Dipl.-Ing.(FH) Florian Pölzlbauer

Communication-Centric Task Allocation for Designing and Maintaining Networked Embedded Real-Time Systems

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der technischen Wissenschaften
(Dr. techn.)

eingereicht an der

Technischen Universität Graz

Betreuer:

Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Eugen Brenner

Institut für Technische Informatik

Graz, im August 2014

Kurzfassung (German)

Die vorliegende Arbeit widmet sich Methoden für Entwurf und Synthese von verteilten, echtzeitfähigen, eingebetteten Systemen. Der Trend weg von monolithischer Software hin zu modularen Software-Komponenten erhöht zwar Flexibilität und Wiederverwendbarkeit, bedarf jedoch auch zusätzlicher Design-Entscheidungen: Wie sollen die Software-Komponenten auf Prozessoren verteilt werden; finden einer effizienten Netzwerk-Konfiguration; finden eines geeigneten Schedules für Software und Kommunikation.

In der Literatur werden diese Aufgaben meist nur für das Design-Szenario adressiert, bei dem ein System vollkommen neu entwickelt wird. Vorliegende Arbeit fokussiert jedoch auf ein evolutionäres Design-Szenario, d.h. ein System wird durch Weiterentwicklung eines Vorgänger-Systems entworfen, wodurch sich diverse Design-Constraints ergeben. Zusätzlich muss beim Erstentwurf beachtet werden, dass das System in Zukunft verändert werden kann, und es bei einem System-Upgrade rückwärts-kompatibel bleibt. All diese Anforderungen erschweren den Entwurf, spiegeln jedoch industrielle Entwicklungsprozesse wider.

Der wissenschaftliche Neuwert dieser Arbeit ist: Betrachtung von Constraints für die Software-Verteilung und Netzwerk-Konfiguration; Minimierung der Netzwerk-Auslastung durch optimiertes Frame Packing; Entwurf von erweiterbaren Netzwerk-Konfigurationen; Upgrade einer Netzwerk-Konfiguration unter Beibehaltung der Rückwärts-Kompatibilität. Die Methoden wurden in einem Prototypen implementiert und anhand realistischer Datensätze evaluiert.

Schlagwörter: Softwareverteilung, Netzwerk-Konfiguration, Frame Packing, Scheduling, Design-Constraints, Erweiterbarkeit, Rückwärts-Kompatibilität, Echtzeitsysteme, Optimierung, Entwurfsraum Exploration

Abstract

This work contributes to methodologies for designing networked embedded real-time systems. Recent trends in embedded systems show a shift from large monolithic software-systems towards smaller reusable software-components. This offers increased flexibility, but also leads to additional design-decisions which need to be taken. The design-decisions include: which application software-component to allocate onto which processor; how to configure data-transmission between processors via networks and gateways; how to schedule task-execution and data-transmission.

While most related work in the literature addresses the scenario that the system is designed "from scratch", this work instead focuses on designing the system in an *evolutionary manner*. This means that legacy decisions have to be taken into account, which poses significant constraints on the design space. In addition, the system shall be designed in a way that it is extensible towards future modifications. If the system is upgraded, the new configuration shall be backward-compatible to the old configuration. All these requirements make the engineering task hard to solve, but at the same time ensure that the nature of industrial embedded systems is adequately captured.

The main contributions of this work are constraint-aware task allocation, network configuration (especially frame packing) which minimizes bandwidth-utilization, extensible network configuration, and backward-compatible network configuration. All proposed methodologies have been implemented as proof-of-concept algorithms, and successfully evaluated on realistic data-sets.

Keywords: task allocation, network configuration, frame packing, scheduling, design-constraints, extensibility, backward-compatibility, real-time systems, design space exploration, optimization

Acknowledgement

The contributions of this thesis were elaborated in the course of the projects BNO-II and HYBCONS that were financially supported by the *COMET – Competence Centers for Excellent Technologies program* of the *Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT)*, the *Federal Ministry of Science, Research and Economy (BMWFW)*, the *Austrian Research Promotion Agency (FFG)*, the *province of Styria* and the *Styrian Business Promotion Agency (SFG)*.

Foremost, I want to thank Prof. Eugen Brenner for supervising this thesis, for his guidance, for his valuable discussions and feedback concerning technical aspects, and for his patience.

I want to thank VIRTUAL VEHICLE for enabling me to spend 3 months abroad, while working in the HYBCONS project. Thanks to Prof. Alan Burns, I could spend this time at the University of York in the Real-Time Systems research group. During that time I had valuable discussions with Prof. Alan Burns, Dr. Rob Davis, Dr. Leandro Soares Indrusiak and especially with Dr. Iain Bate. These discussions helped me to narrow and focus my research questions. Thanks to the group's openness, just a few days after arriving there, it felt like being a permanent member of the research group.

I also want to thank my work-colleges at VIRTUAL VEHICLE for valuable discussions and feedback. Thank goes to Dr. Daniel Watzenig, Dr. Hannes Stippel, Dr. Martin Benedikt, and especially to Dr. Allan Tengg and Mario Driussi.

Additional thanks goes to: Dr. Marek Jersak for a presentation on scheduling analysis back in 2007, which initiated my interest in real-time systems in the first place; Prof. Christian Magele for his valuable insights concerning meta-heuristics and optimization; Dr. Raimund Kirner for the tool `calc_wcet_167`; Dr. Mike Holenderski for the tools `grasp` and `real-time view`; Dr. Paul Emberson for the tool `GenTAP` and for research questions I could derive from reading his PhD thesis; and several engineers at AVL, Delphi, Audi, BMW and VW for discussions which increased my insight into industrial requirements for designing embedded systems (especially automotive systems).

Finally, I want to thank my family and friends for their support and patience.

Graz, Austria
August 2014 Florian Pölzlbauer

Contents

Glossary

Within this chapter the *abbreviations, symbols, terms* and *definitions* which are used throughout this work are presented. They are used frequently throughout the entire thesis.

Abbreviations

AUTOSAR *automotive open system architecture* – Standardization of the software-platform for automotive electronic systems. It is closely related to the OSEK/VDX standard.

BPP *bin packing problem* – NP-hard optimization problem: How shall a set of items be packed into a set of bins in an optimal way.

CAN *controller area network* – Protocol for serial communication network. Network arbitration is based on frame priority.

DMPO *deadline monotonic priority ordering* – Priority ordering approach where priorities are assigned inverse-proportional to the deadlines. Tasks with short deadline get high priority, tasks with long deadline get low priority.

DSE *design space exploration* – Methodology for systematically exploring the design space.

ECU *electronic control unit* – Term which is mainly used in the automotive domain. It contains a CPU, memory, IOs and bus-interfaces. It can be seen as a synonym for *processor*.

EDF *earliest deadline first* – Scheduling policy where the priority of a task is defined at runtime. The task which is closest to its absolute deadline gets the highest priority.

FPP *frame packing problem* – NP-hard optimization problem: How shall a set of messages be packed into a set of frames in an optimal way.

FlexRay *FlexRay* – Protocol for serial communication network. Network arbitration is based on a hybrid approach: both TDMA and priority-based.

GA *genetic algorithm* – Meta-heuristic which mimics the process of natural selection. It is used for solving optimization problems.

GenFPP *generate frame packing problem* – Algorithm/tool for generating pseudo-random synthetic frame packing problem instances.

ILP *integer linear programming* – Technique for the optimization of a linear objective function, subject to linear equality and linear inequality constraints.

LIN *local interconnect network* – Protocol for low-speed, low-cost serial communication network. It follows a master/slave approach (LIN-master sends a data-request and LIN-slave returns the corresponding data) and uses a TDMA arbitration schema.

MILP *mixed integer linear programming* – A derivate of *integer linear programming* (ILP) where decision variables can either be *integer* or *boolean*.

MOST *media oriented systems transport* – Protocol for high-speed communication network. In automotive systems, it is mainly used for audio/video data.

OEM *original equipment manufacturer* – In the automotive domain, OEM is a synonym for car-manufacturer, such as Audi, BMW, Daimler, VW, etc. The OEMs work closely together with their suppliers.

OSEK/VDX *Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug (English: Open Systems and the Corresponding Interfaces for Automotive Electronics) / Vehicle Distributed eXecutive* – Standardization of the software-platform for automotive electronic systems. It contains 3 main parts: operating system, communication, network management.

RTA *response time analysis* – Schedulability-test, which calculates the WCRT of each task, and checks if the WCRT does not exceed the deadline.

RTOS *real-time operating system* – Operating system that uses a real-time scheduler. This allows to upper-bound the timing behaviour of the system, and thus to give timing guarantees.

SA *simulated annealing* – Meta-heuristic which mimics the process of cooling molten metal. It is used for solving optimization problems.

TAP *task allocation problem* – NP-hard optimization problem: How shall a set of tasks be allocated on a set of processors in an optimal way.

TDMA *time division multiple access* – Scheduling schema which divides the available time into a set of time-slots.

Tier *supplier* – In the automotive domain, Tier is a synonym for supplier. The suppliers work closely together with the OEMs. From the OEMs view, there are several levels of Tiers. First-level Tiers supply components such as ECUs, engines, transmissions, etc. Second-level Tiers supply components such as micro-controllers.

TT-CAN *time-triggered CAN* – Protocol for serial communication network. It uses TDMA on top of CAN.

WCET *worst-case execution time* – The max. time it takes to execute a task. This time does not include any interference or blocking.

WCRT *worst-case response time* – The max. time from a task being activated until it finishes executing. This time includes all interferences and blocking.

WCTT *worst-case transmission time* – The max. time it takes to transmit a frame via a bus-system. This time does not include the time it takes to arbitrate the bus-system.

Symbols

Within the context of *real-time analysis* the following symbols are used:

Symbol	Description
B_i	blocking of task i due to lower priority tasks
C_i	WCET of task i
D_i	deadline of task i
I_i	interference of task i due to higher priority tasks
J_i	release jitter of task i
O_i	offset of task i
P_i	priority of task i
R_i	WCRT of task i
T_i	period of task i
U_i	utilization of task i
Z_i	worst-case computation cycles of task i

At several points within this work *set theory* is utilized. Thereby, the following elements, sets and set-operators are used:

Symbol	Description
τ	task
m	message
f	frame
β	bus
ρ	processor
$\mathcal{T}$	set of tasks
$\mathcal{M}$	set of messages
$\mathcal{F}$	set of frames
$\mathcal{B}$	set of bus-systems
$\mathcal{P}$	set of processors
$\cup$	union of sets
$\cap$	intersection of sets
$\setminus$	complement of sets

Terms

In the literature different terms are used in order to describe equivalent elements and methodologies. In order to avoid this confusion, the following terms are used:

Term	**Description**	**Alias in Literature**
task	the smallest unit handled by a scheduler	process, module, software, thread, component, job, operation
job	an invocation of a task	–
task cluster	a group of tasks	task group
message	data exchange between tasks	signal, data link, communication link, data element
software	tasks & messages	task graph
frame	the smallest unit transferred via a bus	packet, message, telegram
processor	electronic device consisting of μC and peripherals	processing node, node, ECU, cluster
bus	communication medium connecting processors	network, communication link
hardware	processors & bus-systems	–
task allocation	local assignment of tasks to processors	task assignment, task mapping, scheduling
task clustering	grouping of tasks	task grouping
task cluster allocation	local assignment of task clusters to processors	task merging
scheduling	temporal assignment of tasks & frames (time-triggered: time slot planning; priority-based: priority assignment)	–
frame packing	assignment of messages to frames	message packing, signal packing, data element allocation
configuration	allocation & frame packing & scheduling	–

Definitions

Based on the proposed terms, the following definitions of *feasibility* can be stated. These definitions will be used throughout this work.

$$\text{Configuration is feasible iff} \left\{ \begin{array}{l} \text{task allocation,} \\ \text{message routing,} \\ \text{frame packing,} \\ \text{resource utilization, and} \\ \text{scheduling} \end{array} \right. \quad \text{are feasible.}$$

$$\text{Task allocation is feasible iff} \left\{ \begin{array}{l} \text{no allocation constraints are violated, and} \\ \text{each task can be allocated to a processor.} \end{array} \right.$$

$$\text{Message routing is feasible iff} \left\{ \begin{array}{l} \text{no routing constraints are violated, and} \\ \text{each message can be routed to its destination.} \end{array} \right.$$

$$\text{Frame packing is feasible iff} \left\{ \begin{array}{l} \text{no packing constraints are violated, and} \\ \text{each message can be packed into a frame.} \end{array} \right.$$

$$\text{Resource utilization is feasible iff } \forall \text{ resources} : u \leq 100\%$$

$$\text{Scheduling is feasible iff} \left\{ \begin{array}{l} \text{time slot plan,} \\ \text{priority assignment, and} \\ \text{real-time behaviour} \end{array} \right. \quad \text{are feasible.}$$

$$\text{Time slot plan is feasible iff} \left\{ \begin{array}{l} \text{no time slots are overlapping, and} \\ \forall \text{ time-triggered objects} : C \leq \text{time slot length} \end{array} \right.$$

$$\text{Priority assignment is feasible iff priorities are unique.}$$

$$\text{Real-time behaviour is feasible iff} \left\{ \begin{array}{l} \forall \text{ schedulable objects} : R \leq D \\ \forall \text{ end-to-end-delays} : R \leq D \end{array} \right.$$

Chapter 1

Introduction

In recent years, a common trend in electronic devices can be observed: key product features are defined by embedded software (e.g. software-defined radio instead of traditional radio-transceivers). This trend is also true for mechatronic systems, such as cars. Estimates state that up to 80% of customer-visible innovation in a modern car originates from embedded software. Often, these are time- and safety-critical features (e.g. steer-by-wire). In addition, the use of embedded software enables a wider range of product-variants, as well as increased flexibility. During the last several years, the most significant **trends** [13, 50] in embedded computing (especially in the automotive domain) are:

- from *federated* to *integrated* architecture:
  In a *federated architecture*, each functionality is implemented on a dedicated hardware unit. In several cases, the function is implemented in a *bare bone* manner, i.e. no operating system is underlying. In an *integrated architecture*, several functionality is implemented on one hardware unit. In most cases, an operating system is used.

- from *special-purpose* to *general-purpose* hardware:
  In the past, functionality was often implemented by special-purpose hardware (e.g. FPGA). Current trends are (more and more) towards using general-purpose processing units, and implement functionality in software.

- *component-oriented software* and *middleware* [5]:
  In order to improve reuseability of functionality, software is developed according to *component-oriented* approaches, where the components have standardized interfaces. In addition, hardware-specific functionality is encapsulated by a *middleware*, thus making the application-software quite hardware-independent.

- *increased complexity* and *more product-variants*:
  Modern products (such as cars) contain ever more functionality. In addition, the manufacturers provide ever more product-variants.

These trends – in combination with the need of ever shorter *time-to-market* – lead to **new challenges** for embedded-systems engineers:

- Hardware-independent software-components, with standardized interfaces, allow to move the software to different processors. The number of possibilities grows exponentially, therefore it would be very helpful to provide the engineers with a methodology which assists the engineer with respect to this decision-making.

- Moving software-components between processors has a significant impact on the communication-demand between the processors. Therefore, a methodology for *allocation software-components to processors* needs to go hand-in-hand with a methodology for *configuring the bus-communication* between processors.

- Many industrial systems (such as cars) are developed in an *evolutionary manner*, i.e. a previous product-version is taken as a baseline, and this version is extended by new features and/or some changes are applied, in order to satisfy the "new system requirements". Therefore, a set of constraints to the design-space may exist. These constraints must be considered and handled appropriately.

The **objective** of this thesis is to address these challenges, and to provide methodologies for solving them. To a large degree, the methodologies developed within this thesis are generally applicable to a wide range of embedded real-time systems. However, the problems are inspired by the *automotive domain* and their standards.

Outline

The thesis is structured as follows. First, some basics are addressed: Chapter 2 gives an overview of related work and state-of-the-art. Chapter 3 states the research questions of the thesis. Then, the core of the thesis is presented:

- Chapter 4 describes the basic search- and optimization-framework, and outlines which aspects are subject to enhancements.

- Chapter 5 develops a methodology which allows to design a realistic and efficient configuration of the cross-processor communication infrastructure. Key aspect is *frame packing* for minimizing the network bandwidth demand.

- Chapter 6 develops a methodology, how a set of *heterogeneous design-constraints* can be handled efficiently. This is key to solve realistic industrial problems which are subject to design- and legacy-constraints.

- Chapter 7 enhances the methodology for communication configuration by 2 aspects: (1) How to design a communication configuration which is capable of *handling future changes*? (2) How can the communication configuration be *extended / upgraded* in a way that it *maintains compatible* to a previous configuration? These aspects are key to evolve a system design.

Finally, chapter 8 concludes the thesis and highlights relevant future research directions.

Chapter 2

Related Work & State of the Art

In nowadays cars, electronics and embedded software are key factors for innovation. Most of these automotive embedded systems have to satisfy real-time behavioural requirements.

> *"A real-time computer system is a computer system in which the correctness of the system behaviour depends not only on the logical results of the computations, but also on the physical instant at which these results are produced."*
>
> – Hermann Kopetz [40]

Designing such embedded real-time systems is a challenging engineering task. It involves a set of challenging design decisions as well as analysis steps.

2.1 Real-Time System Model

Nowadays, the majority of embedded real-time systems are implemented by a set of software-tasks which are executed on a set of processors. Thus, a real-time system can be described by two model-aspects:

- *software* – A set of software-tasks implement the *business logic* of the real-time application. Each task is responsible for a certain aspect of the application. Tasks exchange data via messages. The entire application can be modeled as a *directed graph* where nodes represent tasks and edges represent messages.

- *hardware* – A set of processors execute the software-tasks. The processors are interconnected via bus-systems. The hardware can be modeled as an *undirected graph* where nodes represent processors and edges represent bus-systems.

In order to make it easier to execute a software-task on different processors, hardware-specifics are encapsulated in dedicated software-modules. This concept is often referred to as a *middleware* approach.

2.1.1 Real-Time Task Model

The task model is a representation which models the *temporal aspects* of a software component. It allows to model (and analyse) the execution of a software-task on a processor. Likewise, it can also be used to model the transmission of a frame via a bus-system.

A task can be in one of several states (see figure 2.1). These states are in alignment with the *real-time operating system* (RTOS) scheduler capabilities. Most RTOS support the following states:

- *ready* – task is activated by its triggering event, and waiting to be executed
- *running* – task is currently executed
- *blocked* – task is waiting for a shared resource, which is currently used by another task. This state is also known as *waiting*.
- *inactive* – task has finished executing (or has never been triggered yet). This state is also known as *suspended*.

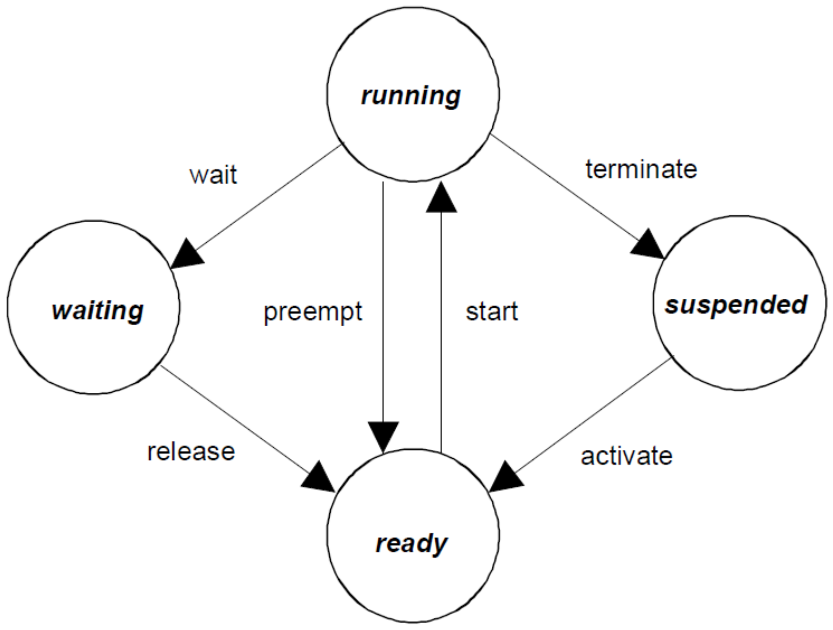


Figure 2.1: State diagram of preemptive real-time task with blocking [52]

Figure 2.2 shows the different timing aspects of the task model. It is plotted using the tool `grasp` [34].

2.2 Real-Time Analysis

Real-time systems are systems, which have to fulfill timing requirements. A response of the system is valid, only if (a) the response is correct, and (b) occurs within a defined time-frame. Thus, if a correct response occurs too early or too late (i.e. does not satisfy

Time	Description	Symbol
arrival time	time at which task arrives (i.e. is triggered by event)	$\downarrow$
release time	time at which task is released (i.e. starts to execute)	r_i
release jitter	jitter of releases	J_i
execution	time during which task executes	■
preemption	time during which task is preempted by higher-priority tasks	□
finishing time	time at which task finishes executing	f_i
deadline	time at which task must finish	$\uparrow$
deadline (rel.)	time within which task must finish (i.e. relative to task's arrival)	D_i
period	time between two consecutive triggering of task	T_i

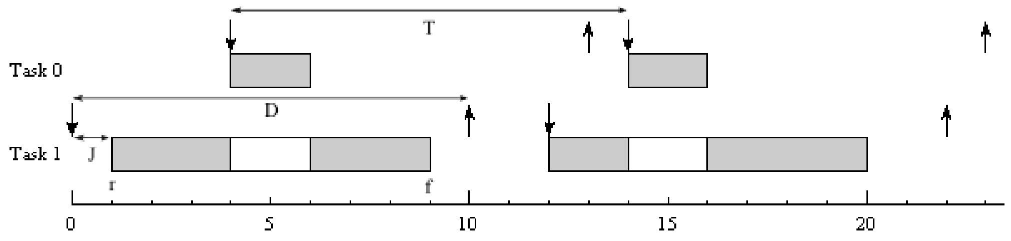


Figure 2.2: Gantt-chart of real-time tasks

the timing requirements), the response is not valid. In order to guarantee that a real-time system meets all its timing requirements, real-time analyses have to be performed. These focus on two aspects:

- *execution time* – For each software-task it needs to be analysed, how long it takes to execute it on a certain processor. By definition, the execution time does not contain any interference by any other task. The max. execution time is referred to as the *worst-case execution time* (WCET).
- *response time* – As several tasks may be executed on the same processor, there may be interference between these tasks (e.g. by preemption). The response time is the time it takes to execute a task, including all interferences. The max. response time is referred to as the *worst-case response time* (WCRT). This WCRT must be smaller or equal the defined deadline.

2.2.1 Execution Time Analysis

For real-time systems, it is important to know how long it takes to execute a software-task on a certain processor. Thereby, the max. time – the so called WCET – is of special interest.

> *"Timing analysis attempts to determine bounds on the execution times of a task when executed on a particular hardware. The time for a particular execution*

depends on the path through the task taken by control and the time spent in the statements or instructions on this path on this hardware. Accordingly, the determination of execution-time bounds has to consider the potential control-flow paths and the execution times for this set of paths."

– Wilhelm et al. [83]

In the literature, several approaches have been proposed to determine the WCET. In [83] a detailed overview is provided. Basically, the approaches can be categorized as follows:

- *measurement-based* – These methods execute the task on the given hardware or a simulator for some set of inputs. They then take the measured times and derive the maximal observed execution times (and the execution time distribution).
  The main issue concerning measurement-based approaches is that the measured execution time is dependent on the used input-data, since different inputs lead to different execution-paths through the software-task.

- *static* – These methods do not rely on executing code on real hardware or on a simulator. They rather take the task code itself, maybe together with some annotations, analyze the set of possible control flow paths through the task, combine control flow with some (abstract) model of the hardware architecture, and obtain upper bounds for this combination.
  The main issue concerning static approaches is to provide an accurate model of the hardware. This is especially challenging for modern hardware.

- *hybrid* – These methods try to combine the benefits of both approaches. Static methods are used for finding adequate control-flows, measurement-based approaches are used to determine the execution-time for that path.

In this thesis, determining the WCET is not the research-focus. Therefore, it is assumed that the WCET is known for each task.

2.2.2 Response Time Analysis / Schedulability-Test

A schedulability-test is a formal analysis which determines if a set of tasks is schedulable according to a scheduling policy. Schedulability-tests can be categorized as follows:

- *necessary* – If the schedulability-test is negative, the task-set is definitely not schedulable (e.g.: if utilization $> 100\%$ then the task-set is not schedulable). However, if the schedulability-test is positive, the task-set may still be un-schedulable.

- *sufficient* – If the schedulability-test is positive, the task-set is definitely schedulable. However, if the schedulability-test is negative, the task-set may still be schedulable [19].

- *exact* – The schedulability-test always gives the correct answer.

