482-0.

No 589 **Rita Kovordányi**: Modeling and Simulating Inhibitory Mechanisms in Mental Image Reinterpretation - Towards Cooperative Human-Computer Creativity, 1999, ISBN 91-7219-506-1.

No 592 **Mikael Ericsson:** Supporting the Use of Design Knowledge - An Assessment of Commenting Agents, 1999, ISBN 91-7219-532-0.

No 593 **Lars Karlsson:** Actions, Interactions and Narratives, 1999, ISBN 91-7219-534-7.

No 594 **C. G. Mikael Johansson:** Social and Organizational Aspects of Requirements Engineering Methods - A practice-oriented approach, 1999, ISBN 91-7219-541-X.

No 595 **Jörgen Hansson:** Value-Driven Multi-Class Overload Management in Real-Time Database Systems, 1999, ISBN 91-7219-542-8.

No 596 **Niklas Hallberg:** Incorporating User Values in the Design of Information Systems and Services in the Public Sector: A Methods Approach, 1999, ISBN 91-7219-543-6.

No 597 **Vivian Vimarlund:** An Economic Perspective on the Analysis of Impacts of Information Technology: From Case Studies in Health-Care towards General Models and Theories, 1999, ISBN 91-7219-544-4.

No 598 **Johan Jenvald:** Methods and Tools in Computer-Supported Taskforce Training, 1999, ISBN 91-7219-547-9.

No 607 **Magnus Merkel:** Understanding and enhancing translation by parallel text processing, 1999, ISBN 91-7219-614-9.

No 611 **Silvia Coradeschi:** Anchoring symbols to sensory data, 1999, ISBN 91-7219-623-8.

No 613 **Man Lin:** Analysis and Synthesis of Reactive Systems: A Generic Layered Architecture Perspective, 1999, ISBN 91-7219-630-0.

No 618 **Jimmy Tjäder:** Systemimplementering i praktiken - En studie av logiker i fyra projekt, 1999, ISBN 91-7219-657-2.

No 627 **Vadim Engelson:** Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing, 2000, ISBN 91-7219-709-9.

No 637 **Esa Falkenroth:** Database Technology for Control and Simulation, 2000, ISBN 91-7219-766-8.

No 639 **Per-Arne Persson:** Bringing Power and Knowledge Together: Information Systems Design for Autonomy and Control in Command Work, 2000, ISBN 91-7219-796-X.

No 660 **Erik Larsson:** An Integrated System-Level Design for Testability Methodology, 2000, ISBN 91-7219-890-7.

No 688 **Marcus Bjäreland:** Model-based Execution Monitoring, 2001, ISBN 91-7373-016-5.

No 689 **Joakim Gustafsson:** Extending Temporal Action Logic, 2001, ISBN 91-7373-017-3.

No 720 **Carl-Johan Petri:** Organizational Information Provision - Managing Mandatory and Discretionary Use of Information Technology, 2001, ISBN-91-7373-126-9.

No 724 **Paul Scerri:** Designing Agents for Systems with Adjustable Autonomy, 2001, ISBN 91 7373 207 9.

No 725 **Tim Heyer**: Semantic Inspection of Software Artifacts: From Theory to Practice, 2001, ISBN 91 7373 208 7.

No 726 **Pär Carlshamre:** A Usability Perspective on Requirements Engineering - From Methodology to Product Development, 2001, ISBN 91 7373 212 5.

No 732 **Juha Takkinen:** From Information Management to Task Management in Electronic Mail, 2002, ISBN 91 7373 258 3.

No 745 **Johan Åberg:** Live Help Systems: An Approach to Intelligent Help for Web Information Systems, 2002, ISBN 91-7373-311-3.

No 746 **Rego Granlund:** Monitoring Distributed Teamwork Training, 2002, ISBN 91-7373-312-1.

No 757 **Henrik André-Jönsson:** Indexing Strategies for Time Series Data, 2002, ISBN 917373-346-6.

No 747 **Anneli Hagdahl:** Development of IT-suppor-ted Inter-organisational Collaboration - A Case Study in the Swedish Public Sector, 2002, ISBN 91-7373-314-8.