In this thesis, exact schedulability-tests are used. One class of exact schedulability-tests is the *response time analysis* (RTA). Its main benefit is that it not only gives a correct true/false result. By calculating the WCRT more insight can be gained, especially when the results are plotted via a gantt-chart which visualizes blocking- and preemption-times (see figure 2.2).

For a set of independent tasks, which are scheduled according to *fix priority preemptive scheduling*, the WCRT of each task is calculated as follows [2]:

$$R_i = J_i + B_i + C_i + \sum_{k \in hp(\tau_i)} \left\lceil \frac{R_i + J_k}{T_k} \right\rceil C_k \tag{2.1}$$

where the longest time a task can be blocked (due to waiting for a shared resource) is:

$$B_i = \max_{k \in lp(\tau_i)} \{C_k\} \tag{2.2}$$

In a similar way, the scheduling of frames on a CAN can be calculated [21]. The RTA is performed for the so called *critical instance*. However, in case the tasks have offset-dependency between them, the above stated equation produces overly-pessimistic results. Therefore, offset-aware RTA can be used, in order to get more precise results [53, 80].

2.3 Real-Time Scheduling

In order to satisfy the real-time behaviour constraints, real-time systems use RTOS and real-time aware communication protocols. At runtime of the system, the software-tasks are scheduled by the RTOS scheduler, and the bus frames are transmitted according to the bus protocol. Therefore, different scheduling policies have been proposed in the literature, and a set of them have been implemented in industrial systems. The most common scheduling policies are:

2.3.1 Time-Triggered

Each task is executed at pre-defined points in time. All information, at which time each task is executed, is stored in a *scheduling table* which needs to be built at design time. There must not be any temporal overlap within the scheduling table entries. This approach is also known as *time division multiple access* (TDMA).

- pros – As every action happens at pre-defined points in time, the system is precisely deterministic.

- cons – Building the scheduling table is challenging. If changes need to be applied to the system configuration, it may be necessary to re-build the entire scheduling table. Thus, this approach is not flexible nor easily extensible. Only few RTOS support this approach.

2.3.2 Event-Driven

A task may be triggered at any point in time. However, most systems use periodic triggering patterns. If several tasks are triggered within the same time-frame, a schema is used to determine the execution order of the tasks. In most cases the schema is *priority-based*. Thus, planing a schedule boils down to assigning priorities to tasks.

- pros – The system can be easily adapted to changing requirements, simply by assigning new priorities. Thus, the approach is suitable to build flexible and extensible systems. Most commercial RTOS use priority-based scheduling.
- cons – As tasks may interfere with each other (e.g. by preempting each other) the system becomes less deterministic. However, the timing behaviour of the system can still be estimated by an upper bound (e.g. by using RTA).

Within this schema, there are different principles to determine the priorities:

- *fix priority* – At design-time, each tasks is assigned an unique priority. This priority is never changed. The only exception is to temporarily raise the priority of a task, in order to avoid deadlocks when using shared resources [46].
- *dynamic priority* – The priority of each task is determined at runtime. It may change at any time. A commonly known priority assignment policy is *earliest deadline first* (EDF).

Remarks

Of course, there is no *single best* scheduling policy. Each approach has its benefits and shortcomings. [45] gives a good comparison of *time-triggered vs. event-driven* scheduling. This work is especially recommended, as it also gives insight into system-performance of a system which is comprised of both policies (e.g. event-driven processor and time-triggered bus-system). In [12] a comparison of *fix priority* and EDF is provided. The comparison findings are summarized in table 2.1.

For this thesis a pragmatic decision is made: RTOS which are used in the automotive domain specify that *fix priority* has to be used. Thus, *fix priority* task scheduling is used within this thesis.

2.4 Priority Assignment

Designing a schedule for a real-time system which uses the *fix priority* scheduling policy, boils down to assigning a priority to each task. The problem is defined as follows:

> Given a set of tasks $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$ which shall be scheduled on a processor. Each task is defined by its WCET, the period at which it is triggered, and its deadline. Find a priority assignment, such that all tasks meet their deadline.

In the literature, a set of strategies have been proposed for solving this problem. Most commonly known approaches are:

Policy	Pros	Cons
RM	• supported by many RTOS • high-priority tasks will always be executed, even if processor is highly utilized • jitter of response time is small for high-priority tasks	• if processor is highly utilized, low priority tasks are likely to miss their deadline • higher number of preemptions and context switches • jitter of response time is large for low-priority tasks
EDF	• all tasks meet their deadlines, even if processor is highly utilized (up to 100%) • fewer number of preemptions and context switches • jitter of response time is evenly distributed	• only supported by few RTOS • higher implementation complexity of scheduler • if processor is overloaded then all tasks will miss their deadlines

Table 2.1: Comparison: RM vs. EDF

- *rate monotonic (RM)* – The priority of the tasks is assigned inverse-proportional to their period. Tasks which have a short period are assigned a high priority, tasks which have a long period are assigned a low priority. This approach is proven to be optimal for a set of independent tasks, where the task's deadline is equal to its period [44].

- *deadline monotonic (DM)* – The priority of the tasks is assigned inverse-proportional to their deadline. Tasks which have a short deadline are assigned a high priority, tasks which have a long deadline are assigned a low priority. This approach is also known as *deadline monotonic priority ordering* (DMPO), and is proven to be optimal for a set of independent tasks, where the task's deadline is less than or equal to its period [4].

- *Audsley's algorithm (OPA)* – Audsley found that the WCRT of a task is influenced by higher-priority tasks, but not by lower-priority tasks. From that he derived an *optimal priority assignment (OPA)* algorithm, which assigns each task the lowest priority at which it is still schedulable [1,3].

2.5 Task Allocation

Within the real-time systems community, the *task allocation problem* (TAP) is one of the main research questions. The TAP is defined as follows:

> Given a software-application consisting of a set of tasks $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$ which communicate via a set of messages $\mathcal{M} = \{m_1, m_2, ..., m_k\}$. Further given an execution-platform consisting of a set of processors $\mathcal{P} = \{\rho_1, \rho_2, ..., \rho_o\}$ which are interconnected via a set of bus-systems $\mathcal{B} = \{\beta_1, \beta_2, ..., \beta_p\}$. Find a feasible allocation of tasks to processors (and messages to bus-systems). The allocation should be optimal, according to defined optimization-criteria.

The TAP is known to be a NP-hard optimization problem. In the literature, a set of approaches to solve the TAP have been proposed. These approaches can be grouped by the solving-principles which are applied.

2.5.1 Graph Theory

Early work [77] address the TAP by using graph theory. The goal is to allocate a task graph onto 2 identical processors. The optimization objective is to maximize the parallel execution time, by minimize the communication between processors. This problem could be solved by using *max-flow/min-cut* algorithms. Communication demand is indicated by edge-weights in the task graph.

Later work [42] extended the problem to > 2 processors, and address it as a *quadratic binary program*. However, both approaches do not consider constrained processing power of the processors nor any real-time behavioural requirements.

In [10, 11] the TAP is solved using *clustering* algorithms. Again, the optimization objective is to minimize the communication between processors, thus reducing the bus utilization.

2.5.2 Heuristics

Several authors have developed heuristics for solving *soft real-time* TAP. Here, the optimization-objective is to minimize response time of tasks and/or to balance processor-load. However it is not guaranteed that task deadlines are met in every case (i.e. hard real-time systems).

In [16] a two-stage approach is presented: First, tasks are clustered. Second, the task-clusters are allocated to processors. Clustering reduces the size of the allocation design-space. Tasks are clustered in pairs so that tasks which exchange much data are put into a cluster. This reduces the communication demand between processors.

In [54] the optimization-objective is to minimize the max. task response time. The author uses a *branch and bound* approach with a tailored heuristic for exploring high quality parts of the solution space, while much of the search tree can be bounded. The exhaustive nature of branch and bound algorithms ensures that an optimal solution is guaranteed.

In [71,72] an approach similar to the two-stage approach (i.e. clustering and allocation) is applied to solve the TAP for *safety-critical hard real-time systems*. Here, tasks may have replicated tasks, and the replica-tasks must be allocated onto separated processors. The approach constructs a static cyclic schedule.

In [67] the authors present a heuristic for allocating hard real-time tasks to processors which are connected to a multi-cluster network (i.e. time-triggered and event-driven bus-systems). Tasks with different trigger-method (time- vs. event-triggered) can be mapped to the same processor, since processors use a hierarchical scheduling policy.

In [68] the approach from [67] is extended with respect to *partitioning*. This means that the algorithm decides if a task shall be implemented as a time-triggered task or as an event-triggered task. On the other hand, this approach assumes that tasks with different trigger-method (time- vs. event-triggered) must not be mapped to the same processor.

In [37] the authors address the TAP for mixed-critical tasks. They assume that the WCET of the tasks is dependent on the criticality-level. The authors propose an allocation algorithm, which is based on bin-packing heuristics.

2.5.3 Meta-Heuristic Search and Optimization

Besides using problem-specific heuristics, a widely used approach is to apply meta-heuristics.

Simulated Annealing

In [81] the authors use *simulated annealing* (SA) to solve the TAP for hard real-time systems. The tasks are scheduled according to fixed priority scheduling, and priorities are assigned according to DMPO. The allocation must satisfy several constraints (resource utilization, timing, replicated tasks to be allocated to separate processors).

In [15] the authors apply SA to the TAP with static cycle scheduling (i.e. time-triggered scheduling). Their SA-approach uses different neighbour moves at different stages of the search and depending on the current solution. For example, if the processors are utilized very unbalanced, then neighbour moves are used which improve load balancing. Therefore, heuristic information is being encoded into the neighbour moves, rather than the cost function. The schedule is created according to a *latest possible start time* heuristic.

In [75] the author apply SA to solve the TAP for homogenous multi-DSP systems with point-to-point communication. The approach considers task-to-processor constraints and task-replica constraints. In addition, a static schedule is generated for each processor.

Genetic Algorithm

Unlike SA, which is based on the manipulation of a single solution, *genetic algorithm* (GA) is based on a population. Within that population *genetic operations* (e.g. re-combination, mutation, selection, etc.) are applied, in order to generate new solutions.

In [79] GA is used to solve the TAP for a soft real-time application. The authors propose an improved way to encode the genome, resulting in an improved search-performance. The approach is demonstrated on a sensor fusion application. The hardware platform consists of a set of processors, connected via Ethernet.

In [6] the TAP is addressed as part of an architecture synthesis problem. Beside allocation of tasks, also scheduling of tasks and selection of hardware (i.e. ASIC vs. general purpose processor) is addressed. The author applied 3 different meta-heuristics:

SA, GA, and tabu-search. The author's conclusion is that tabu-search being the best approach for this particular problem.

2.5.4 Alternative Formulation

In [85] the TAP (as well as the priority assignment) for hard real-time tasks is formulated via *mixed integer linear programming* (MILP). Processors, which may have different processing capabilities, are interconnected via a CAN bus-system. Optimization-objective is to minimize end-to-end latencies.

In [25] the TAP is solved, while taking into account several constraints (e.g. limited resources, timing, dedicated processors, task grouping, task separation). The authors apply a constraint programming approach to solve the problem.

2.6 Automotive Standards

Within the automotive domain, a set of standards have been established which are used to implement automotive embedded real-time systems. A sub-set of them – which are most relevant for this thesis – are covered in this section.

2.6.1 CAN

The *controller area network* (CAN) standard [14] specifies a wired serial communication bus. It allows to transmit up to 8 bytes of payload per frame. The baudrate has to be set in alignment with cable length and topology. In automotive systems, typical baudrates are 125, 250, 500 and 1000 kb/s.

CAN is a multi-master bus. The arbitration of the bus is based on bit-wise CSMA/CA (carrier sense, multiple access, collision avoidance). Each frame has an unique ID which is used for frame-identification, and also determines the frame's priority. Thus, CAN can provide real-time behaviour, and can be analysed according to *fix priority non-preemptive scheduling* [21].

The *time-triggered CAN* (TT-CAN) standard [82] is an extension towards time-triggered scheduling, and represents a higher-level protocol for CAN.

2.6.2 LIN

The *local interconnect network* (LIN) standard [43] specifies a low-speed wired serial communication bus. It allows transmitting up to 8 bytes payload per frame. The baudrate is up to 19.2 kb/s.

LIN is a master/slave bus. The LIN-master sends requests to LIN-slaves, and the LIN-slaves answer (either to the master, or to another slave). The LIN-master schedules the entire communication according to TDMA.

2.6.3 FlexRay

The *FlexRay* (FlexRay) standard [28] specifies a wired serial communication bus which is specially designed towards reliability. It allows transmitting up to 254 bytes of payload per frame. The baudrate is up to 10 Mb/s. FlexRay can be configured to use 2 physical channels in parallel. This can either improve reliability (e.g. by transmitting replicated data via both channels) or increase throughput.

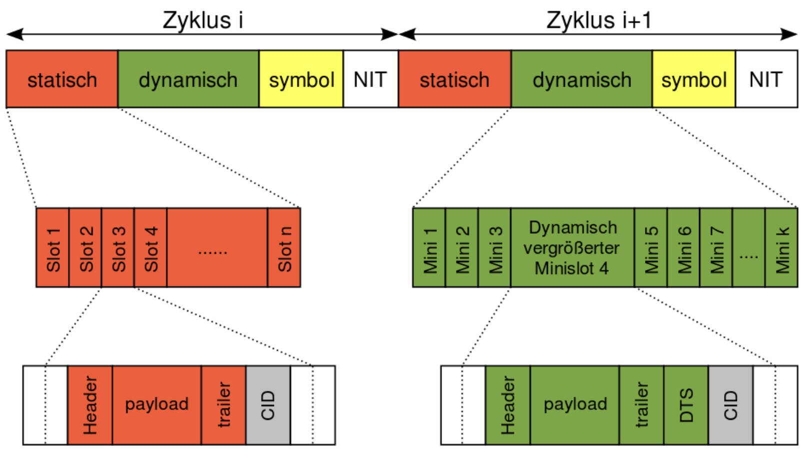


Figure 2.3: FlexRay communication scheduling schema

FlexRay uses a hybrid scheduling schema (see figure 2.3). Basically it uses TDMA. However, each communication-cycle contains 2 segments: In the *static segment* frames are scheduled according to TDMA. In the *dynamic segment* frame-arbitration is based on frame-priority.

2.6.4 OSEK/VDX

Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug (English: Open Systems and the Corresponding Interfaces for Automotive Electronics) / Vehicle Distributed eXecutive (OSEK/VDX) is a standard [52] of the software-platform for automotive systems. It contains 3 major modules:

- *operating system (OS)* – real-time execution of ECU software and base for the other OSEK/VDX modules
- *communication (COM)* – data exchange within and between control units
- *network management (NM)* – configuration determination and monitoring

As part of the *operating system* specification OSEK/VDX specifies the real-time scheduler. It applies *fix priority preemptive scheduling*. Thus it can be analysed [33] using known RTA methods. However, it also supports non-preemptive scheduling. The extension *OSEK-time* allows time-triggered scheduling.

As part of the *communication* specification OSEK/VDX specifies the communication mechanisms between software components within the same processor, as well as the communication-stack for automotive bus protocols (such as CAN).

2.6.5 AUTOSAR

The *automotive open system architecture* (AUTOSAR) standard [5] is currently one of the most important standardization efforts in the automotive domain. AUTOSAR builds upon a set of other standards, and is focused on 3 main topics:

1. *methodology* – Definition of a methodology, how AUTOSAR-compliant automotive software-systems shall be developed (see figure 2.5). It defines all engineering steps, and which engineering-data is exchanged in between these steps.

2. *system description* – Definition of a language / format for describing an AUTOSAR-compliant system. All elements in the language have a precise semantic. The persistent representation is XML.

3. *middleware* – Standardization of a middleware for automotive software-systems. The middleware is composed of a set of basic software components. Each component has a precisely defined functionality and interfaces. The components can be grouped according to their responsibilities: I/O drivers, communication-stack for bus-protocols, memory management, operating system. Much of this specification is closely linked to the OSEK/VDX standard. On top of this middleware, application software can be executed. The application software is separated from the underlying middleware by the runtime environment (RTE). The RTE provides a set of standardized interfaces.

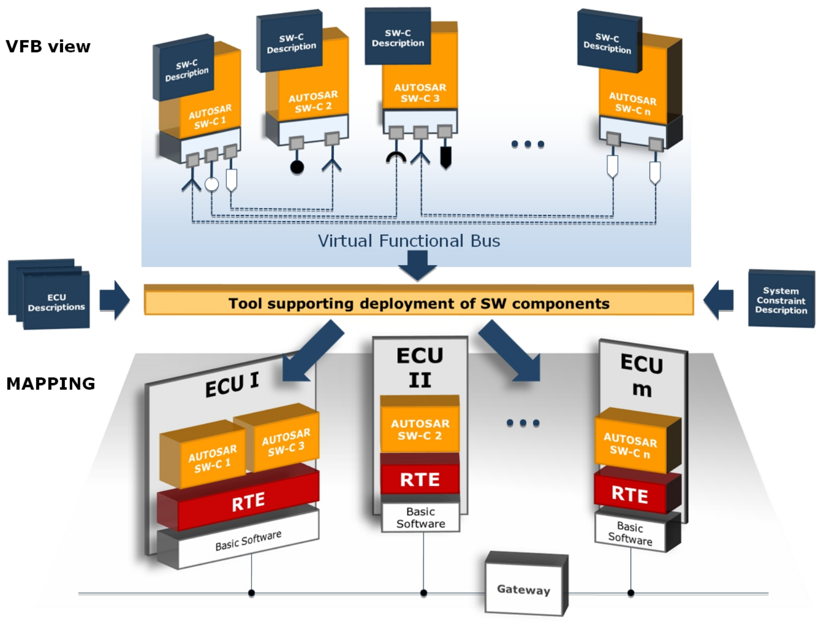


Figure 2.4: Basic AUTOSAR approach [5]

Figure 2.4 outlines the basic AUTOSAR approach: The hardware-independent software-components are mapped to ECUs, which again determines the communication demand between processors. Once mapped, communication between software-components is established via the middleware (indicated by *RTE* and *basic software* in the figure).

In terms of the operating system, AUTOSAR requires an OSEK/VDX-compliant operating system. Thus, scheduling is performed according to *fix priority preemptive scheduling*. Adequacy between AUTOSAR and the real-time scheduling theory is discussed in [32].

AUTOSAR Methodology & Thesis Goal

The AUTOSAR methodology describes the engineering steps which need to be performed for designing an AUTOSAR-compliant system. The methodology is shown in figure 2.5. The goal of this thesis is to provide solutions which can be applied to a sub-set of these design steps. The focus is on the following steps:

- *System Configuration Generation* – allocation of software components to processors, routing of data via bus-systems and gateways, configuration of bus-systems
- *ECU Configuration Generation* – scheduling the execution of software components

However, the approaches presented in this thesis are not solely focused on automotive. The methodologies are generally applicable.

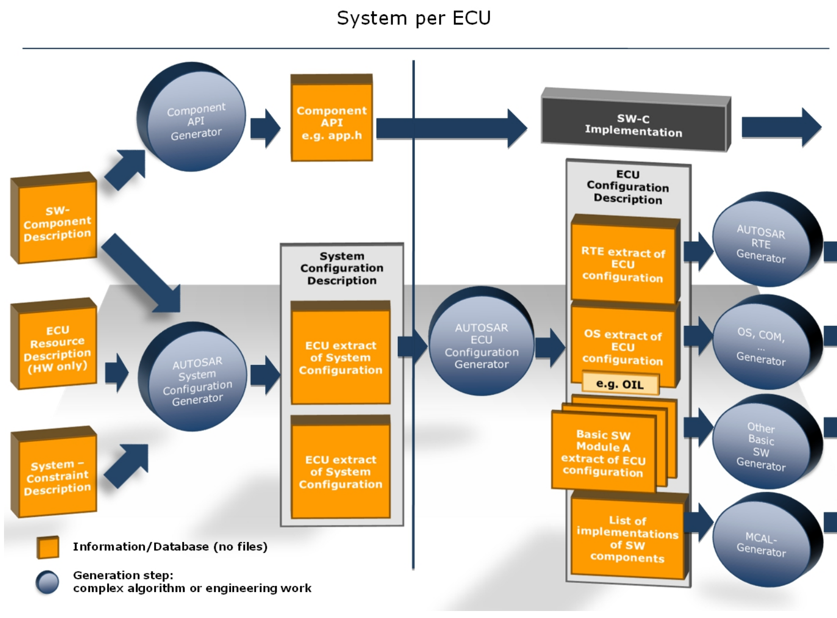

System per ECU
Component API Generator
Component API e.g. app.h
SW-C Implementation
SW-Component Description
ECU Resource Description (HW only)
System – Constraint Description
AUTOSAR System Configuration Generator
System Configuration Description
ECU extract of System Configuration
ECU extract of System Configuration
AUTOSAR ECU Configuration Generator
ECU Configuration Description
RTE extract of ECU configuration
OS extract of ECU configuration
e.g. OIL
Basic SW Module A extract of ECU configuration
List of implementations of SW components
AUTOSAR RTE Generator
OS, COM, ... Generator
Other Basic SW Generator
MCAL-Generator
Information/Database (no files)
Generation step: complex algorithm or engineering work

Figure 2.5: Methodology for designing AUTOSAR-compliant system [5]

Chapter 3

Research Question

3.1 Conclusion and Open Issues of State of the Art

Although there has been a substantial amount of research on the TAP (and related sub-problems) over the last decades, the following open issues can be identified:

- Almost all works that address the TAP tackle cross-processor communication in a simplified way: If the size of a message is smaller than or equal to the max. payload of a bus frame, then each message is packed into a separate bus frame. If the size of a message is larger than the max. payload of a bus frame, then the message is split into several parts, each part is transmitted in a separate bus frame and the message parts are re-joined (in order to build the initial message) at the receiving processor. However, if the size of a message is smaller than the max. payload of a bus frame, almost no work considers packing several messages into a single bus frame. The few works that consider this, only loosely (or not at all) incorporate this approach into the TAP *design space exploration* (DSE).
  By these simplifying assumptions, one core-problem of automotive real-time system configuration, namely the configuration of the communication infrastructure, is not addressed in required detail.

- Almost all works that address the TAP focus on solving the problem "from scratch". Algorithms have a large degree of freedom while searching for a system configuration. Although some works try to find configurations that are robust against future system modifications (e.g. varying WCET of tasks), almost no work can be found that actually addresses *system configuration upgrade* scenarios, where a system configuration needs to be found that is based on an initial system configuration.
  However, in the automotive domain, systems are usually designed by taking an initial system as the starting point and then 1) *improve* this configuration, or 2) *upgrade* this configuration, in order to meet the new requirements.

- Since automotive systems are design in an *evolutionary* manner (by improving and upgrading an initial system), a large set of constraints may be in place. These constraints represent *legacy system configuration decisions.* While performing *improvement*- or *upgrade*-scenarios, these additional constraints need to be considered. However, most works do not take into account all of these constraints.

Although several of these additional constraints may be incorporated in state-of-the-art TAP solving frameworks, the resulting TAP solving performance may become poor. Therefore, the additional constraints need to be handled in an efficient way, in order to achieve satisfactory TAP solving performance.

3.2 Hypothesis

Methods for automated search for optimal system configurations are accepted in the automotive domain, if and only if:

- all relevant sub-problems are addressed / solved in sufficient detail
- all relevant standards are taken into account
- all relevant domain-specific constraints are satisfied
- the way these methods work is transparent and can be understood by engineers (who nowadays are designing system configurations by hand)
- these methods can be extended with moderate effort, in order to satisfy future demands / issues
- the optimal system configuration satisfies all relevant *legacy decisions*
- the optimal system configuration can be upgraded later on, by having to apply only moderate changes

3.3 Research Questions

The overall objective of this work is to provide methods for automated search for optimal system configurations (especially for automotive real-time systems). In order to achieve this objective, the following research questions are answered in this thesis:

- *How can the configuration of the communication infrastructure be solved in a realistic way?* – This is answered in chapter 5.
- *How can design constraints (that mainly stem from legacy decisions and safety-relevant considerations) be incorporated into the search, and be satisfied in an efficient way?* – This is answered in chapter 6.
- *How can a system configuration be designed so that it can handle future modifications? How can a given system configuration be extended / upgraded in the future, in order to meet new requirements?* – These are answered in chapter 7.

Chapter 4

Search Framework for Finding Optimal System Configurations

The overall objective of this work is to provide a methodology for finding *near-optimal system configurations* of distributed real-time systems, in a fully automated way. The methodology is implemented inside a search framework. In order to find a system configuration, the following design steps need to be performed:

- *task allocation* – local assignment of tasks to processors
- *message routing* – local assignment of messages to bus systems (and gateways)
- *frame packing* – packing of messages into bus frames
- *system scheduling* – temporal planning of system execution
- *system performance analysis* – schedulability, utilization, ...

All these steps are usually performed by engineers, and complex design decisions must be taken, in order to be able to perform these steps. Consequently, the search framework needs to be able to take the same design decisions.

Ultimately the search is executed on a computer platform (e.g. personal computer). Thus, the system, for which a system configuration should be found, must be represented in a way that can be handled by the search framework. Therefore, a *system model* is used that captures all relevant attributes of the system.

4.1 System Model

The system is represented by two models. These models are commonly used in the literature.

- *software* – tasks, messages (task-graph)
- *hardware* – processors, bus systems, bus frames (processor-network)

By performing system configuration steps, the two models are merged into one system model. The dependencies between the two models are uniquely described by the *system configuration*. This configuration is the overall output of the search.

In the literature, the task-graph must be non-cyclical. This is because of some limitations of schedulability tests. If there is a cycle, the release jitter (and thus also the WCRT) becomes infinitely large, because of the circular dependencies. However, this restriction can be weakened: The task-graph may contain cycles, if and only if at least one element of the cycle is time-triggered. The time-triggered element does not inherit the release jitter from its precedent's element WCRT. This stops the circular increase of the release jitter. For the task-graph one assumption must be satisfied: The data size of a message must be smaller than (or equal to) the max. payload of a bus frame. This ensures that frame packing can be performed.

For the processor-network one assumption must be satisfied: There exists at least one path/route between each pair of processors. Consequently, the processor-network must not consist of two (or more) independent sub-processor-networks. This assumption is also in alignment with the *domain of interest* (which is automotive). However, this assumption does not mean, that the methodology is not generally applicable to systems which do not satisfy the mentioned assumption. Why not? Any system that contains independent sub-processor-networks can only be configured in a feasible way, if and only if it also contains independent sub-task-graphs. Such a system can always be split into smaller sub-systems, and be solved at the sub-system level. Finally the system can be re-build, by merging the solved sub-systems.

4.1.1 Limitations

There is (at least) one case, in which the traditional task-graph representation runs into difficulties. Assume the model shown in figure 4.1. Depending on the chosen task allocation, different message routing is needed, and also has impacts on subsequent design steps (see table 4.1). Configuration B is hard to model using the traditional task-graph. The main problems occur during frame packing and scheduling, since the same message needs to be used several times within different frames and buses.

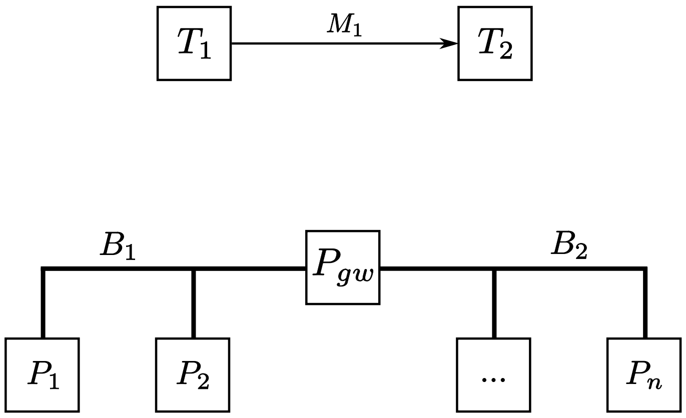


Figure 4.1: Model of multi-bus system

	Configuration A	**Configuration B**
task allocation	$T_1 \to P_1$ $T_2 \to P_2$	$T_1 \to P_1$ $T_2 \to P_n$
message routing	$M_1 \to B_1$	$M_1 \to \{B_1, P_{GW}, B_2\}$
frame packing	$M_1 \to F_1$	$M_1 \to \{F_1, F_2\}$
frame scheduling	$F_1 \to B_1$	$F_1 \to B_1$ $F_2 \to B_2$

Table 4.1: Impact of task allocation on subsequent design decisions

4.1.2 Extensions to Overcome Limitations

A simple approach can be used, in order to avoid these problems. The task-graph is represented by two views:

1. *application view* – This view represents the traditional task-graph. Within this model, task allocation and resulting message routing can be performed. However, frame packing and system scheduling is not performed in this model.

2. *implementation view* – This view is a more detailed representation of the application view. An *application view message* that is routed via several bus systems, is represented by several *implementation view messages*. Each of these messages is assigned to one bus of the route. In addition, *gateway tasks* are added each time a message is routed via a gateway. These tasks represent the fact that data must be transferred from one bus-interface to the other bus-interface of the gateway processor (which obviously takes some time).

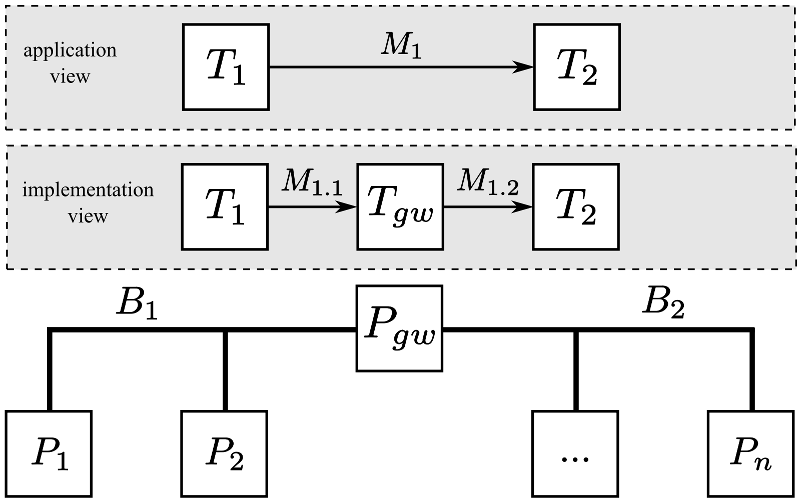


Figure 4.2: Transformation between application view and implementation view of multi-bus system

Figure 4.2 shows how configuration B looks like in the *implementation view*. Therein, each message is only assigned to one bus. Because of this, no problems occur during frame packing and scheduling. After all system configuration steps have been performed and system performance is evaluated, relevant data (e.g. WCRT, utilization, ...) are transferred back into the *application view*.

The *implementation view* not only helps to avoid some problems during finding a system configuration, it is also a more realistic representation of the real system. If a message is routed via several bus-systems, then several instances of this message do exist. Each of these instances is packed into different frames, and has different attributes (e.g. WCRT). Gatewaying the message needs processing resources on the gateway and takes time. Table 4.2 shows the equivalences between the system model and the AUTOSAR model.

System Model	**AUTOSAR**
task	SWC, runnable
message (appl. view)	system signal
message (impl. view)	signal, I-PDU, N-PDU
frame	frame, L-PDU
bus	bus
processor	ECU

Table 4.2: Equivalences between system model and AUTOSAR model

The transformation between application view and implementation view only concerns messages. All other elements (tasks, bus-systems, processors) have a 1:1 relation. For messages, the following transformation rules apply:

- 1 application message $\Rightarrow$ 1 implementation message
  if the application message is sent processor-internal

- 1 application message $\Rightarrow$ N+1 implementation messages + N gateway tasks.
  N is the number of gateways the application message is routed via.

- The period of the application message is inherited to all implementation messages and gateway tasks. The data size of the application message is inherited to all implementation messages.

The processing demand of the gateway tasks is modeled as follows: The processing demand consists of a constant term and a linear term. The constant term represents the effort of invoking the gateway task, the linear term represents the effort of transferring the message data from one bus-interface to the other bus-interface. A linear relationship between message data size and processing effort seems appropriate.

$$Z_{gw} = Z_{\text{init}} + a \cdot s_m \tag{4.1}$$

If gateways are passed by the application message, the deadline of the application message needs to be represented by an end-to-end-delay deadline in the implementation view. In addition, individual deadline could be derived for the involved elements. This is not necessary, but can be useful for scheduling (e.g. by applying DMPO). The following approach of splitting the end-to-end-delay deadline is proposed: The implementation messages should be assigned a deadline which is indirect proportional to the bus baudrate they are sent via. The rational behind this approach is: A higher baudrate enables smaller transmission time, and thus smaller deadline can be assigned.

$$D_{\text{end-to-end}} = D_1 + D_2 + \cdots + D_n = b\left(\frac{1}{br_1} + \frac{1}{br_2} + \cdots + \frac{1}{br_n}\right) \tag{4.2}$$

The approach of using two views has one significant consequence: The implementation view is dynamic. This means that the number of tasks and messages in the implementation view are varying. This has some implications and consequences on *where* and *how* certain design steps are performed. Details will be discussed at later point of this work.

4.2 Basic Search Framework – Simulated Annealing

The problem of finding an optimal system configuration is addressed using a meta-heuristic search algorithm, called SA. It is a well known algorithm in the domain of artificial intelligence. Its name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material [38]. SA has already been applied to solve the TAP [81]. However, the main reason for using SA is because it is shown in [26] how SA can be tailored towards solving different aspects of system configuration: Multi-mode systems, fault-tolerant and graceful degrading systems, as well as flexible systems (systems that can be adapted to future requirements with small number of changes).

Also, SA is a relatively simple meta-heuristic, and thus easy to understand. This is especially benefitial if the search has to be adopted to future requirements. Due to its simplicity, only a few few elements need to be adopted in order to apply SA to a specific problem. By using this meta-heuristic search technique, the entire problem is split into two major parts:

- system configuration generation
- system configuration evaluation

Starting from an initial configuration, a new configuration is generated. This new configuration is evaluated, and a *cost* is calculated. Based on the cost of a new configuration, the algorithm decides if the new configuration is accepted, and thus becomes the source for subsequent exploration. These main steps are repeated until a termination criteria is reached. Algorithm 4.1 shows the overall search algorithm.

The search algorithm consists of the following main methods:

- *neighbour* – Create a new configuration, by applying modifications to the current configuration.
- *cost* – Evaluate a configuration, according to a cost-function, in order to determine the quality of a configuration.
- *accept move* – Determine, if the new configuration should be accepted, and thus become the source for subsequent exploration steps. This decision depends on the cost-values and the current temperature.

Algorithm 4.1: System configuration optimization algorithm (based on simulated annealing)

```
Input: config.init /* initial configuration */
Data: T /* temperature */
Data: iter.max /* max.  iterations */
Data: iter.at.T.max /* max.  iterations at constant T */
begin SystemConfigurationSimulatedAnnealing
    /* initialize */;
    cost.init = cost(config.init);
    config.current = config.init /* start at initial configuration */;
    cost.current = cost.init;
    config.best = config.init;
    cost.best = cost.init;
    /* search, until stopping-criteria is reached */;
    while stop() = false do
        while iter.at.T < iter.at.T.max do
            /* propose new configuration */;
            config.new = neighbour(config.current);
            cost.new = cost(config.new);
            /* accept move? */;
            if acceptMove() = true then
                /* improvement of best configuration? */;
                if cost.new < cost.best then
                    /* remember best */;
                    config.best = config.new;
                    cost.best = cost.new;
                end
            end
            iter.at.T++;
            /* next iteration at constant T */;
        end
        cool(T);
        iter.at.T = 0;
        /* resume search at lower T */;
    end
end
Output: config.best /* best configuration found */
```

- *cool* – Over time, the current temperature is decreased, due to a cooling schedule. Lower temperature decreases the probability of accepting *worse* configurations.
- *remember best configuration(s)* – The currently best obtained configuration can be stored. Alternatively, a set of *near-optimal configurations* can be stored, so that the user can make the final decision on which configuration to pick.

4.2.1 Accept Move

A search algorithm is called *greedy* if it only accepts solutions with improved cost. This has one major disadvantage: The search algorithm is easily trapped inside a local optimum (e.g. minimum).

There are some ways to prevent a search algorithm to get trapped in a local optimum. E.g. the search can be restarted at a randomly picked starting position. Another approach is to allow exploration steps that do not improve cost. This way, the search algorithm can escape from local optima.

Within SA, this concept is incorporated. After the cost of a configuration is evaluated, a decision is made, if the new configuration should be picked:

- If the new configuration improves the cost of the current configuration, then the new configuration is always picked.
- If the new configuration does not improve the cost of the current configuration, then the new configuration is picked according to an *acceptance probability.*

The idea behind is as follows: At the beginning of the DSE, a wide design space should be covered. Therefore, configurations that do not immediately improve the cost should also be picked for subsequent exploration. This represents some kind of pseudo-random DSE. As the search progresses the acceptance probability for *worse* configurations decreases. Thus, almost only configurations that improve the cost are accepted. This represents a narrowed/focused search towards the optimum.

The acceptance probability function that is widely used in SA literature is:

$$P = \begin{cases} 1 & \text{if} \quad cost_{\text{new}} < cost_{\text{current}} \\ e^{\frac{-\Delta cost}{T}} & \text{otherwise} \end{cases} \tag{4.3}$$

with:

$$\Delta cost = |cost_{\text{current}} - cost_{\text{new}}| \tag{4.4}$$

Thus the acceptance probability is determined by the cost difference and the temperature. High temperature ensures high probability. In order to achieve the desired DSE behaviour, an adequate cooling schedule is needed.

4.2.2 Cooling

After a defined number of iterations, the temperature is decreased. The temperature influences the probability of accepting *worse* system configurations.

$$T = T \cdot coolingFactor \tag{4.5}$$

Algorithm 4.2: Accept move to neighbour

```
Input: cost.current
Input: cost.new
Data: T
begin acceptMove
    if $cost.new < cost.current$ then
        /* improvement */;
        accept = true;
    else
        /* no improvement */;
        P = exp((cost.current - cost.new)/T);
        rnd = randomUniform() /* value in [0, 1) */;
        if $P > rnd$ then
            accept = true;
        else
            accept = false;
        end
    end
end
Output: accept
```

The *cooling factor* should be between 0.85 and 0.99. Smaller numbers result in faster cooling, larger numbers result in slower cooling. The cooling speed has a significant impact on the performance of SA. In order to find an adequate cooling factor, some experiments need to be performed. However, a cooling factor of 0.95 seems to be a good starting point, that is also proposed in the literature.

4.2.3 Neighbour

The *neighbour*-function is one of the key-elements, when applying SA to solve a specific search problem. It implements the basic modification steps that can be performed. Generally speaking, these steps represent the modification an engineer would perform manually. The following neighbour steps are used:

- re-allocate a task to another processor
- swap the allocation of two tasks (that reside on different processors)

These steps are very common in the literature. Each of these neighbour steps is performed, due to a defined probability (see table 4.3). This way it is possible to encode domain-knowledge how appropriate each neighbour step is. At the same time, a certain degree of randomness is involved.

Modifying a task allocation is performed using the *application view* of the task-graph. Based on the modified allocations, message routing is performed. Again this is done in the *application view*. Knowing all task allocations and message routes, the *implementation view* is generated, using the transformation rules presented in section 4.1.2. Finally, system schedulability is evaluated, using the *implementation view*.

Neighbour Move	**Probability**
re-allocate	0.80
swap allocation	0.20

Table 4.3: Probability of neighbour moves

Re-Allocate Task

Algorithm 4.3: Re-allocate task

```
   Input: set of tasks
   Input: set of processors
1  begin reAllocateTask
2      task $\tau$ = randomly pick a task from $\mathcal{T} \setminus \mathcal{T}_{fixed}$;
3      processor $\rho$ = randomly pick a processor from $\mathcal{P} \setminus \mathcal{P}_{current}$;
4      allocate task $\tau$ to processor $\rho$;
5  end
   Output: modified allocation
```

Swap Allocation

Algorithm 4.4: Swap allocation of tasks

```
   Input: set of tasks
   Input: set of processors
1  begin swapTaskAllocation
2      task $\tau_1$ = randomly pick a task from $\mathcal{T} \setminus \mathcal{T}_{fixed}$;
3      task $\tau_2$ = randomly pick a task from $\mathcal{T} \setminus \mathcal{T}_{fixed}$ which resides on a different
       processor than $\tau_1$ does;
4      swap allocation of $\tau_1$ and $\tau_2$;
5  end
   Output: modified allocation
```

4.2.4 Cost

Real-time system configuration is driven by a set of objectives and constraints. Commonly used statements are:

- System must be schedulable. No deadline must be missed.
- No resource (processor, bus) must be overloaded.
- A minimum number of processors should be used.
- Processors should be equally utilized. (load balancing)
- The utilization of bus systems should be minimized.

However, SA is driven by only a single value, called *cost*. Thus, if several optimization objectives (and constraints) exist, they have to be combined into a single value. This task is performed by the cost-function. Usually the individual objectives o_i (and constraints) are combined using a weighted sum. Thereby, the weights w_i are used for scaling (to similar ranges) and prioritizing.

$$cost = w_1 \cdot o_1 + w_2 \cdot o_2 + \cdots + w_n \cdot o_n \tag{4.6}$$

If the number of objectives grows, it becomes quite hard to find adequate weights. To overcome this issue, the cost-function can be formulated in a hierarchical way [63]. Thereby, several objectives are combined, forming a sub-cost-function. These sub-cost-functions are then combined forming the cost-function. In theory, several hierarchies can be used. However, a two-level hierarchy is sufficient in most cases. In order to make the cost-function more intuitively (to model and to read), the following concepts should be applied:

- Each objective is formulated in a way, that it returns values between 0 and 1 (where 0 is best and 1 is worst). This way, no scaling between objectives is needed. Thus, weights only represent the *importance* of the individual objective.

- The cost-function is normalized, so that it returns values between 0 and 1. This is done at each level of hierarchy. This makes it more intuitive to read/interpret.

$$cost = \frac{\sum w_i \cdot o_i}{\sum w_i} \tag{4.7}$$

The way the *cost-function* and all its terms are modeled, is one of the most important aspects, when applying SA to a specific problem (such as task allocation). The cost-function shapes the entire design space, which is explored. SA performance mainly depends on the shape of the design space (and thus on the cost-function). One of the most important attribute of the cost-function is that it should not contain *flat regions*. E.g. in order to find a schedulable system, no deadlines must be missed. If the cost-function only outputs 0/1 (in order to represent: any deadlines missed: false/true) then it is hard to find any appropriate search direction. If the cost-function outputs 0..1 (in order to represent: 0..100% of the tasks miss their deadline) then a search direction can be derived.

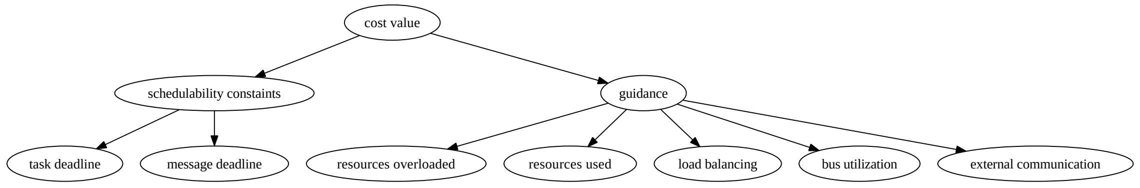


Figure 4.3: Hierarchical cost-function

In this work a hierarchical cost-function is used. The overall cost-value is calculated by combining two cost-sub-functions: schedulability constraints and guidance. These cost-sub-function consist of additional cost-sub-functions.

Schedulability Constraints

This sub-function shall guide the search towards configurations which are schedulable. If the configuration is not schedulable, the degree of un-schedulability can be measured by: how many tasks/messages miss their deadline. This term should be minimized during the search, thus leading to a schedulable system.

$$dv(.) = \begin{cases} 1 & \text{if} \qquad\qquad R > D \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

$$cost_{\text{schedulability constraints}} = \frac{1}{2}\left(\frac{1}{\|\mathcal{T}\|}\sum_{\tau\in\mathcal{T}} dv(\tau_i) + \frac{1}{\|\mathcal{M}\|}\sum_{m\in\mathcal{M}} dv(m_i)\right) \tag{4.9}$$

Guidance Heuristics

Although schedulability is a key-attribute of real-time systems, not all schedulable configurations can be considered equally "good". Several additional attributes can be used (see user-driven optimization objectives). In order to guide the search towards configurations that optimize these objectives, several guiding heuristics are used.

Resource Overload

A configuration must not overload any resource. This would rule out schedulability of the system. In addition, overloading of memory demand would also make correct system behaviour impossible. Thus, overloading must be avoided.

$$ro(.) = \begin{cases} 1 & \text{if} \qquad\qquad u > u_{max} \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

$$cost_{\text{resource overload}} = \frac{1}{2}\left(\frac{1}{\|\mathcal{P}\|}\sum_{\rho\in\mathcal{P}} ro(\rho_i) + \frac{1}{\|\mathcal{B}\|}\sum_{\beta\in\mathcal{B}} ro(\beta_i)\right) \tag{4.11}$$

If u_{max} is set to 100%, a safe upper bound is used. However, the max. can be set smaller. E.g. the max. utilization could be set to 80%, in order to reserve resources for future system upgrades.

Resource Usage

Cost-efficient configurations can perform their computational tasks using a minimum amount of hardware resources. Not all resources that have been considered are used. This attribute can be measured by: how many resources are used.

$$ru(.) = \begin{cases} 1 & \text{if} \qquad\qquad u > 0.0 \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

$$cost_{\text{hardware usage}} = \frac{1}{2}\left(\frac{1}{\|\mathcal{P}\|}\sum_{\rho\in\mathcal{P}} ru(\rho_i) + \frac{1}{\|\mathcal{B}\|}\sum_{\beta\in\mathcal{B}} ru(\beta_i)\right) \tag{4.13}$$

Load Balancing

By equally utilizing the available resources, the probability of finding a schedulable solution is increased. Also, it reserves resources for future system extensions / upgrades.

$$cost_{\text{load balancing}} = \frac{1}{\|\mathcal{P}\|} \sum_{\rho \in \mathcal{P}} |\bar{u} - u_{\rho_i}| \tag{4.14}$$

Bus Utilization

Tasks that reside on different processors need to communicate via external bus systems. However, external communication leads to higher delays, which again is dependent on the bus utilization. Therefore, bus utilization should be minimized.

$$cost_{\text{bus util}} = \frac{1}{\|\mathcal{B}\|} \sum_{\beta \in \mathcal{B}} u_{\beta_i} \tag{4.15}$$

Processor-External Communication

An efficient way to reduce bus utilization is to allocate communicating tasks onto the same processor. Thus communication can be performed processor-internal. This heuristic is often used in the literature. In order to guide SA towards such configurations, the number of messages that need to be sent processor-external is measured / punished.

$$pec(m) = \begin{cases} 1 & \text{if} \quad src_\rho(m) \neq dst_\rho(m) \\ 0 & \text{otherwise} \end{cases} \tag{4.16}$$

$$cost_{\text{processor-external messages}} = \frac{1}{\|\mathcal{M}\|} \sum_{m \in \mathcal{M}} pec(m_i) \tag{4.17}$$

This term only counts the number of messages which are not processor-internal. It does not take into account their communication intensity, since this is already taken into account in the *bus utilization* term.

Term	**Source**	**Description**	**Feasibility**
schedulability	must	penalty for un-schedulable objects	yes
resource overlaod	must	penalty for overloaded resources	yes
resources used	user	reward for non-used resources	no
load balancing	user	penalty for non-balanced processors	no
bus utilization	user	reward for low bus utilization	no
external communication	guiding	penalty for cross-processor messages	no

Table 4.4: Terms of cost-function

Table 4.4 summarizes the terms of the cost-function. It also categorizes the terms by their source (e.g. user-driven), and whether or not they determine the feasibility of a configuration.

4.2.5 Tabu Search

Since SA is based on a randomized search, the question arises: Is it possible (or probable) that SA will traverse the design space in a closed loop, so that it re-visits regions of the design space that it already evaluated? To answer this question, a tabu-list is implemented. In the tabu-list, all configurations that have already been evaluated are stored. Each new configuration is compared to the elements in the tabu-list, and is evaluated only if it is a *unique* configuration.

Experiments show that about 95% of all generated configurations are unique configurations. Thus only 5% of the generated configurations are re-visited. The drawback of the tabu-list approach is its runtime complexity. Each new configuration needs to be compared to all elements in the tabu-list. This has $O(n)$ complexity for each new configuration. Thus, in total this results in $O(n^2)$ complexity, where n is the number of configurations that are evaluated by the SA.

The experiments show that the tabu-list approach does not scale well. Allowing more than 10 000 SA search iterations results in unacceptable runtime. However, applying SA to the TAP typically requires a high number of iterations (e.g. 100 000). Therefore, the tabu-list approach is not applicable. However, the experimental results also show that the number of re-visited configuration is sufficiently low (it is only 5%). Thus, the benefits of the tabu-list do not out-balance the computational costs. Therefore, the tabu-list is deactivated during subsequent experiments.

4.2.6 Parameter Identification

Both the SA parameters (i.e. initial temperature, cooling rate) as well as the weights for the cost-terms need to be set. This is a challenging task. In order to find adequate settings, a systematic approach is advised [69]. Since the search-framework which is used in this thesis is similar to the one in [26], and the search-problem is similar as well, the parameters from [26] are taken as a reference point.