No 749 **Sofie Pilemalm:** Information Technology for Non-Profit Organisations - Extended Participatory Design of an Information System for Trade Union Shop Stewards, 2002, ISBN 91-7373-
318-0.

No 765 **Stefan Holmlid:** Adapting users: Towards a theory of use quality, 2002, ISBN 91-7373-397-0.

No 771 **Magnus Morin:** Multimedia Representations of Distributed Tactical Operations, 2002, ISBN 91-7373-421-7.

No 772 **Pawel Pietrzak:** A Type-Based Framework for Locating Errors in Constraint Logic Programs, 2002, ISBN 91-7373-422-5.

No 758 **Erik Berglund:** Library Communication Among Programmers Worldwide, 2002,
ISBN 91-7373-349-0.

No 774 **Choong-ho Yi:** Modelling Object-Oriented
Dynamic Systems Using a Logic-Based Framework, 2002, ISBN 91-7373-424-1.

No 779 **Mathias Broxvall:** A Study in the
Computational Complexity of Temporal
Reasoning, 2002, ISBN 91-7373-440-3.

No 793 **Asmus Pandikow:** A Generic Principle for
Enabling Interoperability of Structured and
Object-Oriented Analysis and Design Tools, 2002, ISBN 91-7373-479-9.

No 785 **Lars Hult:** Publika Informationstjänster. En studie av den Internetbaserade encyklopedins bruksegenskaper, 2003, ISBN 91-7373-461-6.

No 800 **Lars Taxén:** A Framework for the Coordination of Complex Systems´ Development, 2003, ISBN 91-7373-604-X

No 808 **Klas Gäre:** Tre perspektiv på förväntningar och förändringar i samband med införande av informationsystem, 2003, ISBN 91-7373-618-X.

No 821 **Mikael Kindborg:** Concurrent Comics - programming of social agents by children, 2003, ISBN 91-7373-651-1.

No 823 **Christina Ölvingson:** On Development of Information Systems with GIS Functionality in Public Health Informatics: A Requirements Engineering Approach, 2003, ISBN 91-7373-656-2.

No 828 **Tobias Ritzau:** Memory Efficient Hard Real-Time Garbage Collection, 2003, ISBN 91-7373-666-X.

No 833 **Paul Pop:** Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems, 2003, ISBN 91-7373-683-X.

No 852 **Johan Moe:** Observing the Dynamic Behaviour of Large Distributed Systems to Improve Development and Testing - An Emperical Study in Software Engineering, 2003, ISBN 91-7373-779-8.

No 867 **Erik Herzog:** An Approach to Systems Engineering Tool Data Representation and Exchange, 2004, ISBN 91-7373-929-4.

No 872 **Aseel Berglund:** Augmenting the Remote Control: Studies in Complex Information Navigation for Digital TV, 2004, ISBN 91-7373-940-5.

No 869 **Jo Skåmedal:** Telecommuting's Implications on Travel and Travel Patterns, 2004, ISBN 91-7373-935-9.

No 870 **Linda Askenäs:** The Roles of IT - Studies of Organising when Implementing and Using Enterprise Systems, 2004, ISBN 91-7373-936-7.

No 874 **Annika Flycht-Eriksson:** Design and Use of Ontologies in Information-Providing Dialogue Systems, 2004, ISBN 91-7373-947-2.

No 873 **Peter Bunus:** Debugging Techniques for Equation-Based Languages, 2004, ISBN 91-7373-941-3.

No 876 **Jonas Mellin:** Resource-Predictable and Efficient Monitoring of Events, 2004, ISBN 91-7373-956-1.

No 883 **Magnus Bång:** Computing at the Speed of Paper: Ubiquitous Computing Environments for Healthcare Professionals, 2004, ISBN 91-7373-971-5

No 882 **Robert Eklund:** Disfluency in Swedish human-human and human-machine travel booking dialogues, 2004. ISBN 91-7373-966-9.

No 887 **Anders Lindström:** English and other Foreign Linquistic Elements in Spoken Swedish. Studies of Productive Processes and their Modelling using Finite-State Tools, 2004, ISBN 91-7373-981-2.