4.3 Extensions

This initial search framework is based on state-of-the-art approaches to find system configurations. However, some essential problems are not addressed:

- The configuration of the communication infrastructure is done in a too simplified way.
- Domain-specific constraints are not addressed, thus such constraints are not handled in an efficient way.
- The system configuration problem is solved "from scratch". Configurations are not evolving from existing configurations. Thus, *system configuration upgrade* scenarios are not supported.

Within the next chapters these open issues are addressed, thus answering the research questions stated in chapter 3. While answering these research questions, the search framework is extended by:

- In chapter 5 the configuration of the communication infrastructure is addressed in detail. Therefore, the *cost function* is extended. In addition, solving heuristics are added.
- In chapter 6 domain-specific constraints are included into the system configuration problem. In order to satisfy these constraints in an efficient way, *neighbour moves* are adopted and additional *cost terms* are introduced. In addition, a pre-processing phase is added.
- In chapter 7 the system configuration problem is solved in an evolutionary way. Therefore, additional *cost terms* are introduced and *solving heuristics* are provided.

Chapter 5

Configuration of Communication Infrastructure

In order to enable communication between tasks that reside on different processors, a communication media is needed. In the early days of automotive electronics, the number of tasks and processors was small. Thus, little communication between processors was needed. Therefore, each data exchange could be performed via a dedicated physical line. As the number of tasks and processors increased, so did the need for data exchange between them. Soon, dedicated physical lines were not appropriate any more, due to cost and weight constraints. In order to enable cost- and weight-efficient communication between processors, a new communication media was introduced: the bus system. A bus can be seen as *several data exchange multiplexed on one physical line.* Nowadays several different bus systems (LIN, CAN, FlexRay, *media oriented systems transport* (MOST), ...) are used in automotive, each tailored to different requirements (such as cost, speed, reliability, ...).

In the automotive domain, different parts of the entire electronic system are typically developed by different *supplier*s (Tiers). E.g. each processor (and its software) is developed by another Tier, and integrated by the *original equipment manufacturer* (OEM). Consequently, the bus systems are one of the key-elements for *system integration*, since all communication between processors needs to be performed via the bus systems. Thus the configuration of the bus systems is a crucial issue. Typically, several bus systems are used within a car. Therefore, the inter-connecting processors (gateways) are becoming more and more important.

The configuration of the entire communication infrastructure (consisting of bus systems and gateways) is a challenging and crucial issue, and has significant impact on the overall system performance. However, state-of-the-art works on system configuration often simplify the configuration of the communication infrastructure. In [66] the authors clearly point out this problem:

> *"Researchers have often ignored or very much simplified the communication infrastructure. One typical approach is to consider communications as processes with a given execution time (depending on the amount of information exchanged) and to schedule them as any other process, without considering issues such as communication protocol, bus arbitration, packaging of messages,*

clock synchronization, and so on."

– Pop et al. [66]

To overcome the drawbacks of these state-of-the-art approaches, methods for detailed configuration of the communication infrastructure are presented in this chapter.

5.1 Introduction, Assumptions, and Definitions

The term *communication infrastructure* (as used in this chapter) comprises all bus systems and the gateways which interconnect these bus systems, as well as the communication-stack of the involved processors.

The configuration of bus systems consists of the following steps. At the hardware level:

- *topology design* – It has to be decided how the processors are interconnected via bus systems. Depending on the bus technology, different topologies are possible (e.g. star topology for FlexRay). Also, if more than one bus system is used, gateways have to be inserted in order to interconnect the bus systems.
- *bandwidth configuration* – Bus systems can be used at different baudrates, thus providing different bandwidth. However, the max. baudrate is constrained by the used protocol, topology and the environment the bus is used in.

At this point, it is assumed that the configuration at the hardware level is already done and cannot be changed. At the middleware level, the following configuration steps have to be performed:

- *message routing* – Based on a given task allocation, messages need to be routed from source processor to destination processor. This may involve several bus systems and gateways.
- *frame packing* – Messages have to be packed into bus frames, before they can be transmitted. Packing should be bandwidth-efficient.
- *gateway configuration* – If messages/frames are transmitted via several bus systems, they have to be processed by the interconnecting gateways. Frames can either just be forwarded to the next bus system, or frames can be un-packed and the individual messages are again packed into frames for the next bus system.
- *scheduling* – A schedule for frame transmission has to be found for all bus systems.

Currently bus topologies in the automotive domain are still (more or less) simple. Although several bus systems are used, they are usually connected via a single *central gateway*. This pattern may be used at several levels (high-speed, medium-speed, low-speed). Thus, message routing is a simple task, since there is only one route from the source processor to the destination processor. In some cases there may be more routes, however, the number of routes is very small.

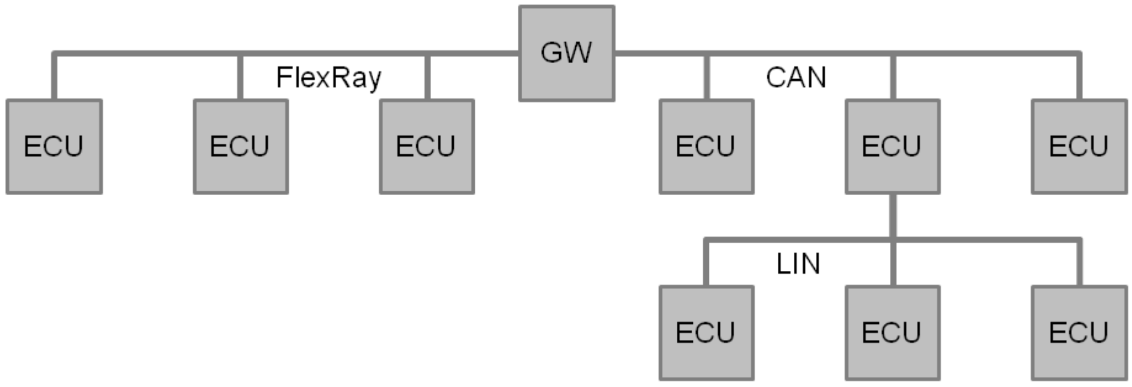


Figure 5.1: Bus-topology of a medium-class car, featuring different bus-system interconnected via gateways

Gateways are special processors that are connected to several bus systems. They can either just process the frames of the connected bus systems, or in addition perform computational tasks. In alignment with the AUTOSAR standard (module: network management gateway), the following assumptions are used in this work:

- Besides processing the frames of the connected bus systems, a gateway can additionally perform computational tasks.
- A gateway may un-pack frames and re-pack the individual messages into new frames.

5.2 Frame Packing – Why?

Bus systems are used to enable cost- and weight-efficient communication between processors. Since several processors are using the same bus, some kind of resource sharing / arbitration protocol is needed. This can either be a time-triggered (LIN, TT-CAN, FlexRay) or a priority-based (CAN) approach.

Because of these protocols, not only the application data is transmitted, but also some protocol overhead data. Application messages are packed into a bus frame, before they can be transmitted via the bus. The frame not only contains the application messages, but also protocol-specific data (e.g. header, checksum, ...). This protocol overhead data is needed for arbitration, routing, data protection, etc.

The bandwidth that can be provided by a bus system is limited by the interaction of the following properties:

- arbitration schema (time-triggered, priority-based)
- propagation delay of one bit
- cable length
- cable material (and shielding)

- environment the bus is operated in (having an effect on the electro-magnetic interference)

To put it short: The bandwidth of a bus system is a limited resource, and cannot easily be increased. E.g. the baudrate of CAN is limited to 1 Mbit/s for a cable length of up to 40 m. Thus, the available bus bandwidth has to be utilized in an efficient way.

However, most works on the TAP that can be found in the literature neglect this important issue. They assume a simple frame packing, where each message is packed into a separate bus frame. This simplistic assumption has several negative implications/effects:

- high number of frames are sent via the bus
- bad ratio between transmitted payload vs. transmitted bits (due to the overhead bits)
- high utilization of the bus
- a lot of bandwidth is wasted

These effects are small, if the data size of a message is similar to the max. payload of a frame. However, in several applications (e.g. in the automotive domain) the data size of a message is significantly lower than the max. payload of a bus frame. Typically, the range of data of messages (in the automotive domain) is between 1 and 16 bits, whereas a bus frame can contain up to 64 bits of payload (CAN, LIN). The overhead of a frame is 64 bits (LIN, CAN). Thus the efficiency of the bus is:

- 50% when using all 64 bits of payload and 64 bits of overhead (64 : 64+64)
- 1.5% when using only 1 bit of payload and 64 bits of overhead (1 : 1+64)

Even in the best case, only 50% of the bandwidth can be used for the transmission of actual data, the rest is consumed by the protocol overhead. Thus the main issue is to use the available bandwidth as efficient as possible. This is why frame packing is heavily used in the automotive domain. Analysis of some real systems shows that typical frames contain between 48 and 64 bits payload.

In addition, another interesting effect can be seen, if simple frame packing (1 message = 1 frame) is used: Due to the high number of frames that are generated, the calculated WCRT of these frames is increased, especially for low priority frames. Thus the system may be deemed to be unschedulable. In reality though, several messages would be packed into a single frame, thus having less frames, and consequently having less interference on the bus. This leads to significantly smaller WCRT of the frames.

It could be argued that this pessimism can be accepted, since it will lead to the fact that the real system will perform better than the worst case estimation. However, there is a major issue: System configurations that are deemed to be unschedulable are usually rejected or punished by the TAP optimization algorithm (since in general only feasible configurations are of interest). Thus, by rejecting a configuration that is deemed to be unschedulable, it is likely to reject a configuration that is actually a feasible (and potentially good) configuration, if a "realistic" frame packing would have been used.

Conclusion

In order to generate realistic system configurations, realistic frame packing approaches need to be used. These approaches should be bandwidth-efficient and result in schedulable frame-sets. The goal of this chapter is to provide methods for building such frame packing configurations.

Symbols

Besides the symbols that are already used, the following additional symbols are used in the context of *communication infrastructure configuration*, especially in the context of *frame packing*.

Symbol	**Description**	**Unit**
s_m	data size of message	bits
pay_f	payload of frame	bits
pm_f	max. payload of frame	bits
oh_f	overhead of frame	bits
bw	bandwidth demand	bits/s

Outline

First, the frame packing problem is defined 5.3, and an overview of state-of-the-art solving approaches is provided 5.4. Second, an improved frame packing approach is presented 5.5. Third, it is shown, how the frame packing problem can be incorporated into the task allocation problem 5.7. Finally, it is shown how the frame packing problem is connected to the gateway configuration problem 5.8.

5.3 The Frame Packing Problem

The *frame packing problem* (FPP) is defined as follows:

> A set of messages $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ must be packed into a set of bus frames $\mathcal{F} = \{f_1, f_2, \ldots, f_k\}$, subject to the constraint that the set of messages in any frame fits that frame's max. payload.

Usually, the FPP is stated as an optimization problem. The most common optimization objectives are:

- minimize the number of needed frames
- maximize the schedulability of the resulting frame-set

A message is defined by its key attributes: the sending processor ρ_s, the receiving processors $\mathcal{P}_r$, its data size s_m, the period T_m at which the data is refreshed, and its deadline D_m. A frame is defined by: the sending processor ρ_s, the receiving processors $\mathcal{P}_r$,

Algorithm 5.1: System-level frame packing

```
Input: buses
Input: processors
begin SystemLevelFramePacking
    frames = { };
    foreach processor do
        foreach bus interface do
            messages = processor.busInterface.outMessages;
            frames += packMessagesIntoFrames(messages) /* add to list */;
        end
    end
    calcTiming(frames) /* T, D */;
end
Output: frames
```

its payload pay_f, its max. payload pm, the period T_f at which the frame is sent, and its deadline D_f. In general each frame may have its individual max. payload (depending on the bus protocol, e.g. CAN), however, usually all frames on the same bus have the same max. payload.

The FPP has to be solved for each bus-interface of each processor independently (see algorithm 5.1) as only messages which originate from the bus-interface of the same processor can be packed into the same frame.

The FPP can be seen as a special case of the *bin packing problem* (BPP). However, the FPP has some special attributes. Thus, BPP algorithms cannot be directly applied for solving the FPP. The main reason is that in the BPP the items have only one attribute (i.e. size), whereas in the FPP the messages have several attributes (i.e. data size, period, deadline). Nonetheless, the main concepts of the heuristics for the BPP can be used for the FPP.

5.3.1 The Bin Packing Problem

The BPP[1] is defined as follows:

> A list of *items* $A = \{a_1, a_2, \ldots, a_n\}$ must be packed into a minimum-cardinality set of *bins* $B_1, B_2, \ldots, B_m$ subject to the constraint that the set of items in any bin fits within that bin's capacity.

Based on this definition, the following equivalences between the BPP and the FPP can be identified (see table 5.1):

In the literature several algorithms can be found for solving the BPP. [18] gives a survey. In general the algorithms are categorized by:

- *on-line* – A bin packing algorithm is called *on-line* if it packs every item a_i solely on the basis of the sizes of the items a_j , $1 \leq j \leq i$ without any information on

[1]Definition of problem and summary of approximation algorithms are taken from [17].

Bin Packing	**Frame Packing**
item	message
item size	data size of message
bin	frame
bin capacity	max. frame payload

Table 5.1: Equivalence between bin packing and frame packing

subsequent items. The decisions are irrevocable; packed items cannot be repacked at a later time.

- *off-line* – A bin packing algorithm that can use full knowledge of all items for packing A is called *off-line*.

Since the BPP is known to be a NP-hard optimization problem, several heuristics for solving the BPP can be found in the literature. Well known on-line heuristics are:

- *next fit (NF)* – NF makes a single scan through the list A and packs the items one after the other into a unique, *active* bin. In case an item does not fit into the active bin, the bin is *closed* (never to be used again), and an empty bin is *opened* and becomes the new active bin. The runtime of NF is $O(n)$.

- *first fit (FF)* – NF does not use the empty space in closed bins. A simple modification gives the FF. FF also makes a single scan through the list A, but it never closes an active bin. When packing a new item, FF puts it into the lowest indexed bin into which it will fit. A new bin is started only if the item does not fit into any non-empty bin. The runtime of FF is $O(n \log n)$.

- *best fit (BF)* – BF behaves like FF, except that it puts the next item into the bin into which it will fit with the smallest gap left over.

However, these algorithms do not always work well; e.g. on lists ordered by increasing item size. Well known off-line heuristics are:

- *next fit decreasing (NFD)* – First sort the list A by decreasing item size, and afterwards behave like NF.

- *first fit decreasing (FFD)* – First sort the list A by decreasing item size, and afterwards behave like FF.

- *best fit decreasing (BFD)* – First sort the list A by decreasing item size, and afterwards behave like BF.

Since the FPP is solved at design time, off-line BPP algorithms are of main interest for solving the FPP.

5.4 Literature Review Refinement – Frame Packing

Although the FPP is similar to the BPP, there are some differences. In the BPP only two attributes (item size, bin capacity) are used. For those attributes equivalence can be found in the FPP. However, in the FPP additional attributes (message period, message deadline, frame period, frame deadline) need to be taken into account. Unfortunately no equivalences can be found for those attributes in the BPP. Because of this, BPP algorithms cannot be applied directly for solving the FPP. This is why some (however few) FPP algorithms can be found in the literature.

In [74] two algorithms are presented. Both algorithms have a preparation step, in which the messages are sorted in increasing order according to their deadline.

1. *Fixed Frame Size* – A frame (with fixed frame size) is created. Messages (from the sorted list) are packed into the frame, as long as there is space left in it. If a message does not fit into the frame any more, the frame is *closed*, and a new frame is created. From that point on, no more messages are packed into the closed frame (even if a message may still fit).
   This algorithm mimics the *next fit decreasing* heuristic of the BPP. By closing a frame as soon as a message does not fit, frames will not be highly filled. This may seem like a poor decision, but actually it is not. Since the messages are sorted by their deadline, closing frames helps to create frames where the packed-in messages have similar deadlines. Since the deadline of the frame must be the smallest deadline of the packed-in messages, closing frames helps to avoid wasting bandwidth.

2. *Linear Frame Selection* – A frame is created with the min. possible frame size. Messages are packed into the frame. As soon as a message does not fit, a decision is made: Either create a new frame, or increase the size of any existing frame. The decision is made based on the resulting bandwidth costs.
   This algorithm mimics the *best fit decreasing* heuristic of the BPP. Since frames are not closed at the first occurrence of a non-fitting message, the resulting frames are filled close to the capacity. However, the messages inside a frame may have more diverse deadlines, thus resulting in a higher bandwidth wasting.

In these two approaches, schedulability of the frames is not taken into account during packing.

In [73] a slightly modified approach is presented. At first, messages are sorted in increasing order according to their period. The algorithm mimics the *best fit decreasing* heuristic for the BPP. A frame is created, messages are packed-in until a message does not fit. At that point the algorithm jumps to the end of the sorted message-list, creates a frame and packs messages from the end of the list until a message does not fit. The algorithm jumps back to the front of the message-list (to the index where it was before). This is repeated until all messages are packed.

By packing messages from both ends of the list alternately, the algorithm tries to avoid packing messages with too diverse periods into one frame. This reduces the wasting of bandwidth.

After packing of all frames, a frame-relaxation phase is performed. This attempts to *separate* some messages from frames (by putting them into new/empty frames), in order to make the resulting frame-set more schedulable.

In [66] the FPP is included into the scheduling problem. Besides finding a set of frames, the priority of frames and tasks is also found. In addition the time-slots are found for time-triggered frames and tasks. The FPP is addressed by two algorithms:

1. *SA* – In order to find high-quality frame packing solutions, SA is used. Therein, a neighbour configuration is found by:
    - moving a message from one frame into another frame (or into a new frame)
    - swapping the priority of frames
    - swapping the slot sequence for time-triggered frames
2. *Heuristic* – Messages are sorted due to their offset. Afterwards, adjacent messages are packed into frames. Packing solutions are evaluated in order to determine the schedulability.

In [85] the FPP is included into the TAP. Both problems are formulated as a MILP problem. For the FPP the following constraints are assumed: Messages can only be packed into the same frame, if all of the following is true:

- they are sent from the same processor
- they are sent to the same processor
- they have the same period
- they fit into the frame (message.size $\leq$ frame.payload.left)

The optimization goal is to minimize the end-to-end delay. However, the exact steps in order to pack the messages into frames are not transparent, since they are taken within the MILP solver. The solving routines of the MILP solver are not described.

In order to solve the entire configuration problem, a two-phased approach is used. In phase one, near-optimal task allocation and task priority assignment is searched. Thereby simple frame packing (1 message = 1 frame) is used and frame priorities are set based on the DMPO. The solution is found by using a MILP solver. Based on the best solution of phase one, the configuration is refined within phase two. Thereby task allocation is fixed. Frame packing, task and frame priority assignment is refined. Again, the solution is found by using a MILP solver.

In [87] the approach of [85] is extended towards *extensibility*. The additional question is: How much can the WCET of tasks be increased, until end-to-end delays violate their deadlines. Again, a two-phased approach is used: In phase one an initial task allocation is found, assuming simple frame packing (1 message = 1 frame), and given priorities for tasks and frames. A MILP solver is used. In phase two, frame packing and priority assignment is refined, using heuristics. First, frame packing is refined. Therefore, task allocation is fixed. Messages are grouped based on the processor that sends them and their periods. Messages within a group are sorted according to their priority. The priority is assumed to be given. The grouped and sorted messages are then packed into frames, as long as their size fit into a frame. Later, priority assignment is refined. Therefore, task allocation and frame packing is fixed. Initial priorities are assigned to tasks and frames, based on DMPO. Then, priorities are iteratively changed, due to the criticality. The criticality states, how much a change in task WCET influences the probability of missing the deadline of tasks and frames. Elements with high criticality are assigned high priority.

Besides these approaches that are tailored to the automotive domain, frame packing can also be found in other domains. Although these approaches show some interesting aspects, in general they cannot be applied to the automotive FPP directly.

In [49] frame packing is used to enhance the performance of the Totem protocol. The goal is to reduce overhead data. The idea is as follows: Each message has a message header. However, two instances of the same message have the exact same information inside their headers. By packing several instances of a message into a frame, the information inside the headers would be transmitted several times. Since the information inside the message headers is the same, only one header would be needed. All other headers could be removed. This is exactly the key idea of the proposed approach: By packing several instances of a message into a frame, all message headers (except for one) can be removed, without losing information. Thus, less data needs to be transmitted in total. The implementation of the approach works as follows: Several subsequent instances of a message are buffered. When the buffer reaches a defined level, the buffered messages are packed into a frame. During this packing, all message headers are stripped off, only the message header of the first message instance stays intact. This approach works, since all instances of the message have to be transmitted to the same destination.

In [51] frame packing is used for wireless communication. There, different users are multiplexed in time and frequency. Thus, frames are two-dimensional (time and frequency), and have a rectangle shape. The task is to place the frames into the two-dimensional time/frequency plane. The problem is modeled as a *2D strip packing problem*, and is solved using a genetic algorithm. Thereby, the *next fit decreasing height (NFDH)* heuristic is utilized.

5.4.1 Summary

Table 5.2 summarizes the objectives that are used by state-of-the-art FPP approaches. It can be concluded that it is hard/impossible to compare the approaches, since different objectives are followed.

Approach		**Objective**
Sandstroem et al.	[74]	minimize bandwidth demand of frame-set
Saket et al.	[73]	maximize schedulability of frame-set
Pop et al.	[66]	maximize schedulability of system
Zheng et al.	[85]	minimize end-to-end-delays
Zhu et al.	[87]	maximize extensibility of task-set

Table 5.2: Optimization objectives of state-of-the-art frame packing approaches

Besides these differences, a common issue can be seen. In BPP algorithms, both sorting as well as packing is performed due to the same attribute: item size. In FPP algorithms, different attributes are used. Sorting is done by deadline/period/offset, whereas packing is done due to messages' data size.

5.4.2 Open Issues

By analysing state-of-the-art FPP approaches, the following open issues can be identified:

1. The FPP is solved by heuristics and/or approximation algorithms. Although they may work fine in general, they all are based on a simple decision making. Packing is performed, if the necessary condition is satisfied:

   $$\text{message size} \leq \text{free frame payload} \tag{5.1}$$

   Packing decisions, based on more sophisticated reasoning, are missing.

2. If the FPP is included into the TAP, this is always done by a two-phased approach. Within the first phase, the TAP is solved assuming simple frame packing (1 message = 1 frame). Within the second phase, the best task allocation from phase one is fixed, and the FPP is solved in detail for that task allocation. The FPP is never solved in detail during solving the TAP.

3. The FPP is always solved without taking into account sophisticated constraints, e.g. that two messages must not be packed into the same frame.

4. The FPP is always solved "from scratch". FPP algorithms do not address the scenario, where there already exist some frames, and additional messages must be packed into the existing frame-set.

In order to overcome these open issues, the following steps are taken:

1. In section 5.5 optimality criteria are derived. These criteria represent the trade-offs that have to be taken into account, if optimal frame packing should be achieved. These criteria are then used, in order to perform sophisticated packing decisions, resulting in optimized frame packing results.

2. In section 5.7 the FPP is included into the TAP. Thereby the FPP is solved in detail during the TAP.

3. In chapter 6 domain-specific constraints are introduced. It is then shown how these constraints can be taken into account by FPP algorithms.

4. In chapter 7 *system configuration upgrade* scenarios are introduced. It is then shown how the FPP can be solved for these scenarios.

5.5 Optimized Frame Packing

Besides the differences between state-of-the-art FPP approaches, one common issue can be identified: The packing decision is based on only a single property:

$$\text{message size} \leq \text{free frame payload} \tag{5.2}$$

Although this is a necessary condition, it is not a sufficient one, if bandwidth-optimal frame packing should be achieved. Since the bus bandwidth is a limited (and valuable) resource, frame packing should be bandwidth-optimal.

Simply speaking: Bandwidth-efficient frame packing should cause as few overhead data as possible. Intuitively, this can be achieved by having as few frames as possible. Consequently, as many messages as possible should be packed into each frame. Thus, it seems the ideal situation is to have all frames fully utilized. This means that the max. amount of payload data is used within each frame.

The bandwidth consumption of a frame is:

$$bw_f = \frac{pay_f + oh_f}{T_f} \tag{5.3}$$

Since a frame may have several messages packed-in, the frame must be sent at a rate that satisfies the rate of all packed-in messages. Thus, the message with the smallest period determines the period of the frame.

$$T_f = \min_{m \in f} \{T_m\} \tag{5.4}$$

Packing messages with different periods into the same frame results in the situation that certain messages are sent more frequently than needed. As a consequence, these messages cause additional bandwidth demand.

$$bw_{m\ \text{additional}} = \frac{s_m}{T_m - T_f} \qquad \text{if} \quad T_m > T_f \tag{5.5}$$

Consequently, the ideal situation is to have fully utilized frames and all messages inside a frame having the same (or very similar) periods. In general (since both the data size as well as the period of messages vary) this ideal situation represents a trade-off: If messages with different periods are packed into a frame, the frame must be sent at the shortest period, and thus some of the messages are sent more frequently than needed. This increases the bandwidth consumption. On the other hand, the more messages that are packed into a frame, the fewer frames are needed. Thus less overhead data is sent. This reduces the bandwidth consumption.

Some heuristics for the FPP that can be found in the literature. [73, 74] tackle this trade-off by trying to pack messages with *similar* periods into the same frame. This is done by sorting the messages due to their period, before packing them. However, during packing only the necessary packing condition (see equation (5.2)) is used. Frame and message periods are not taken into account. Because of that, bandwidth-efficient frame packing cannot be guaranteed.

5.5.1 Trade-Off Optimality Criteria

In order to achieve bandwidth-optimal frame packing, the trade-off that has to be faced within the FPP has to be taken into account during each packing step. For this purpose, an optimality criteria is needed. Based on this criteria, near-optimal frame packing configurations can be found.

Assume the following minimal example: There exists a frame that already has some messages packed-in. Another message needs to be packed-in and it can fit into the existing frame. The question is: Should the message be packed into the existing frame (thus extending it), or should the message be packed into a new frame?

The optimal decision can be taken, by analysing the bandwidth demand of the two alternatives (left and right side of equation):

$$\underbrace{\frac{pay_f + s_m + oh_f}{T_f^{'}}}_{\text{extended frame}} = \underbrace{\frac{pay_f + oh_f}{T_f}}_{\text{existing frame}} + \underbrace{\frac{s_m + oh_f}{T_m}}_{\text{new frame}} \tag{5.6}$$

Note that the period of a frame is determined by the message with the smallest period inside the frame. By adding a message, the period of the extended frame $T_f^{'}$ may change. Originally it is T_f.

$$T_f = \min_{i=m\in f} \{T_i\} \tag{5.7}$$

$$T_f^{'} = \min\{T_f \cup T_m\} \tag{5.8}$$

Depending on the ratio between T_m and T_f, there are 3 cases for this packing situation. For each of them, an optimal decision can be made.

Case I: $T_m = T_f$

If the frame's period is equal to the message's period, it is always beneficial to extend an existing frame. Creating a new frame is never beneficial, because it would cause additional overhead.

$$\frac{pay_f + s_m + oh_f}{T} = \frac{pay_f + oh_f}{T} + \frac{s_m + oh_f}{T} \tag{5.9}$$

$$oh_f < 2 \cdot oh_f \tag{5.10}$$

Case II: $T_m > T_f \Rightarrow T_f^{'} = T_f$

The trade-off is: By extending the frame, the message will be sent more frequent than needed, but no additional overhead is created. By creating a new frame, additional overhead is created, but the message will not be sent too frequent.

$$\frac{pay_f + s_m + oh_f}{T_f} = \frac{pay_f + oh_f}{T_f} + \frac{s_m + oh_f}{T_m} \tag{5.11}$$

$$\frac{s_m}{T_f} = \frac{s_m + oh_f}{T_m} \tag{5.12}$$

At the threshold period of the message, the two alternatives perform equally.

$$T_m^{\star} = T_f \frac{s_m + oh_f}{s_m} \tag{5.13}$$

Thus, the optimal decision is:

- $T_m < T_m^\star \Rightarrow$ extending the frame is beneficial
- $T_m > T_m^\star \Rightarrow$ creating a new frame is beneficial

Case III: $T_m < T_f \Rightarrow T_f^{'} = T_m$

The trade-off is: By extending the frame, the frame will need to be sent more frequent, but no additional overhead is created. By creating a new frame, the original frame will not be sent more frequent, but additional overhead is created.

$$\frac{pay_f + s_m + oh_f}{T_m} = \frac{pay_f + oh_f}{T_f} + \frac{s_m + oh_f}{T_m} \tag{5.14}$$

$$\frac{pay_f}{T_m} = \frac{pay_f + oh_f}{T_f} \tag{5.15}$$

At the threshold period of the message, the two alternatives perform equally.

$$T_m^\star = T_f \frac{pay_f}{pay_f + oh_f} \tag{5.16}$$

Thus, the optimal decision is:

- $T_m < T_m^\star \Rightarrow$ creating a new frame is beneficial
- $T_m > T_m^\star \Rightarrow$ extending the frame is beneficial

Example

Table 5.3 shows the threshold period for both cases (II and III) assuming $T_f = 1$, $oh_f = 64$

s_m	$T_m^\star$
1	65
8	9
16	5
24	3.666
32	3
40	2.6
48	2.333
56	2.143
63	2.016

case II: $T_m > T_f$

pay_f	$T_m^\star$
1	0.015
8	0.111
16	0.2
24	0.272
32	0.333
40	0.385
48	0.429
56	0.466
63	0.496

case III: $T_m < T_f$

Table 5.3: Optimal packing decision based on threshold-period of message

Algorithm 5.2: Heuristic for bandwidth-optimal frame packing

```
Input: messages
begin packMessagesIntoFrames
    sort(messages, T, increasing) /* sort messages by T [0..n] */;
    frame = new frame;
    foreach message do
        if $frame.payload.left \geq message.size$ then
            /* take most beneficial decision */;
            benefit = extendOrNew(message, frame);
            if $benefit = extend$ then
                /* extend existing frame */;
                pack(message, frame);
            else if $benefit = new$ then
                /* create new frame */;
                close(frame);
                frame = new frame;
                pack(message, frame);
            end
        else
            close(frame);
            frame = new frame;
            pack(message, frame);
        end
    end
end
Output: frames
```

5.5.2 Heuristic for Optimized Frame Packing

By combining the basic concepts of state-of-the-art frame packing approaches with the optimality criteria for bandwidth-optimal packing decision making, an optimized frame packing heuristic can be derived. Algorithm 5.2 shows the optimized frame packing heuristic. The main structure is based on the *Fixed Frame Size* approach from [74], which mimics the *next fit decreasing* heuristic for the BPP. However, messages are not sorted by their deadline, but by their period. This is inspired by [73].

The core-improvement is to incorporate the optimized packing decision making, which is based on the optimality criteria (derived in section 5.5.1): If a message fits into an existing frame, the message is not automatically packed into this frame – as is done in [74]. Instead, it is only packed into the frame, if this packing step is bandwidth-beneficial. If it is not beneficial, the message is packed into a new/empty frame, although there is still space left in the existing frame. Within the *Extend Or New* method, the most beneficial decision is determined using the optimality criteria, derived in section 5.5.1.

This approach guarantees that each packing step has minimal increase of bandwidth demand. Consequently, since each step is bandwidth-optimal, the final packing configuration has the lowest possible bandwidth demand.

Packing Candidates

Algorithm 5.2 assumes that a set of messages is provided. In order to find these packing candidates, the following necessary conditions need to be satisfied: *Only messages, which are sent from the same processor via the same bus-interface, can be packed into the same frame-set.*

Automotive bus systems (LIN, CAN, FlexRay) are transmitting in a *broadcast* manner. Consequently, each processor connected to a bus receives all frames that are transmitted via that bus. Each processor identifies frames which contain relevant messages, and extract these messages by un-packing the frame. Thus, packing candidates can be found by:

- packing candidates = all out-going messages from a processor via a bus-interface

Receiving and un-packing relevant frames imposes computational effort on the processors. Thus, if relevant messages are spread over high number of frames, this may impose significant computational effort on the receiving processors (especially if they have low processing power). To tackle this issue, packing candidates can be pre-grouped by taking into account the destination processors.

- packing candidates = out-going messages from a processor via a bus-interface, having the same destination processor

This pre-grouping does not invalidate the optimality of the presented algorithm. However, it reduces bandwidth-efficiency. For experimental evaluation, pre-grouping is not applied.

5.6 Experimental Evaluation – Frame Packing

In order to evaluate the optimized frame packing heuristic, a set of experiments need to be performed. These experiments should cover a wide range of potential FPP instances that might occur in real-life industrial use cases. Therefore, a test case generation tool was developed, which is capable of generating synthetic pseudo-random FPP instances.

5.6.1 Test Case Generation Tool – GenFPP

In order to generate test cases for the FPP, the tool/algorithm *generate frame packing problem* (GenFPP) is developed. Based on a set of parameters which specify the desired FPP, GenFPP generates a FPP which satisfies that specification. Algorithm 5.3 shows the test case generation methodology. Input parameters are:

- bus baudrate [baud/s]
- nominal bus utilization, due to payload only [% of bus baudrate]
- number of sending processors
- range of message's size (min., max.) [bits]
- range of message's period (min., max.) [s]

Algorithm 5.3: GenFPP – Test case generation tool for frame packing problems

```
Input: bus baudrate
Input: nominal bus utilization /* payload only; % of bus baudrate */
Input: number of processors
Input: size.min , size.max /* range of message's data size */
Input: T.min , T.max /* range of message's period */
Input: T.dist /* distribution of message's period */
begin GenerateFPP
    /* generate hardware model */;
    processors = { };
    bus = new bus(baudrate);
    for 1 to nr.processors do
        processor = new processor;
        processor.connectTo(bus);
        processors.add(processor) /* add to list */;
    end
    /* generate messages */;
    bwd.target = bus.baudrate * nom.bus.util/100;
    bwd = 0 /* bandwidth demand of messages */;
    messages = { };
    repeat
        message = new message;
        message.size = randomUniform(size.min, size.max);
        if T.dist = uniform then
            message.T = randomUniform(T.min, T.max);
        else if T.dist = log uniform then
            message.T = randomLogUniform(T.min, T.max);
        end
        message.D = message.T;
        bwd += message.size / message.T /* update bandwidth demand */;
        messages.add(message) /* add to list */;
    until bwd ≥ bwd.target;
    /* assign messages to processors */;
    foreach message do
        p = randomUniform(nr.processors);
        processor = processors[p];
        assign(message, processor);
    end
end
Output: hardware model
Output: outMessages per processor
```

- distribution of message's period (uniform or log uniform)

The following should be considered concerning the parameters:

- An uniform distribution of message's periods represents a general test case. By using a *log uniform* distribution, the number of periods between 1 and 10 is the same than between 10 and 100, etc. This more precisely models industrial use cases, where periods typically are assigned dedicated, pre-defined values (e.g. 10, 25, 50, 100, 200, ...).

- The parameter *number of sending processors* represents the number of processors that are connected to a bus system. Thus it can be used to tailor the test case towards different bus sizes. A low number (up to 5) represents a *private* bus that is dedicated to a separated sub-domain (e.g. electric components of a hybrid vehicle powertrain). A high number (10+) represents a *public* bus that is shared by many applications (e.g. body electronics of a car).

5.6.2 Test Cases & Evaluation Metrics

Table 5.4 provides the parameters which are used to generate realistic FPP test cases. The parameters are within ranges that are typical for automotive communication systems.

Parameter	**Value/Range**	**Unit**
bus baudrate	125, 250, 500, 1000	kbaud/s
nominal bus utilization	10, 15, 20	%
number of sending processors	1 .. 20	
message size range	1 to 16	bits
message period range	5 to 1000	ms
message period distribution	uniform, log uniform	
number of test cases per specification	100	

Table 5.4: Test case parameters for frame packing evaluation

Due to the large specification space, the experiments have to be performed fully-automatically. Therefore, a test-framework (TestFPP) is developed. As the input TestFPP takes the specification for FPP evaluation. For each point in the specification space, a FPP test case is generated, using GenFPP. For this test case, the FPP is solved (using different FPP solving algorithms), and results are stored in a file for later post-processing.

As shown in algorithm 5.3, GenFPP uses random numbers. In order to reduce the effects that these random numbers may have on FPP results, not only one FPP test case is generated per specification point, but several test cases are generated. Due to the random numbers, these test cases have a slightly different message-set. By generating several FPP test cases per specification point, statistical data can be extracted. This way more solid results can be gained, and confidence in the results can be improved.

The FPP test cases are solved by applying the *optimized frame packing heuristic.* In addition, the same test cases are also solved using the following methods, in order to have reference results:

- simple frame packing (1 message = 1 frame): This approach is used in most TAP approaches. However it represents the *worst-case frame packing.*
- Sandstroem et al. [74]: This state-of-the-art frame packing approach tries to minimize the bandwidth demand of the resulting frame-set. Thus a fair comparison can be done.

No comparison with other state-of-the-art frame packing approaches can be done, since these approaches target different objectives (e.g. min. end-to-end delays). Thus a comparison would not be fair, and also would show misleading results.

For each FPP test case, the resulting frame-set is analysed. Therefore, the following metrics are used:

- *bandwidth demand* – Frame packing should minimize the bandwidth demand of the frame-set.
- *frame payload* – Frames should be highly utilized, in order to have a good data-to-overhead ratio. Thus the used frame payload should be close to maximum.
- *message period variation* – Messages should not be sent too frequently. Thus messages that are packed in the same frame should have similar periods. Message period variation inside a frame should be low.
- *schedulability of frame-set* – Frame packing should have a positive impact on schedulabilty. Fewer frames should miss their deadline.

5.6.3 Results

Evaluation shows that the *optimized frame packing* heuristic outperforms state-of-the-art approaches. Thus, by applying this novel heuristic, better bus configurations can be achieved. Details of the evaluation follow below.

Bandwidth Demand

The main reason for performing frame packing is to efficiently use the available bus bandwidth. For a given set of messages that need to be transmitted via the bus, the bandwidth demand of the resulting frame-set should be as low as possible.

- The bus baudrate marks the available bandwidth of the bus, and represents a threshold for feasibility. Frame-sets that exceed this threshold clearly are infeasible. However, frame-sets that do not exceed this threshold can still be infeasible, e.g. if deadlines are missed.
- The bandwidth demand, that is induced by the messages only, is:

$$bw_{\text{nom.}} = \sum_{m \in \mathcal{M}} \frac{s_m}{T_m} \tag{5.17}$$

This does not include any overhead. Thus, this can be seen as a lower bound for frame packing. However, this lower bound can never be reached by any frame packing algorithm [1].

- The bandwidth demand that is caused by applying *simple* frame packing is:

$$bw_{\text{simple}} = \sum_{m \in \mathcal{M}} \frac{s_m + oh_f}{T_m} \tag{5.18}$$

  This approach assumes that each message is packed into a separate frame. Thus each frame's payload equals the packed-in message's data size. This frame packing strategy can be seen as the *worst case* and as an upper bound for frame packing. Any *real* frame packing approach will cause (significantly) less bandwidth demand.

- The bandwidth demand that is caused by applying *real* frame packing algorithms is:

$$bw = \sum_{f \in \mathcal{F}} \frac{pay_f + oh_f}{T_f} \tag{5.19}$$

  Since all automotive bus protocols only allow payloads that are multiples of bytes, the payload of each frame is rounded up to the next full byte, before calculating the bandwidth demand.

By applying the *optimized frame packing* heuristic to a wide range of FPP instances, its performance can be evaluated in detail. Table 5.5 summarizes its improved performance (in terms of the frame-set's bandwidth demand) for different scenarios (i.e. number of sending processors, bus baudrate). As the reference point *Sandstroem et al.*'s approach is used. The improvement is calculated by:

$$\text{improvement} = \frac{|x_{\text{ref.}} - x|}{x_{\text{ref.}}} \tag{5.20}$$

Based on results from table 5.5, a set of conclusions and trends can be derived:

- The optimized frame packing approach outperforms state-of-the-art frame packing (in terms of bandwidth demand of the frame-set) in any scenario. It never performs worse.

- For a low number of sending processors, both approaches perform similar. For a higher number of sending processors, the optimized frame packing approach performs significantly better than state-of-the-art frame packing approaches.

- The optimized frame packing approach performs significantly well for lower bus baudrates. For increasing bus baudrates, the optimized frame packing approach still performs better than state-of-the-art approaches, however, the improvements become lower.

[1] Assuming a CAN bus, where the max. frame payload is 8 bytes and the frame overhead is 64 bits. In this case, the best obtainable bandwidth demand of a frame-set is twice the nominal bandwidth demand of the message-set.

	Improvement [%]			
Processors	125 k	250 k	500 k	1000 k
1	0.0	0.0	0.0	0.0
3	6.4	1.6	0.0	0.0
5	9.7	4.0	0.8	0.1
10	14.0	8.9	4.2	1.0
15	17.7	12.4	7.2	2.3
20	18.9	14.5	9.4	3.9

message period distribution: uniform

	Improvement [%]			
Processors	125 k	250 k	500 k	1000 k
1	0.1	0.0	0.0	0.0
3	6.4	0.5	0.0	0.0
5	13.7	4.4	0.2	0.0
10	14.7	13.9	4.6	0.2
15	13.9	14.4	10.3	1.8
20	10.3	14.3	14.2	4.6

message period distribution: log uniform

Assumptions: s_m = 1 to 16 bit, T_m = 5 to 1000 ms

Table 5.5: Improvement in bandwidth demand of optimized frame packing over state-of-the-art frame packing

- The performance of the frame packing approaches is noticeable different for different message period distributions (uniform, log uniform). For an *uniform* distribution the highest improvements can be achieved for low baudrates and high number of sending processors. For a *log uniform* distribution the highest improvements can be achieved for low baudrates and medium number of sending processors.

In order to gain deeper insight how different frame packing approaches perform in terms of bandwidth-efficiency, let's investigate the bandwidth-demand of the resulting frame-sets.

Bandwidth Demand vs. Nominal Bus Utilization: One way to evaluate the bandwidth efficiency of a frame packing strategy is to plot the bandwidth-demand of the resulting frame-set over the nominal bus utilization. For a linear increase in nominal bus utilization, the bandwidth-demand of bandwidth-efficient frame packing algorithms will increase (significantly) lower than for non-bandwidth-efficient frame packing algorithms.

Figure 5.2 shows the bandwidth demand for varying nominal bus utilization. It visualizes the following scenario: 10 processors are connected to a CAN operated at 125 kb/s; the data which is exchanged between the processors shall be increased by factor 1.5 to 2 (e.g. because additional application software was added to the processors, and these soft-

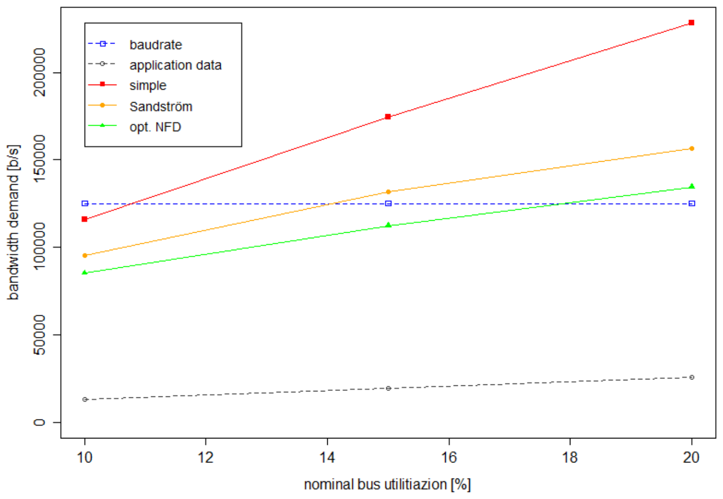


Figure 5.2: Bandwidth demand of frame-sets for different nominal bus utilization. Each line represents a frame-set generated by a different frame packing strategy. Experiment-assumptions: T_m = log uniform, br = 125 kb/s, sending processors = 10

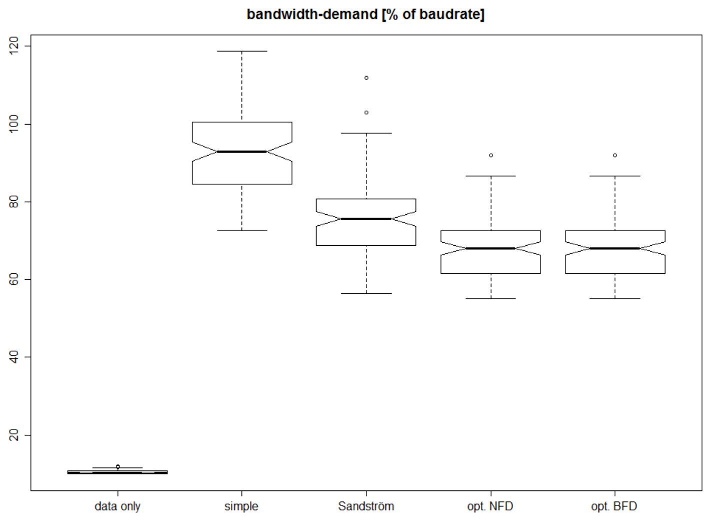


Figure 5.3: Notched box-plot confirming the statistical significance of the improvements from figure 5.2. Each box represents the frame-set generated by a different frame packing strategy. The box-plot shows the case: nominal bus utilization = 10 %

ware need to communicate). The results clearly show that the optimized frame packing approach significantly outperforms simple frame packing, as well as state-of-the-art frame packing approaches.

Note that each point in the plot represents the average over 100 FPP instances. Therefore, the improvements can also be evaluated in terms of statistical significance. An elegant – yet powerful – way to do this is the use of a notched box-plot.

> *"A box-plot (also known as a box-and-whisker diagram) represents numerical data through their five-number summaries: the smallest observation (sample minimum), lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation (sample maximum). It may also indicate which observations, if any, might be considered outliers. A box-plot does not make any assumptions of the underlying statistical distribution. Notched box-plots apply a "notch" or narrowing of the box around the median. Notches are useful in offering a rough guide to significance of difference of medians: if the notches of two boxes do not overlap, this offers evidence of a statistically significant difference between the medians."*
>
> – Massart et al. [47]

Simply speaking: if the notches do not overlap, the above stated improvements are statistically significant. Figure 5.3 shows the notched box-plot of one point. It confirms that the improvements of the *optimal frame packing* algorithm are statistically significant.

Bandwidth Demand vs. Number of Sending Processors: Another important aspect is how the number of sending processors impacts the performance of the FPP algorithm. This will show, if larger-scaled networks can be efficiently configured. In order to evaluate this aspect, the following scenario is addressed: as additional software is added to the system, additional processors are needed to provide the processing power; however, by wise task-allocation decisions, the data which is sent via the bus-system doesn't increase. The hypothesis is, that the number of sending processors however has significant impact on the resulting bandwidth demand, since there is fewer potential to dense packing.

Figure 5.4 shows the bandwidth demand for a varying number of sending processors. Again, each point is an average over 100 FPP instances. A low number of sending processors represent a *private* bus, whereas a high number of sending processors represents a *public* bus. Analysis of the results reveals and re-confirms a set of trends and conclusions:

- Simple frame packing results in very high bandwidth demand. In many cases (however not shown in this example) the frame-set becomes infeasible due to bus overload.
- The optimized frame packing algorithm needs significantly less bandwidth, compared to state-of-the-art frame packing.
- The optimized frame packing performs best, if the number of messages per processor is small. This indicates that it is very efficient in finding near-optimal packing. If the number of messages is higher, this benefit cannot be seen so clear.

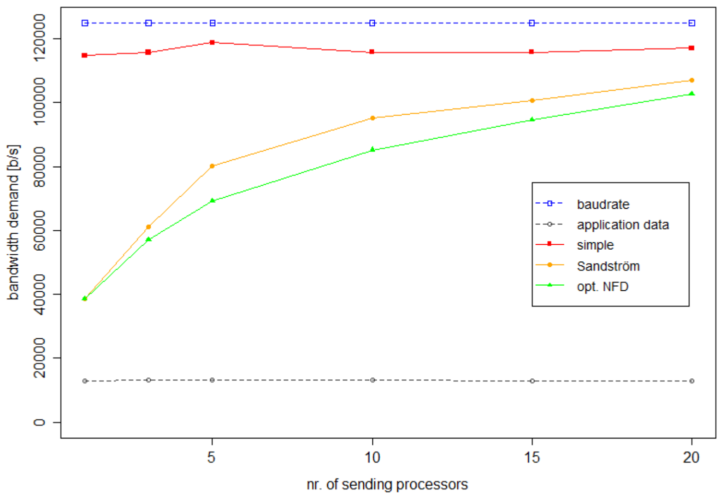


Figure 5.4: Bandwidth demand of frame-sets for different nr. of sending processors. Each line represents a frame-set generated by a different frame packing strategy. Experiment-assumptions: T_m = log uniform, br = 125 kb/s, nominal utilization = 10 %

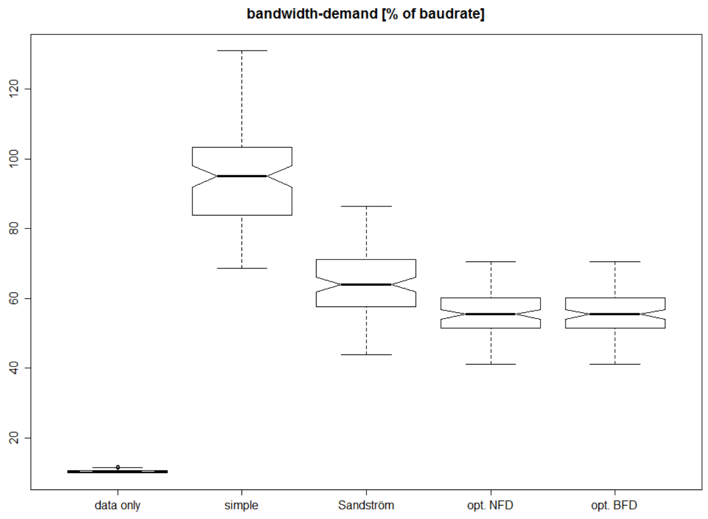


Figure 5.5: Notched box-plot confirming the statistical significance of the improvements from figure 5.4. Each box represents the frame-set generated by a different frame packing strategy. The box-plot shows the case: sending processors = 5

Figure 5.5 shows the notched box-plot or one point. Again, it reveals that the improvements of the optimized frame packing algorithm are statistically significant.