No 889 **Zhiping Wang:** Capacity-Constrained Production-inventory systems - Modellling and Analysis in both a traditional and an e-business context, 2004, ISBN 91-85295-08-6.

No 893 **Pernilla Qvarfordt:** Eyes on Multimodal Interaction, 2004, ISBN 91-85295-30-2.

No 910 **Magnus Kald:** In the Borderland between Strategy and Management Control - Theoretical Framework and Empirical Evidence, 2004, ISBN 91-85295-82-5.

No 918 **Jonas Lundberg:** Shaping Electronic News: Genre Perspectives on Interaction Design, 2004, ISBN 91-85297-14-3.

No 900 **Mattias Arvola:** Shades of use: The dynamics of interaction design for sociable use, 2004, ISBN 91-85295-42-6.

No 920 **Luis Alejandro Cortés:** Verification and Scheduling Techniques for Real-Time Embedded Systems, 2004, ISBN 91-85297-21-6.

No 929 **Diana Szentivanyi:** Performance Studies of Fault-Tolerant Middleware, 2005, ISBN 91-85297-58-5.

No 933 **Mikael Cäker:** Management Accounting as Constructing and Opposing Customer Focus: Three Case Studies on Management Accounting and Customer Relations, 2005, ISBN 91-85297-64-X.

No 937 **Jonas Kvarnström:** TALplanner and Other Extensions to Temporal Action Logic, 2005, ISBN 91-85297-75-5.

No 938 **Bourhane Kadmiry:** Fuzzy Gain-Scheduled Visual Servoing for Unmanned Helicopter, 2005, ISBN 91-85297-76-3.

No 945 **Gert Jervan:** Hybrid Built-In Self-Test and Test Generation Techniques for Digital Systems, 2005, ISBN: 91-85297-97-6.

No 946 **Anders Arpteg:** Intelligent Semi-Structured Information Extraction, 2005, ISBN 91-85297-98-4.

No 947 **Ola Angelsmark:** Constructing Algorithms for Constraint Satisfaction and Related Problems - Methods and Applications, 2005, ISBN 91-85297-99-2.

No 963 **Calin Curescu:** Utility-based Optimisation of Resource Allocation for Wireless Networks, 2005. ISBN 91-85457-07-8.

No 972 **Björn Johansson:** Joint Control in Dynamic Situations, 2005, ISBN 91-85457-31-0.

No 974 **Dan Lawesson:** An Approach to Diagnosability Analysis for Interacting Finite State Systems, 2005, ISBN 91-85457-39-6.

No 979 **Claudiu Duma:** Security and Trust Mechanisms for Groups in Distributed Services, 2005, ISBN 91-85457-54-X.

No 983 **Sorin Manolache:** Analysis and Optimisation of Real-Time Systems with Stochastic Behaviour, 2005, ISBN 91-85457-60-4.

No 986 **Yuxiao Zhao:** Standards-Based Application Integration for Business-to-Business Communications, 2005, ISBN 91-85457-66-3.

No 1004 **Patrik Haslum:** Admissible Heuristics for Automated Planning, 2006, ISBN 91-85497-28-2.

No 1005 **Aleksandra Tešanovic:** Developing Reusable and Reconfigurable Real-Time Software using Aspects and Components, 2006, ISBN 91-85497-29-0.

No 1008 **David Dinka:** Role, Identity and Work: Extending the design and development agenda, 2006, ISBN 91-85497-42-8.

No 1009 **Iakov Nakhimovski:** Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis, 2006, ISBN 91-85497-43-X.

No 1013 **Wilhelm Dahllöf:** Exact Algorithms for Exact Satisfiability Problems, 2006, ISBN 91-85523-97-6.

No 1016 **Levon Saldamli:** PDEModelica - A High-Level Language for Modeling with Partial Differential Equations, 2006, ISBN 91-85523-84-4.

No 1017 **Daniel Karlsson:** Verification of Component-based Embedded System Designs, 2006, ISBN 91-85523-79-8.