Knowing that the optimized frame packing approach generates frame-sets which consume less bandwidth than those generated by other approaches, it is interesting see how this bandwidth demand saving is composed of. Bandwidth demand saving can only stem from two sources:

- *overhead* – Messages are packed more compact into frames, thus fewer frames are needed, and consequently less overhead is generated. This improves the data-to-overhead ratio.

- *period variation* – The messages that are packed into a frame have very similar periods, thus messages are not sent more frequently than needed.

Less overhead data can only be achieved, if frames are packed dense, thus needing fewer frames. An interesting fact is, however, that the optimized frame packing approach generates more frames than state-of-the-art approaches. On average, 10.1% more frames are generated, and thus more overhead is generated. However, the frames that are generated by the optimized frame packing approach contain messages with more similar periods. As a consequence, messages are not sent too frequently, thus saving bandwidth. This bandwidth-saving over-compensates the increased overhead data.

These results clearly show how the optimized frame packing approach is able to outperform state-of-the-art approaches: The key is not to try to pack the frames as compact as possible (like state-of-the-art approaches try to do), but as *bandwidth-efficient* as possible. The trade-off optimality criteria seem to solve this task extremely well.

Payload Utilization

In order to get more insight into the results, the resulting frame-set is analysed in detail. An important aspect of frame packing is the packing density: How much of the frame's available payload is used.

In industrial scenarios, where frame packing is usually performed manually, the frame payload is utilized very high (6 to 8 of 8 bytes). These industrial frame-sets are used as a reference, for comparing the payload utilization of the optimized frame packing approach. By analysing the frame payload after applying automated frame packing, it can be determined how "close" these approaches can get to industrial scenarios. This will give indications on how well automated frame packing approaches will get accepted within industry.

Figure 5.6 shows the probability distribution of frame payloads. The industrial frame-set uses quite high packing density. About 50% of all frames use 8 bytes payload, 15% use 7 bytes and 20% use 6 bytes. The remaining 25% use 5 or less bytes.

However, the frame-set generated by automated frame packing approaches use even higher packing density. About 70% of the frames use 8 bytes, 20% use 7 bytes, and the remaining 10% use 6 or less bytes. Sandstroem et al. generate slightly more frames with 8 bytes payload, thus needing fewer frames. The optimized frame packing approach generates slightly more frames using 7 bytes.

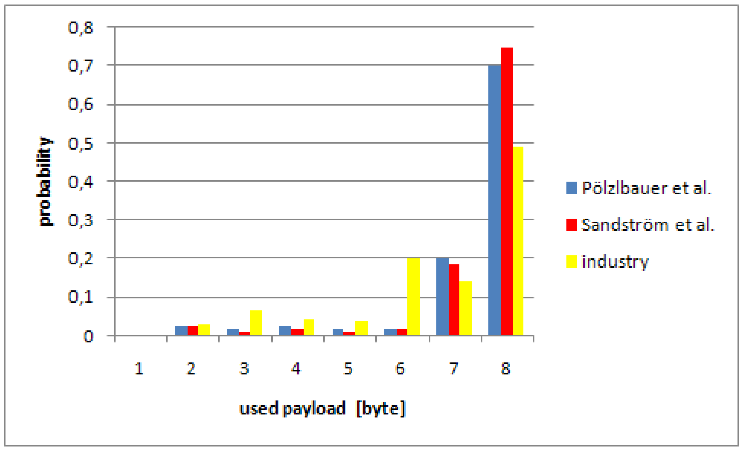


Figure 5.6: Payload utilization for different frame packing strategies

The results show that automated frame packing approaches can achieve a packing density that is even higher than industrial approaches. This gives indication that the automated approaches are efficient, and may get accepted in industrial use cases.

By analysing manually generated industrial frame-sets, additional insight can be gained. It seems, bandwidth consumption is not the only issue that is considered during manual frame packing. Additional aspects are considered.

- In several cases, messages are not only packed due to bandwidth demand, but also due to *logical grouping*. This means to group messages that are associated to similar functions or have a similar semantic (e.g. engine speed and engine torque). This will be re-visited in chapter 6.
- In some cases, frames are not fully utilized in order to leave some free space left for future extensibility. This will be re-visited in chapter 7.

Schedulability

Besides minimizing the bandwidth demand, it is of upmost importance that the frame-set is schedulable. Thus, another interesting aspect of frame packing is its impact on the schedulability of the resulting frame-set.

Since a frame consists of several messages, the frame must satisfy the deadlines of all packed-in messages. Thus the deadline of a frame is [73]:

$$D_f = \min\{D_m - \text{offset}(T_{\min}, T_m)\} \tag{5.21}$$

As deadlines may be arbitrary (do not need to match the period), priority assignment for frames should be performed due to the DMPO policy [4] which is known to be optimal for independent elements (here: frames). In order to evaluate the schedulability of the frame-set[1], WCRT analysis is applied [21]. Results are expressed by the *un-schedulability ratio*, i.e. the number of un-schedulable frames compared to the number of frames.

[1] A more fine-grained approach would be check schedulability of messages instead of frames. This can be achieved, since messages inherit their WCRT from the frame they are packed in. For the sake of simplicity, schedulability is checked at frame-level. However, this does not decrease the expressiveness of the results.

Figure 5.7 shows the un-schedulability ratio for varying nominal bus utilization. It shows that an increased utilization leads to an increased un-schedulability. However, the proposed packing strategy leads to a lower number of deadline-misses.

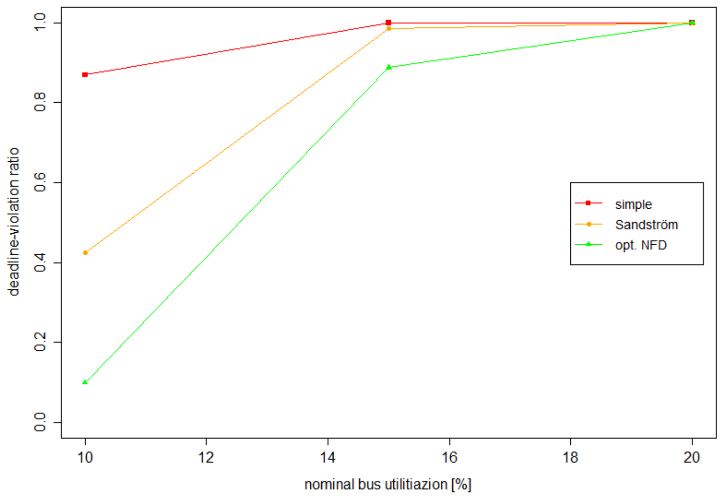


Figure 5.7: Un-schedulability of frame-set for different nominal bus utilizations. Each line represents a frame-set generated by a different frame packing strategy. Experiment-assumptions: T_m = log uniform, br = 125 kb/s, sending processors = 10

Figure 5.8 shows the notched box-plot, confirming that the results are statistically significant.

In addition, the impact of *number of sending processors* on schedulability of the frame-set is investigated. Figure 5.9 shows the results: a higher number of sending processors lead to a higher number of deadline-violations. Again, the proposed optimized frame packing strategy outperforms state-of-the-art approaches.

Figure 5.10 shows the notched box-plot, confirming the statistically significance of the resuts.

Overall, the results can be summarized as follows;

- Simple frame packing generates a high number of frames. Because of that, a high number of frames (especially low priority frames) miss their deadline.

- By using real frame packing approaches, the number of frames is significantly reduced. Based on the lower number of frames, the interference between frames is also significantly reduced. As a consequence, significantly fewer frames miss their deadline.

- The optimized frame packing strategy outperforms all other strategies w.r.t. schedulability in most cases (in some cases they perform equivalent).

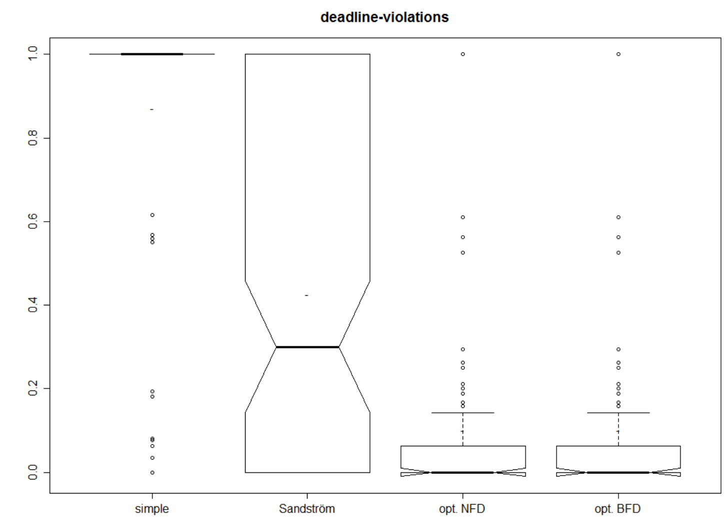


Figure 5.8: Notched box-plot confirming the statistical significance of the improvements from figure 5.7. Each box represents the frame-set generated by a different frame packing strategy. The box-plot shows the case: utilization = 10 %

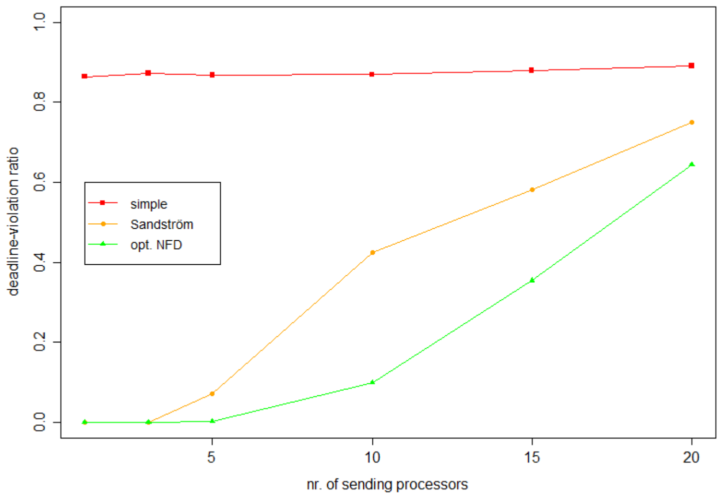


Figure 5.9: Un-schedulability of frame-set for different nr. of sending processors. Each line represents a frame-set generated by a different frame packing strategy. Experiment-assumptions: T_m = log uniform, br = 125 kb/s, nominal bus utilization = 10 %

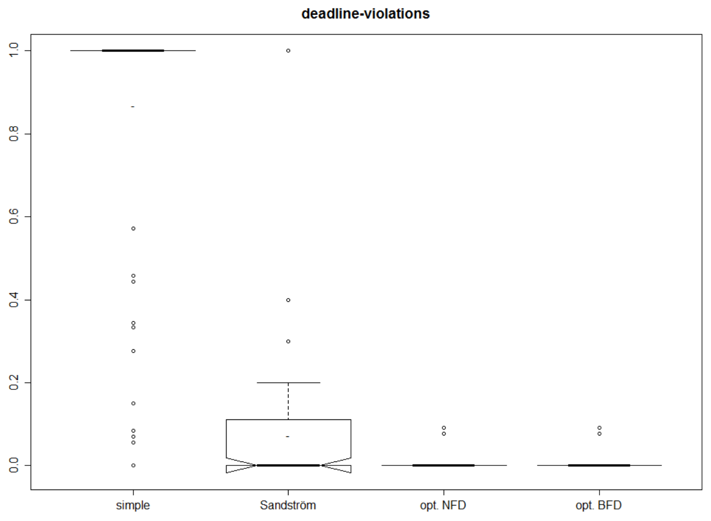


Figure 5.10: Notched box-plot confirming the statistical significance of the improvements from figure 5.9. Each box represents the frame-set generated by a different frame packing strategy. The box-plot shows the case: sending processors = 5

Robustness of Schedulability

Another important aspect of schedulability is its *robustness*. The question this metric tackles is: How much does schedulability of the frame-set change, if there exists uncertainties in parameters? In order to answer this question, *sensitivity analysis* is applied.

Thereby, several different parameters could be varied. However, not all variations make sense, due to the lack of occurrence in real-life systems. Table 5.6 provides an overview of parameters and an estimation how realistic these variations are in real-life systems.

	Parameter	**Probability of Real-Life Variation**
s_m	message data size	does not change during run-time of system
T_m	message period	does not change, however may have some jiter
pay_f	frame payload	does not change during run-time of system
T_f	frame period	does not change, however may have some jitter
C_f	frame transmission time	is subject to variation due to *bit-stuffing* (CAN); may also vary if frame transmission is corrupted and therefor the frame must be re-transmitted, thus increasing C_f
br	bus baudrate	does not change during run-time of system

Table 5.6: Possible parameters for sensitivity analysis of frame scheduling

It is concluded that the most realistic parameter variation is the *frame transmission time* (C_f). Thus, sensitivity analysis is performed due to this parameter in the following manner:

- If the frame-set is schedulable, C_f is up-scaled until the frame-set becomes un-schedulable (i.e. a scaling-factor larger than 1.0). The more C_f can be up-scaled, the more robust is the frame-set.
- If the frame-set is un-schedulable, C_f is down-scaled until the frame-set becomes schedulable (i.e. a scaling-factor smaller than 1.0). The more C_f needs to be down-scaled, the more "un-schedulable" is the frame-set.

$$C_f^{'} = C_f \cdot \text{scale C} \quad \forall f \in \mathcal{F} \tag{5.22}$$

Figure 5.11 shows the sensitivity-analysis results for a varying nominal utilization. In this particulat plot, the frame-sets are un-schedulable, thus C_f needs to be down-scaled. The results show, that the frame-set generated by the optimized frame packing approach need the fewest down-scaling. This indicates the increased robustness.

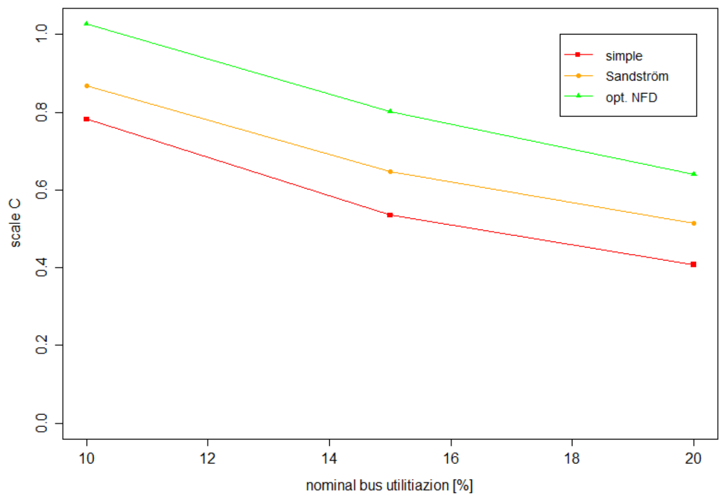


Figure 5.11: Sensitivity analysis: Impact of transmission time on schedulability for different nominal bus utilizations. Each line represents a frame-set generated by a different frame packing strategy. Experiment-assumptions: T_m = log uniform, br = 125 kb/s, sending processors = 10

Figure 5.12 show the notched box-plot, which confirms the statistically significance.

Figure 5.13 shows the results for a varying number of sending processors. It reveals that the number of sending processors has a significant influence on the robustness of the frame-set. Best robustness is found for a small number of sending processors. It also shows clearly that the optimized frame packing strategy outperforms state-of-the-art ones.

Figure 5.14 shows the notched box-plot, confirming the statistical significance.

Note: Although *bus baudrate* does not change during run-time in real-world scenarios, sensitivity analysis can be performed due to this parameter as well. From these experiments interesting results can be derived. They can be used in order to answer a design-related question: Which (min.) bus baudrate is necessary in order to achieve real-time communication for a given frame-set? This approach helps to find the best-fitting

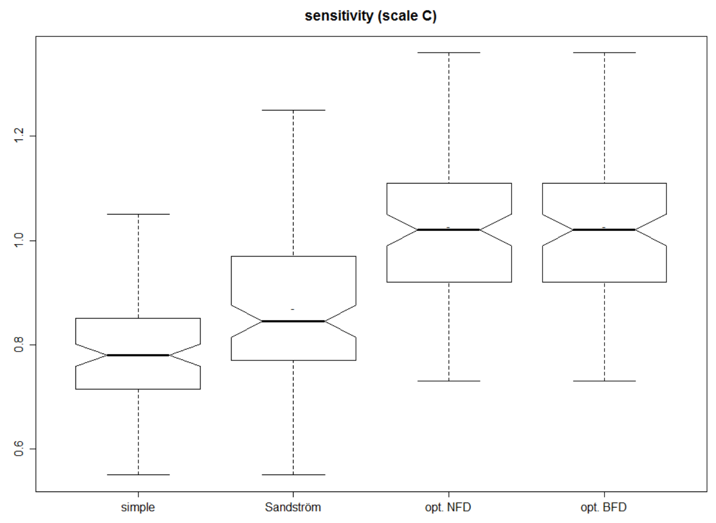


Figure 5.12: Notched box-plot confirming the statistical significance of the improvements from figure 5.11. Each box represents the frame-set generated by a different frame packing strategy. The box-plot shows the case: nominal bus utilization = 10 %

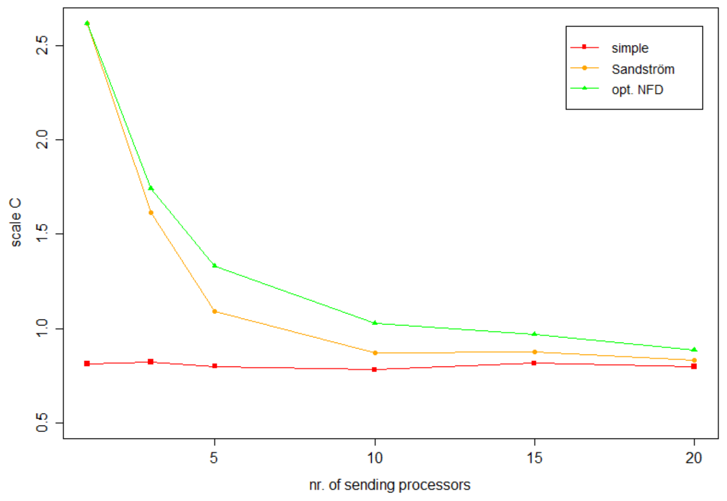


Figure 5.13: Sensitivity analysis: Impact of transmission time on schedulability for different nr. of sending processors. Each line represents a frame-set generated by a different frame packing strategy. Experiment-assumptions: T_m = log uniform, br = 125 kb/s, nominal bus utilization = 10 %

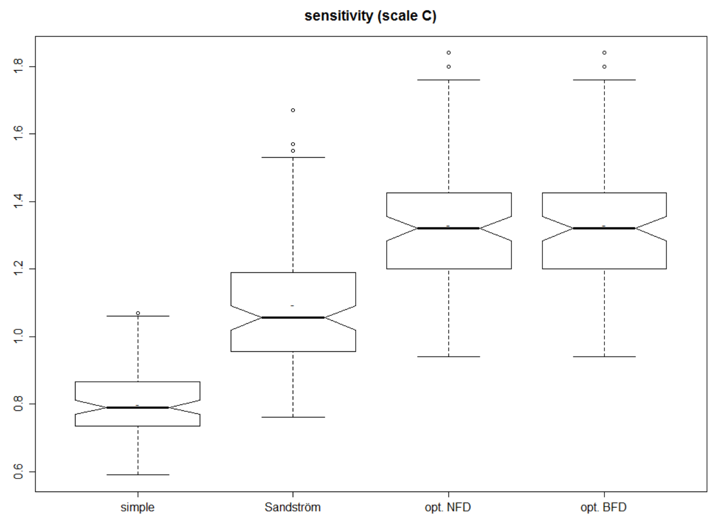


Figure 5.14: Notched box-plot confirming the statistical significance of the improvements from figure 5.13. Each box represents the frame-set generated by a different frame packing strategy. The box-plot shows the case: sending processors = 5

bus configuration for a given communication demand. This way, cost-efficient solutions can be developed.

5.6.4 Conclusions

The optimized frame packing heuristic, which mimics the *next fit decreasing (NFD)* heuristic for the BPP, outperforms state-of-the-art frame packing approaches due to several aspects. Since there are other heuristics for the BPP, the question arises, if another BPP heuristic would form a better starting point for developing a heuristic for the FPP. In order to answer this question, two more FPP heuristics were developed and evaluated: one mimics the *first fit decreasing (FFD)* and the other one mimics the *best fit decreasing (BFD)* heuristic for the BPP. For both cases, the trade-off optimality criteria for the FPP were also included. The same experiments as described in the previous section have been performed, in order to evaluate these two heuristics. These experiments show the following:

- The FFD approach performs similar to Sandstroem et al. In some cases FFD performs even worse.
- The BFD approach performs (more or less) exactly like the NFD. The differences between NFD and BFD are so minor that they cannot be treated as statistically significant.

Based on these results, it can be concluded that the structure of the heuristic (NFD, BFD) is not the most important factor. No statistically significant difference between NFD and BFD can be discovered. The results also give strong evidence that the determining factor is the usage of the optimized packing decision making (based on the trade-off optimality criteria).

At this point, the use of NFD or BFD is equivalent, since both perform the same. However, there is one significant difference in the way they work: NFD has only one *active* frame at the same time. Once a message is not packed into that frame, the frame is *closed*, never to be used again. BFD however can have several *active* frames at the same time, and frames are never *closed*. At this point, this difference is minor, and does not affect the FPP solving algorithm performance much. However, if frame packing has to be performed on a message-set that needs to satisfy *packing constraints* (e.g. that two messages are not allowed to be packed into the same frame), this difference becomes major. Thus, this issue will be re-visited in chapter 6, in which *design constraints* for the system configuration problem are introduced. For now, NFD will be used.

5.7 Including Frame Packing into System Configuration

In the previous sections it was shown how *frame packing* can be solved efficiently, and in a bandwidth-optimal manner. Now the question arises, how the FPP can be included into the TAP. This is essential for designing realistic system configurations. In the literature, there are only a few works [85, 87] that combine the FPP with the TAP. Beside some minor differences, they all use the following two-phased approach:

- Within *phase 1* the TAP is solved, assuming *simple* frame packing.
- Within *phase 2* the best allocation of phase 1 is fixed, and the FPP is solved in detail for that allocation.

Although this approach seems to work, it has a major disadvantage: During phase 1, a high number of allocations are deemed to be infeasible, either because the bus is overloaded or because frames miss their deadline. Usually these allocations are rejected / punished by the TAP solving algorithm. However, these allocations are only deemed to be infeasible, because simple frame packing (which is known to be overly pessimistic) is used. If *real* frame packing would be used, most likely the bus would not be overloaded and the frame-set would be schedulable (as shown in section 5.6.3). Thus the allocation would actually be feasible.

By *wrongly* deeming a configuration to be *infeasible*, a high number of task allocations are rejected, although they would be quite good solutions. This pessimism reduces the probability of finding the *best system configuration.* The only way to overcome this issue is to incorporate a *real* frame packing approach into the TAP DSE. By doing so, more feasible allocations can be generated and evaluated, thus leading to a higher design space coverage. Consequently the probability of finding near-optimal system configurations is increased.

5.7.1 Complexity of Frame Packing Solving

Any change of task allocation (i.e. re-allocating a task onto another processor) causes a change in the communication-demand between processors. Thus, the configuration of the communication-infrastructure needs to be re-designed as well. Therefore, frame packing and frame scheduling need to be performed for every TAP-iteration. Since a high number of TAP-iterations are performed during DSE and optimization, the complexity of the FPP solving methodology is a significant factor.

In [66] the FPP is solved using SA. However, solving the FPP takes up to 6 hours. In [85,87] the FPP is solved using MILP. Solving the FPP takes 5.5 hours. These approaches do not scale if the FPP needs to be solved for every TAP-iteration. Therefore, only low-complexity approaches (e.g. heuristics) can be used here. The FPP solving approach presented in section 5.5 is a low-complexity solving approach, which finds bandwidth-optimal frame packing solutions. Therefore, this approach fits well to be integrated into the TAP solving approach.

5.7.2 Cost-Function for Frame Packing Performance

Frame packing is dependent on message routing, which again is highly dependent on task allocation. Therefore task allocation has a significant impact on frame packing. In return, frame packing has significant impact on system performance (e.g. bus utilization and system schedulability).

If the FPP solving performance is not evaluated then a significant contributor of system performance is not taken into account. This would make it hard for the search framework to find good regions in the design space. Therefore, FPP solving performance must be evaluated.

Frame packing mainly influences bus utilization and frame schedulability. Indirectly it also influences system schedulability (since frame WCRT are inherited by the receiving tasks as release jitter). Bus utilization and system schedulability are already considered in the cost-function (see section 4.2.4). However, these terms can also be influenced by other sources, e.g. task allocation. For a given task allocation and message routing, different frame packing solutions can be generated. These frame packing solutions will have different performance. However, these differences cannot be captured by the already used cost-terms. Therefore, an additional cost-term is needed to capture the performance of frame packing.

Since frame packing aims at minimizing the bandwidth demand, the *bandwidth demand of the frame-set* is a suitable metric for FPP solving performance. As a reference, the bandwidth demand of *simple frame packing* is used.

$$cost_{\text{FPP}} = \frac{\sum\limits_{f \in \mathcal{F}} \frac{pay_f + oh_f}{T_f}}{\sum\limits_{m \in \mathcal{M}_\beta} \frac{s_m + oh_f}{T_m}} \tag{5.23}$$

$\mathcal{M}_\beta$ is the set of messages that are sent via a bus system. Processor-internal messages are not considered, since they do not induce bus bandwidth demand. This cost-term is added to the cost-sub-function, which guides the search. Cost-terms are grouped by using a weighted sum.

5.7.3 Experimental Evaluation

The main reason for performing frame packing is to reduce bandwidth demand, and to increase the schedulability of communication. Obviously, the positive effects of frame packing can be seen most clearly for systems with high communication demand. Hence, mainly such systems will be used for evaluating the proposed approach of *incorporating frame packing into system configuration* DSE.

Approaches & Metrics

The evaluation focuses on 3 approaches:

1. *traditional* – The TAP is solved, assuming simple frame packing.
2. *two-phased* – First the *traditional* TAP is solved. Afterwards the best task allocation is fixed and refined, using *real* frame packing.
3. *integrated* – The FPP is solved in detail during TAP DSE.

The *traditional* approach is used by almost all works in the literature. The *two-phased* approach is proposed in [85,87]. The post-processing phase, during which *real* frame packing is performed, will improve bus utilization and communication delays. By comparing the *traditional* to the *two-phased* approach, these improvements should be become clearly visible.

However, the author believes even more improved results can be achieved if *real* frame packing is performed during solving the TAP. This is why the *integrated* approach is proposed. By comparing the *integrated* approach to the other approaches, these improvements should become clearly visible.

The performance of the different approaches is measured by several attributes:

- The *cost* of the *best solution* that is found by the approach represents the overall quality of the obtained solution.
- In addition, the individual cost-terms of the best solution are analysed. This will give more detailed insight, which aspects of the solution is improved (e.g. bus utilization).
- Besides the final values of the cost-terms, it may also be interesting to analyse how the cost-terms evolve over search-time. This will give insight how the search is guided through the search-space.

Selected Examples from the Literature

In [85, 87], where the two-phased approach is proposed, 2 examples are used for evaluation. These examples are good candidates for the evaluation of the integrated approach. Although several attributes of the examples are presented in the papers, some necessary attributes are missing. Therefore, in order to reproduce the examples, the missing attributes have to be assumed / estimated. These assumptions are marked as * within the test case specifications.

Example 1 [85]

Table 5.7 gives the specification of example 1. The assumptions / estimates are as follows: The processor utilization can be estimated, since it is stated that the max. allowed utilization is set to 70%. The nominal bus utilization is estimated by taking into account the results gained in section 5.6.3. The estimated periods represent a range which is commonly used in the automotive domain. The number of transactions is estimated due to the stated number of end-to-end delay deadlines. Transaction length is set to shape task-graphs which are common in the automotive domain.

Parameter	**Value/Range**	**Unit**	
number of processors	9		
processor utilization	60 to 70	%	*
number of bus systems	1		
bus baudrate	500	kbaud/s	
nominal bus utilization	10	%	*
number of tasks	41		
number of messages	83		
message size	1 to 64	bits	
periods	10 to 1000 a 10	ms	*
number of transactions	7		*
transaction length	60 to 85	%	*

Table 5.7: Specification of example 1 [85]

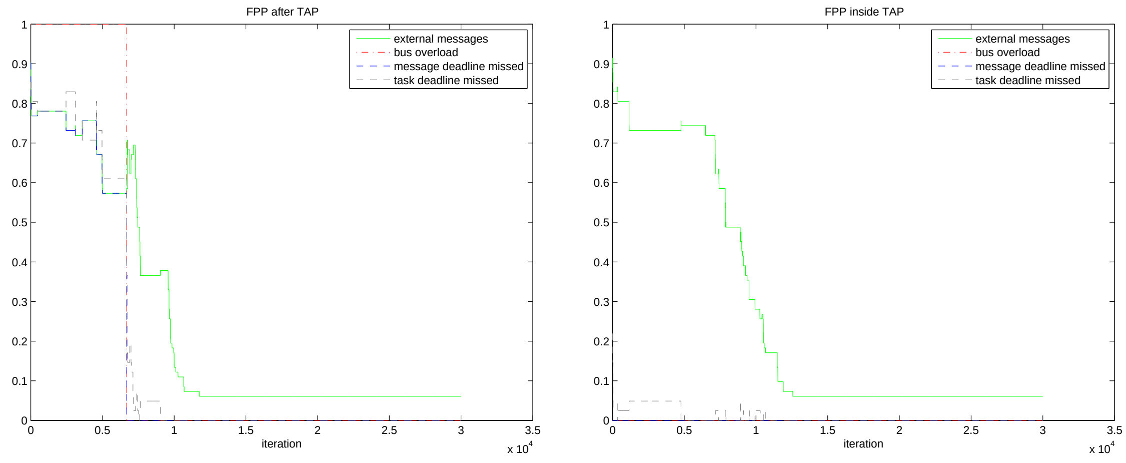
This example system is solved, using the 3 different solving approaches. Table 5.8 provides an overview of the most important attributes of the best obtained solutions.

Metric	**traditional**	**two-phased**	**integrated**	
processors used	7 of 9	7 of 9	7 of 9	
processor utilization	56.3 to 98.9	56.3 to 98.9	56.3 to 98.9	%
external messages	5	5	5	
frames	5	2	2	
bus utilization	2.12	0.94	0.94	%
deadline-violations	0	0	0	
moves to schedulable system	7527	7527	361	

Table 5.8: Best obtained solution for example 1 [85]

The most interesting finding for *example 1* is that all approaches find the same configuration. 7 of 9 processors are used. The same 5 messages are sent via the bus. This gives strong evidence that the best possible allocation was found. Looking at the final solution does not show much benefits for any of the approaches, except that the *traditional* approach results in higher bus utilization due to more overhead data. However, a major finding is that the *integrated* approach converges to feasible (i.e. schedulable) configurations much faster than the other approaches.

Figure 5.15 gives insight into the search performance of the different approaches, by analysing how the cost-terms evolve over search-time: Using simple frame packing (traditional, two-phased) causes bus-overload for the first 25% of the search-iterations. This



(a) two-phased approach (b) integrated approach

Figure 5.15: Search performance for example 1 [85]

again causes message deadline-violations and indirectly task deadline-violations (due to increased release jitter). By re-allocating tasks the search-framework steadily reduces the number of messages that have to be sent via the bus. As soon as the number of cross-processor messages reaches about 60%, the bus is no longer overloaded, and the number of deadline-violations is significantly reduced. This clearly supports the issues raised concerning applying simple frame packing.

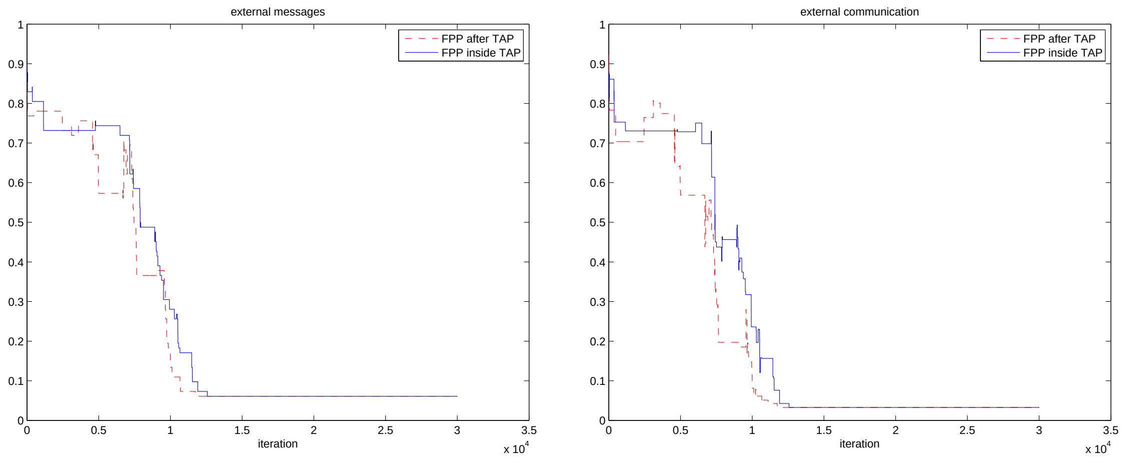
In contrast, by applying real frame packing during TAP, the bus is never overloaded. As a consequence, no message deadline-violations could be observed. Also the number of task deadline-violations is significantly lower. These effects can also be seen in another metric: *search-iterations until first schedulable solution* is found. Here, the integrated approach is about 20 times faster than the traditional / two-phased approach. This shows that the integrated approach's agility.

Another interesting (and not expected) aspect that can be observed is that the number of cross-processor messages is reduced slightly faster if simple frame packing is used (see figure 5.16). This can be explained as follows: Due to the high punishment stemming from bus-overloads and deadline-violations, the search-framework is more eager to find configurations that have a low number of cross-processor messages. However, this benefit does not outweigh the drawback concerning deadline-violations.

Example 2 [87]

Table 5.9 gives the specification of example 2. The following assumptions / estimates are used: The estimated bus baudrate is a commonly used one. Also, the same baudrate is specified in example 1. Message size is estimated due to the specification used in example 1. The number of transactions is again estimated due to the given number of end-to-end delay deadlines. Transaction length is set accordingly.

Again, the example system is solved, using the 3 different solving approaches. Table 5.10 provides an overview of the resulting best obtained solutions.



(a) number of external messages (b) communication-intensity of external messages

Figure 5.16: Reduction of cross-processor communication, example 1 [85]

Parameter	**Value/Range**	**Unit**	
number of processors	7		
processor utilization	30	%	
number of bus systems	1		
bus baudrate	500	kbaud/s	*
nominal bus utilization	10	%	*
number of tasks	43		
number of messages	68		
message size	1 to 64	bits	*
periods	10 to 1000	ms	
number of transactions	6		*
transaction length	35 to 65	%	*

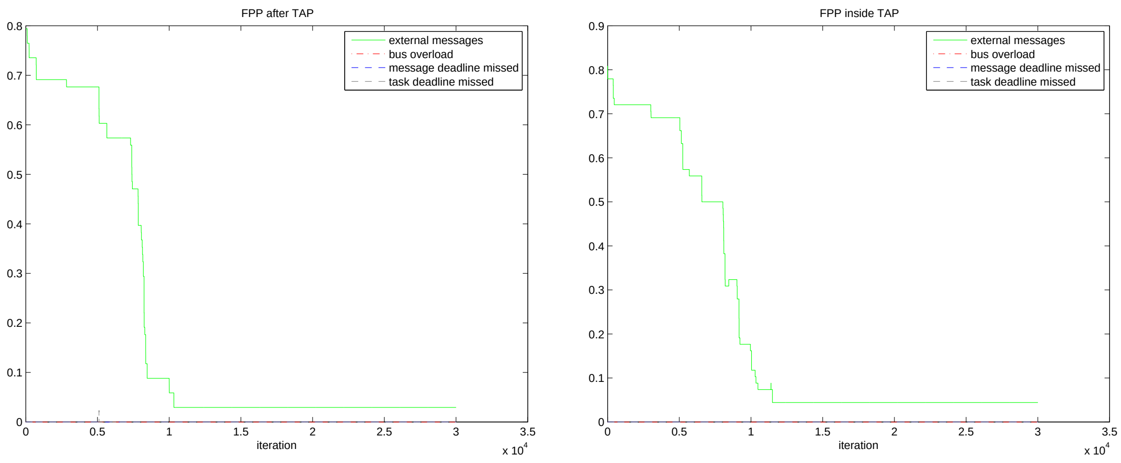
Table 5.9: Specification of example 2 [87]

From *example 2* more diverse findings can be gained. The integrated approach significantly outperforms state-of-the-art approaches (traditional, two-phased) in terms of the best obtained solution. Most important is that the integrated approach needs fewer processors than the other approaches. In addition, although there are more cross-processor messages, significantly less bus bandwidth is consumed.

Figure 5.17 gives insight into the search performance of the different approaches, by analysing how the cost-terms evolve over search-time: For all approaches the bus is never overloaded. Also, no deadline-violations occur for messages. Using real frame packing, also no deadline-violations occur for tasks. In contrast, by using simple frame packing (traditional, two-phased), a few deadline-violations occur for tasks during the first 15% of the search-iterations. These stem from increased release jitter, induced by the increased message WCRT.

Metric	**traditional**	**two-phased**	**integrated**	
processors used	5 of 7	5 of 7	**4** of 7	
processor utilization	12.5 to 89.5	12.5 to 89.5	32.5 to 77.5	%
external messages	2	2	3	
frames	2	2	1	
bus utilization	7.2	7.2	**0.72**	%
deadline-violations	0	0	0	

Table 5.10: Results for example 2 [87]



(a) two-phased approach (b) integrated approach

Figure 5.17: Search performance for example 2 [87]

First Conclusions

Based on the evaluation results, a set of conclusions can be drawn. By integrating *real frame packing* into TAP DSE, the following benefits can be gained:

- The search converges towards feasible (i.e. schedulable) configurations much more rapidly.
- Lower bus utilization can be achieved.
- Processors can be utilized more efficiently, thus resulting in fewer needed processors.

5.7.4 Conclusion & Future Work

The configuration of the communication infrastructure is an essential part of system configuration. Frame packing is a key aspect. By incorporating *frame packing* into task allocation DSE, a set of benefits can be gained: better system configurations (e.g. lower bus utilization, increased schedulability) and better search-performance (e.g. finding a feasible solution within less time).

Cost-Function

By analysing the work of [85, 87], the following conclusion can be drawn: The individual sub-problems (task allocation, frame packing) do not have a constant importance during the DSE.

- At the beginning, the TAP is of high importance, whereas the FPP is of low importance (indicated by using simple frame packing within phase 1).
- At the end, the TAP is of low importance (indicated by fixing one allocation within phase 2), and the FPP is of high importance (indicated by solving the FPP in detail).

The authors claim that this approach works fine. Therefore, the approach could be mimiced. The importance of FPP performance could be dynamically changed during the TAP DSE. The FPP-importance should increase towards the end of the search. This can be achieved by adding an additional factor.

$$(scale_{\mathrm{FPP}} \cdot w_{\mathrm{FPP}}) \cdot cost_{\mathrm{FPP}} \tag{5.24}$$

The factor $scale_{\mathrm{FPP}}$ adjusts the importance of the FPP performance. A scaling of 0 means that the FPP performance is not important at all. Any higher scaling increases the importance. The question arises: Which scaling should be used? To mimic the two-phased approach, scaling would need to be varied from 0 to 1 by an unit step function. However, it is probably wiser to use a smoother transition function:

- The FPP performance shall always have a significant importance during the TAP DSE.
- The importance of the FPP performance may be lower at the beginning. This enables for a wider TAP design space to be explored.
- The importance of the FPP performance shall increase towards the end of the TAP DSE. This will guide the SA towards regions of the design space where high quality frame packing can be achieved.

These considerations can be put into a transition function, as indicated by figure 5.18. The question, to which degree a dynamic cost-function can improve the overall system configuration performance, is beyond the scope of this thesis, and thus subject to future research.

Frame Packing Post-Processing

Evaluation shows that the FPP solving heuristic, which is used during TAP DSE, is able to synthesis realistic low-bandwidth bus configurations. However, the question arises, if an even better bus configuration could be achieved by applying meta-heuristics.

Since meta-heuristics are slow, they can only be applied to the best system configuration, which is available after the TAP DSE. Thus, similar to [85, 87] a post-processing phase could be added, during which the frame packing of the best task allocation is refined, using SA. Therefore, one of the following neighbour moves could be applied (inspired by the neighbour moves from [66] and the refinement steps in [73]):

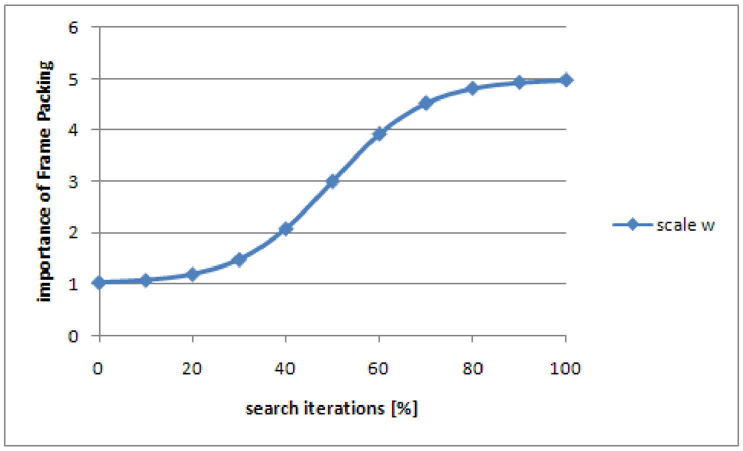


Figure 5.18: Adapting importance of frame packing performance evaluation during task allocation design space exploration

- *swap message* – Move a message from the frame it is currently packed-in, into another frame
- *separate message* – Move a message from the frame it is currently packed-in, into an empty frame
- *split frame* – Split a frame into two frames
- *merge frames* – Move all messages of a frame into another frame

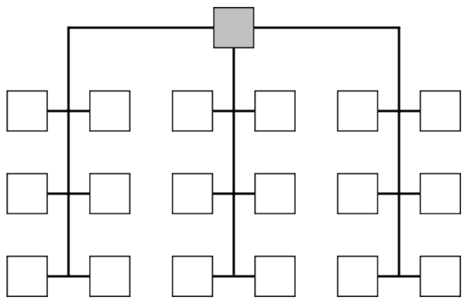
However, the impact of this post-processing optimization onto the system performance is beyond the scope of this thesis, and is subject to future research.

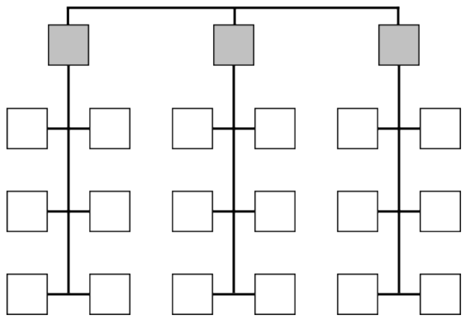
5.8 Gateway Configuration

Sophisticated applications (such as automotive applications) are based on a complex communication infrastructure (see figure 5.19). It involves several different bus systems, which are interconnected via special connecting-processors (often referred to as *gateways*). To enable communication between all processors, data may need to be routed via these gateways. Thereby, data-routing can be achieved at different layers. Table 5.11 gives an overview, tailored to the automotive domain.

In this work, the physical layer is not addressed, only higher layers are of interest:

- *frame repeating* – A frame which is received by the connecting-processor is sent out to all connected bus systems. This is only possible if all bus systems use the same protocol (e.g. CAN). One main disadvantage of this approach is that the frame is also sent to bus systems which do not need the frame.
- *frame forwarding* – A frame which is received by the connecting-processor is sent out to all connected bus systems which need the frame. This way, no non-needed frames are sent. Again, this approach is only possible if all bus systems use the same protocol.


(a) central gateway: all bus systems are connected to a single gateway


(b) backbone gateway: each bus system is connected to a gateway, and all gateways are inter-connected via a backbone bus system

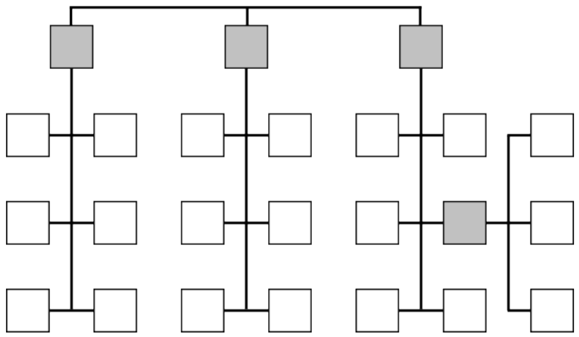

(c) multi gateway: bus systems are inter-connected in a hierarchical manner

Figure 5.19: Gateway topologies

- *message routing* – A frame which is received by the connecting-processor is unpacked, the packed-in messages are routed to their destination bus-interface, and the messages are re-packed into new frames. Messages which are not needed at the destination bus are removed, in order to reduce the bus utilization at the destination bus. This approach can be applied if the bus systems use the same protocol, and must be applied if the bus systems use different protocols. This is because different protocols have different frame structures (e.g. max. frame payload: 8 bytes for CAN vs. 254 bytes for FlexRay).

- *gateway* – In addition to message routing, the connecting-processor may also run application tasks. Messages which are generated by these tasks may be added to the outgoing frames, during re-packing.

Of course, the frames which are output by the connecting-processor must be scheduled on the associated bus system according to its scheduling policy. Therefore, scheduling attributes (such as priorities or time-slots) need to be assigned accordingly.

In this work, it is assumed that a connecting-processor may host application tasks. Hence such processors will be called *gateways*. The gateway may connect different bus systems, thus it will perform un- and re-packing of frames. This assumption is in alignment with the concept of *signal routing* of the AUTOSAR standard.

OSI layer		Connector	Description (Automotive)
OSI 6	presentation	gateway	Additional messages that are generated inside the gateway-processor may be added to the outgoing frames.
OSI 3	network	router	Frames are un-packed, and the included messages are re-packed into new frames. These are sent via the dedicated destination bus system. (e.g. CAN-to-FlexRay)
OSI 2	data link	bridge, switch	The incoming frame is retransmitted via the connected bus systems. This assumes both bus systems use the same protocol. (e.g. CAN-to-CAN)
OSI 1	physical	repeater, hub	The electrical waveform is repeated, to increase signal integrity. The incoming signal is made available to all connected bus systems. Thus it assumes that all connected bus systems use the same protocol. (e.g. FlexRay active star)

Table 5.11: Connecting automotive bus systems at different layers

The work which is performed by the gateway (i.e. receiving of frames, un-packing of frames, message routing, re-packing of frames) is performed by dedicated tasks. These consume processor time. If all these tasks are modeled in detail, it is called *white-box* modeling. In this work the gateway-internals are not modeled in detail, thus called *black-box* modeling. The internals are just considered in terms of adding a delay, which can be considered for end-to-end response time calculation.

Challenges: If the communication infrastructure consists of several inter-connected bus systems, the following aspects need to be considered in the context of TAP.

- *task allocation* – Which tasks may be allocated onto the gateway-processors?
- *message routing* – What are feasible (and "good") routes between two processors which are connected to different bus systems?
- *frame packing* – In case of un- and re-packing, what is the optimal re-packing of frames?
- *frame scheduling* – How shall the outgoing frames of the gateway be scheduled?

5.8.1 Literature Review Refinement – Configuration for Multi-Bus Systems

In [65] the scheduling problem (for communicating tasks) is solved for multi-cluster systems. The hardware consists of several processors connected to two bus systems (i.e. TDMA and CAN). The bus systems are connected via a gateway. The goal is to find a schedule that meets all deadlines, which minimizes the required buffer size on the gateway. The approach builds a TDMA-schedule, and then assigns offsets to the time-triggered elements and priorities to event-driven elements. These steps are performed by problem-specific heuristics. The following assumptions are made: All processors (including the gateway) perform simple frame packing (i.e. 1 application message = 1 bus frame). The gateway just performs frame forwarding. Task allocation is assumed to be given and is not modified.

In [67] the approach of [65] is extended by the TAP. Task allocation is no longer a given value, but is determined by the algorithm. The approach tackles the problem in a 3-phased manner: In phase 1, an initial bus-access configuration as well as an initial task allocation is built. For this configuration, a schedule is found. If the initial configuration is unschedulable, phase 2 is performed. Herein, task allocation and scheduling is improved. If phase 2 doesn't manage to find a schedulable configuration, phase 3 is performed. Herein, the bus-access configuration is optimized. However, the entire approach does never tackle the FPP, since simple frame packing is used (i.e. 1 application message = 1 bus frame).

In [66] the approach of [65] is extended by the FPP. Besides finding a schedule, the issue of packing application messages into bus frames is addressed as well. A heuristic is proposed, which packs messages according to offsets. However, this approach does not address the TAP. Task allocation is assumed to be given, and is not modified.

Open Issues

Configuring a multi-bus system is addressed from different perspectives in the literature. However, no work can be found which addresses all 3 aspects (i.e. task allocation, frame packing, scheduling) at the same time. Always, some of these aspects are assumed to be given. Thus, the question is: how can TAP, FPP, and scheduling for a multi-bus system be solved in an integrated approach?

5.8.2 Including Gateway Configuration into System Configuration

In order to build a system configuration for a multi-bus system, several methodologies (which have already been presented in this work) need to be combined:

- *task allocation* – As the gateway-processor is capable of hosting application tasks, it can be treated as any other processor. The same task allocation methodology as described in section 4.2.3 can be applied. Also, the same optimization objectives as described in section 4.2.4 can be applied.

- *scheduling* – Application tasks which are allocated on the gateway-processor are scheduled according to fixed-priority preemptive scheduling. Thus, state-of-the-art priority-assignment policies (such as DMPO) can be applied.

- *routing* – Finding a feasible route for a message between any processors can be done by applying *depth-first search.* Results of this design-time analysis are stored in the *routing table* inside the gateway-processor.

- *frame packing* – In alignment with AUTOSAR's *signal-gateway*, a gateway-processor is un-packing incoming frames, and is re-packing the messages into new frames at the destination bus-interface. Packing candidates are the routed messages as well as the messages which are generated inside the gateway-processor. Packing can be performed due to the same strategy as described in section 5.5. As the packing heuristic has low complexity, it can be included into the TAP DSE without compromising search-performance (as the performance-bottleneck is the schedulability-test).

5.9 Conclusions

Starting from a state-of-the-art approach (that did not incorporate some essential sub-problems), the system configuration problem was extended by a detailed configuration of the communication infrastructure.

By including this sub-problem into the overall DSE, realistic system configurations can be generated. Based on such system configurations, properties and behaviour of the system can be estimated more precisely/realistically.

Chapter 6

Satisfying Design Constraints

In chapter 5, the *system configuration problem* is solved in a more realistic manner than state-of-the-art approaches do. The key improvement over state-of-the-art is incorporating a more realistic communication-model which especially tackles *frame packing*. However, only a small set of constraints is taken into account so far:

- *task allocation* – The utilization of a resource (processor, bus-system) must not exceed 100% (or a specified threshold, e.g. 95%).
- *frame packing* – Messages can only be packed into the same frame if their accumulated data sizes do not exceed the frame's max. payload, and if the messages originate from the same processor's bus-interface.
- *scheduling* – The WCRT of a schedulable object (task, message/frame) must not exceed its deadline.

These constraints mainly stem from schedulability considerations. Thus, these constraints are widely used in the literature on task allocation and scheduling. However, in most real-life industrial system configuration problems, a set of additional design constraints needs to be taken into account. In this chapter, a methodology is presented, how a set of highly heterogeneous design constraints can be handled. The focus is on methods that are efficient, scalable, and extensible.

This chapter is structured as follows: First, a set of industry-relevant constraints are identified. After reviewing state-of-the-art solving approaches and identifying their limits, a new solving approach is presented. Experimental results show the effectiveness of the methodology. Finally, conclusions are drawn.

6.1 Design Constraints

Constraints tackle different aspects of system configuration, and thus are highly heterogeneous. Design constraints may have a variety of sources. Most relevant sources are:

- *safety considerations* – If safety analysis of the entire system has been performed (e.g. hazard and risk analysis, in accordance with ISO 26262 [35]), safety requirements can be derived. These requirements impose constraints on design decisions.

- *compatibility to legacy systems* – Automotive systems are usually designed in an evolutionary fashion. A previous version of the system is taken as a starting point and is extended with additional features, in order to satisfy current demands / requirements. Thus, legacy components impose constraints on design decisions.
- *engineer's experience* – Engineers who have been designing similar systems typically have figured out "best practices". These may exclude certain design decisions, thus imposing additional constraints.

Other sources, such as legal requirements, company-internal guidelines, standards, etc., can also be mentioned. For now, the sources of the design constraints is of no concern, the only concern is *how to satisfy them*. The aspect of *compatibility to legacy systems* will be re-visited in chapter 7.

Constraint Class	**Constraint Type**	**Literature**
A: limited resources	A-1: processor CPU speed	yes
	A-2: processor memory	yes
	A-3: bus bandwidth	yes
B: real-time behaviour	B-1: task deadline	yes
	B-2: communication deadline	yes
	B-3: end-to-end delay deadline	yes
C: allocation (task to processor)	C-1: dedicated processors	yes
	C-2: excluded processors	yes
	C-3: fixed allocation	yes
D: dependencies (task to task)	D-1: grouping	no*
	D-2: separation	yes
E: message routing (message to bus)	E-1: processor-internal only	no*
	E-2: dedicated bus-systems	no
	E-3: excluded bus-systems	no
	E-4: same bus	no
	E-5: separated bus-systems	no
F: frame packing (message to frame)	F-1: dedicated frame	no
	F-2: same frame	no
	F-3: separated frames	no

* not stated as a constraint, but used as means to reduce bus utilization

Table 6.1: Overview of design-constraint types

Table 6.1 provides a summary of relevant constraint types. They can be categorized within several classes, which correlate with the design-steps for configuring a distributed embedded real-time system. The constraint types are generic and thus can be applied to different domains (e.g. automotive, rail, aerospace, automation, etc.). Within the automotive domain, the AUTOSAR standard [5] is becoming the leading standard. Within AUTOSAR, possible design constraints have been specified in the *AUTOSAR system template*. Table A.1 gives an overview / extract. Beside slightly different names, the semantic of the constraint types from table 6.1 match well with those from the AUTOSAR system template. Below is a description of the constraint types. In addition, rationals and some examples are provided.

- *A: limited resources* – Embedded systems typically only have a small memory and moderate processing power. This imposes significant constraints on which kind of tasks they can execute.
- *B: real-time behaviour* – Real-time systems have to perform their tasks within pre-defined deadlines. This is the key requirement for scheduling and schedulability-tests.
- *C-1: dedicated processors* – A task may only be allocated on a sub-set of processors. This could be due to hardware requirements (e.g. a task was developed for an ARM architecture).
- *C-2: excluded processors* – A task is not allowed to be allocated on a set of processors. This is effectively the opposite of C-1.
- *C-3: fixed allocation* – The given allocation of a task to a processor must not be modified.
- *D-1: grouping* – Two or more tasks have to be allocated on the same processor. This could be due to mutual dependencies (e.g. two tasks sharing a data-structure).
- *D-2: separation* – Two or more tasks must not be allocated on the same processor. A typical reason would be that the two tasks form a software redundancy.
- *E-1: processor-internal communication* – Communication between two tasks is only allowed to occur within the same processor, but not via a bus-system. This could be due to safety requirements.
- *E-2: dedicated bus-systems* – A message may only be routed via a sub-set of bus-systems (e.g. only FlexRay may be used, since it provides a time-triggered communication schema).
- *E-3: excluded bus-systems* – A message must not be routed via a set of bus-systems. This is effectively the opposite of E-2.
- *E-4: same bus* – Two or more messages must be routed via the same bus.
- *E-5: separated bus-systems* – Two or more messages must not be routed via the same bus. Again, this could be due to redundancy.
- *F-1: dedicated frame* – A message must be packed into a dedicated frame. A typical reason would be to guarantee backward-compatibility to a previous bus-configuration.
- *F-2: same frame* – Two or more messages must be packed into the same frame. This could be to guarantee data-validity (e.g. if one message represents voltage, and the other represents current, than the calculated electric power is valid only if these two messages arrive at the same time).
- *F-3: separated frames* – Two or more messages must not be packed into the same frame. Again, this could be due to redundancy.

6.2 Literature Review Refinement – Constraints

Guaranteeing the real-time behaviour of a system is at the heart of the real-time systems community. Thus, all reasonable approaches for scheduling and task allocation which can be found in the literature address *real-time constraints*. Since schedulability is linked to resource utilization, also most works address *resource utilization constraints*.

Most works on task allocation are wrapped around a schedulability-test. The task allocation methodology deals with the resource utilization constraints, while the schedulability-test handles the real-time constraints. In [81] such an approach, based on SA, is presented. In addition, *task separation constraints* are used in order to account for redundancy. These constraints are addressed by adding a cost-term to the cost-function which punishes constraint-violations.

In [26] *allocation constraints* are addressed by means of *dedicated processors*. By specifying only one processor, *fixed allocation* can be modeled as well. Also, if an *excluded processors* constraint should be modeled, this could be done by setting the dedicated processors accordingly (i.e. $\mathcal{P}_{ded} = \mathcal{P} \setminus \mathcal{P}_{ex}$). Again, SA is used for problem solving, and the constraints are addressed by punishing their violations.

In [25] additional constraints are addressed. Beside traditional ones (e.g. limited resources, timing, dedicated processors) it also tackles *task grouping* as well as *task separation* constraints. The authors apply a constraint programming approach to solve the problem, using the `Choco` solver.

Open Issues

Interestingly, several constraint-types are not addressed in the literature. Especially constraints that focus on the configuration of the communication infrastructure have not been tackled yet. This involves aspects of *message routing* and *frame packing*. This can be explained as follows: Most works on system configuration (e.g. task allocation) use simplified models for cross-processor communication. These models do not cover all relevant details of the communication infrastructure. In chapter 5 this open issue is addressed. Due to the simplified models for cross-processor communication, the use of detailed constraints seems obsolete. For real-life systems though, these constraints are of high importance.

6.3 Approach to Satisfy Constraints

In general, there are several approaches how to satisfy constraints. Roughly, they can be categorized into 3 classes. This classification is tightly aligned with the question, how the (unconstrained) optimization problem is solved:

- If the optimization problem is solved by a *problem-tailored heuristic*, constraints can only be addressed by directly incorporating them into the heuristic. In the worst case, the entire heuristic may need to be re-designed, in order to address a constraint.

- If the optimization problem is solved by *constraint programming*, constraints can easily be added. Therefore, the constraint must be stated by a set of equations. The constraint-solver will take care of finding a solution.

- If the optimization problem is solved by *meta-heuristic search*, constraints can be addressed by adding a cost-term which punishes constraint-violations into the cost-function. The cost-term is minimized by the search, resulting in a solution where no constraint is violated.

Requirements

In this work, the system configuration problem is solved by both meta-heuristic search (based on SA), as well as by heuristics (e.g. frame packing heuristic). Thus design-constraints also have to be satisfied by different approaches. The goal of this chapter is to find a methodology for constraint handling which meets a set of requirements:

- *efficient* – During DSE, the number of constraint violations should be low. In the best case, no constraint is violated. The best obtained solution should not violate any constraint.
- *scalable* – The methodology should have low computational effort. This way, the methodology can scale up to industrial-sized problem instances.
- *extensible* – In case that new / additional constraint types need to be taken into account, it should be possible to extend the methodology, without re-designing large parts of the overall methodology.
- *transparent* – The way the methodology works should be (easily) understandable. This will help to extend the methodology, and to gain acceptance from industry.

Approach

A well known approach to handle large, complex problems is to apply *divide & conquer* patterns. This way the overall problem is split into smaller parts, which makes it easier to handle them. One key challenge is to find adequate dividing-points. Thereby not only the above mentioned requirements need to be taken into account, also two main aspects need to be thought of:

- How can a constraint type be satisfied? (e.g. by extending a heuristic)
- What are the pre-conditions that need to be satisfied, so that a constraint can be satisfied at all? (e.g. a *same frame* packing-constraint can only be satisfied if the messages are sent from the same processor, and are routed via the same bus)

In my opinion, an appropriate way is to divide the system configuration problem into the following **sub-problems**. This approach tries to consider all above mentioned requirements. In addition it considers how each (unconstrained) sub-problem is solved.

- *task allocation & message routing* – Task allocation and message routing are highly interwinded. Message routing constraints determines the design space for feasible task allocations. Thus, these two design steps should be handled concurrently.

- *frame packing* – Some frame packing constraints can only be satisfied, if certain task allocation and message routing pre-conditions are satisfied. However, since frame packing is solved by a tailored heuristic (whereas task allocation is solved by a meta-heuristic) it is advised to treat frame packing as a separated problem.

- *scheduling* – Scheduling decisions can be taken more or less independently from the previous design steps. Thus, scheduling can be treated as a separated problem.

Based on the proposed dividing into sub-problems, the subproblem-specific constraints will be addressed by the following **strategies**:

- *task allocation & message routing* – A set of admissible resources (processor for task, bus for message) is derived. Based on these sets, tasks are allocated only to admissible processors.

- *frame packing* – Frame packing is performed in several phases, each addressing a sub-set of packing constraints.

- *scheduling* – Since so new constraint types are introduced, state-of-the-art approaches can still be applied.

Guarantees

In general, only feasible system configurations (i.e. a system configuration where all constraints are satisfied) are of interest. In order to reach this goal, a solving methodology needs to be developed which is capable of satisfying all design constraints. To satisfy a constraint, two aspects need to be addressed. On the one hand, the methodology needs to tackle the constraint itself. On the other hand, the methodology can only succeed if the necessary pre-conditions of each constraint are also satisfied. E.g., in order to be able to satisfy the *same frame* packing constraint, both messages must originate from the *same processor* and need to be routed via the *same bus*. Thus, a key factor of the methodology is to ensure that all necessary pre-conditions are satisfied. Only then can constraint-satisfaction be guaranteed.

For some of the constraint types from table 6.1 a set of rules can be derived which allow to *resolve* the constraint. This means that pre-condition satisfaction is guaranteed. These resolving rules are applied within a preparation phase before starting DSE. Figuratively speaking, by applying *constraint resolving* the constraints are removed from the design space. Thus, the reduced remaining design space does no longer contain these infeasible regions. During DSE only the *probably feasible* regions of the design space are explored. Thus, DSE can be performed more efficiently.

Unfortunately, not all constraint types can be resolved. Table 6.2 shows, which constraint types can be resolved, which constraint types cannot be resolved, and why that is the case.

During DSE the search framework will explore the design space, by applying modification steps. These steps need to be enhanced, so that they incorporate the remaining constraints, i.e. constraint satisfaction rules are *encoded*. However, here pre-condition satisfaction is not guaranteed, thus infeasible system configuration can occur.

Type	before	during	Rationale
A-1		x	CPU utilization can only be checked after task allocation
A-2		x	memory utilization can only be checked after task allocation
A-3		x	bus utilization can only be checked after message routing and frame packing
B-1		x	deadline can only be checked after scheduling
B-2		x	deadline can only be checked after scheduling
B-3		x	deadline can only be checked after scheduling
C-1	x		a set of admissible processors is derived
C-2	x		a set of admissible processors is derived
C-3	x		allocation algorithm does not modify the allocation
D-1	x		tasks are grouped (forming a task cluster); task clusters are handled as "single elements" by task allocation
D-2		x	a set of excluded processors is derived dynamically
E-1	x		sender- and receiver-tasks are grouped
E-2	x		a set of admissible bus-systems is derived
E-3	x		a set of admissible bus-systems is derived
E-4	x		group sender-tasks; group receiver-tasks
E-5		x	message routing results from task allocation
F-1		x	only the dedicated frame will be used
F-2		x	demand E-4; perform frame packing in multiple phases
F-3	x		perform frame packing in multiple phases

Table 6.2: Constraint satisfaction time: "before" or "during" design space exploration

6.4 Extending the Search Framework

In order to incorporate *design constraints* into *system configuration design space exploration and optimization*, the following aspects of the search framework need to be adopted:

- *model* – The system-model needs to be extended, so that constraints can be modeled.
- *cost* – In order to determine how many constraints are violated, an additional cost-term needs to be added into the cost-function. The goal is to satisfy all constraints, thus minimizing the *constraint violations* cost-term.
- *neighbour* – Methods which generate a modified system configuration need to be extended, so that constraints are taken into account. The extended methods utilize information from the *constraint resolving phase.*

6.4.1 Cost Function

In order to determine how well a configuration satisfies the constraints, the cost-function needs to be extended with an additional cost-term. This term punishes constraint violations, and thus guides the search towards configurations where no constraints are violated. This cost-term itself is modeled by several terms. Each term handles a certain constraint type.

$$cost_{\text{constraint violations}} = \frac{\text{\# of elements that violate a constraint}}{\text{\# of elements that have a constraint associated}} \rightarrow \min \quad (6.1)$$

Again, the individual terms are grouped, using a *scaled weighted sum*. This way, each constraint type can be punished due to an individual weight, representing its importance of being satisfied. In addition, more insight can be gained which constraint types are violated due to which pattern. Figure 6.1 shows the extended cost-function.

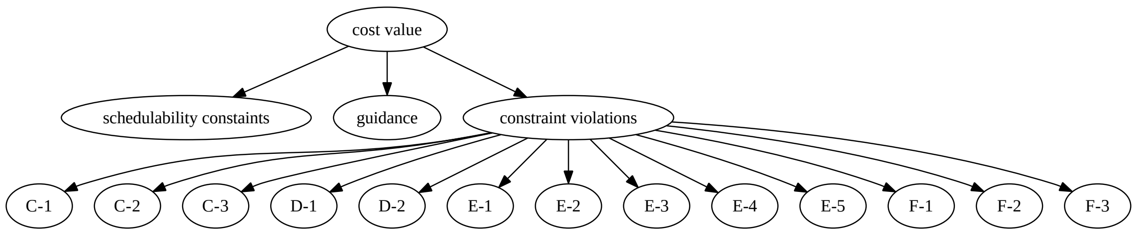


Figure 6.1: Extending cost-function with term for constraint violations

Note that some constraint types from table 6.1 are not encoded into the new cost-term. This is because they are already addressed by other cost-terms. Timing constraints (class B) are already covered by the term *schedulability constraints*, resource constraints (class A) are already covered inside the term *guidance*.

6.4.2 Task Allocation & Message Routing

The goal is to extend the neighbour function in a way that only feasible allocations are generated. This means that the following constraint types are satisfied for the generated allocations:

- task allocation (class C)
- task dependencies (class D)
- message routing (class E)

In order to achieve this goal, a set of rules are applied. These rules are based on problem-specific domain-knowledge. Thereby, *set theory* is used to express most of the rules. Beside the symbols which are already used throughout this work, several additional symbols are introduced.

The overall goal is to have a set of *admissible processors* for each task. This set is calculated in the constraint resolving phase. Then, during the DSE, the task allocation neighbour moves only choose a processor out of these sets.

E-1: To ensure that communication is performed only processor-internal, both sender-task and receiver-task have to reside on the same processor. This can be enforced, by grouping the sender-task and the receiver-task, forming a *task cluster*. A task cluster is treated as a single element by the allocation algorithm. This way, sender-task and receiver-task, which both are part of the task cluster, are always allocated to the same processor, and thus communication is always performed processor-internal.

Symbol	Description
ς	task cluster
$\mathcal{C}$	set of task clusters
adm	admissible
ded	dedicated
ex	excluded
stat	static
dyn	dynamic
gr	grouping
sep	separation

E-4: In order to ensure that two messages are routed via the same bus, their sender-tasks have to reside on the same processor, as well as their receiver-tasks have to reside on the same processor. These pre-conditions can be enforced by grouping of tasks. Sender-tasks are put into one task cluster, and receiver-tasks are put into another task cluster accordingly.

E-2 & E-3: Both constraint types are used to define feasible / infeasible message routings. By combining both constraint types, a set of admissible bus-systems can be derived for each message.

$$\mathcal{B}_{adm} = \begin{cases} \mathcal{B} \setminus \mathcal{B}_{ex} & \text{if} \quad \mathcal{B}_{ded} = \{\} \\ \mathcal{B}_{ded} \setminus \mathcal{B}_{ex} & \text{otherwise} \end{cases} \tag{6.2}$$

An admissible message routing implies a set of admissible processors X for the sender- and receiver-task of the message. Only processors connected to the admissible bus-systems of the message are potential candidates for hosting the sender- and receiver-task. This can be derived from the topology of the hardware architecture. However, since a task may send and receive several messages, only the intersected set is a potentially admissible processor for each task.

$$X = \bigcap_{m \in \tau} \mathcal{P} \text{ connected to } \mathcal{B}_{adm} \tag{6.3}$$

C-1 & C-2: Both constraint types are used to define feasible / infeasible allocations. By combining both constraint types, a set of admissible processors can be derived for each task. Thereby the set of admissible bus-systems (derived from E-2 & E-3) of the sent and received messages needs to be taken into account as well.

$$\mathcal{P}_{adm} = \begin{cases} (\mathcal{P} \cap X) \setminus \mathcal{P}_{ex} & \text{if} \quad \mathcal{P}_{ded} = \{\} \\ (\mathcal{P}_{ded} \cap X) \setminus \mathcal{P}_{ex} & \text{otherwise} \end{cases} \tag{6.4}$$

D-1: In order to ensure that two tasks reside on the same processor, the tasks should be grouped into a task cluster.

The concept of *grouping of tasks* can be utilized to resolve several constraint types, and ensure their satisfaction. If two (or more) tasks are grouped into a task cluster, the following rules apply: The set of admissible processors of a task cluster is the intersected sets of admissible processors of the clustered tasks.

$$\mathcal{P}_{adm}^{(\varsigma)} = \bigcap_{\tau \in \varsigma} \mathcal{P}_{adm}^{(\tau)} \tag{6.5}$$

If the tasks have grouping constraints associated, then the task cluster's grouping constraint is the union of those, except itself.

$$\varsigma_{gr} = \bigcup_{\tau \in \varsigma} \tau_{gr} \setminus \tau_i \tag{6.6}$$

If the tasks have separation constraints associated, then the task cluster's separation constraint is the union of those.

$$\varsigma_{sep} = \bigcup_{\tau \in \varsigma} \tau_{sep} \tag{6.7}$$

By performing pre-processing steps, during which all the above described rules are applied, the task graph is transformed into a *task cluster graph*. Each task cluster has the following attributes, which are of main interest for the allocation algorithm:

- a set of admissible processors
- a set of separation constraints

The allocation algorithm will perform allocation modifications to the task cluster graph. Thereby, a task cluster is allocated only to a processor out of the set of admissible processors. In addition, the allocation algorithm tries to consider the remaining constraint types (e.g. grouping, separation) which could not be resolved before DSE. This can be achieved by applying a set of rules which are performed dynamically during DSE:

C-3: If an allocation is given and defined as *fixed*, the allocation algorithm will not modify this allocation during DSE.

D-2: *Separation* constraints can only be resolved during DSE. Based on the separation constraints, a set of excluded processors can be derived dynamically.

$$\mathcal{P}_{ex.dyn} = \mathcal{P} \text{ of tasks that the current task must be separated from} \tag{6.8}$$

Based on the set of dynamically excluded processors, the set of admissible processors can be dynamically refined. This can be applied to tasks, as well as to task clusters accordingly.

$$\mathcal{P}_{adm.dyn} = \mathcal{P}_{adm} \setminus \mathcal{P}_{ex.dyn} \tag{6.9}$$

Based on these rules and considerations, the search-framework is enhanced as follows: see algorithms 6.1 and 6.2.

Algorithm 6.1: Re-allocate task-cluster

```
  Input: set of task-clusters
  Input: set of processors
1 begin reAllocateTaskCluster
2     cluster $\varsigma$ = randomly pick a task-cluster;
3     processor $\rho$ = randomly pick a processor, out of task-cluster's admissible
      processors $\mathcal{P}_{adm.dyn}$;
4     allocate cluster $\varsigma$ to processor $\rho$;
5     /* also consider grouping-constraints */;
6     foreach task-cluster $\varsigma_{gr}$ which has grouping-constraint with cluster $\varsigma$ do
7         allocate task-cluster $\varsigma_{gr}$ to processor $\rho$;
8     end
9 end
  Output: modified allocation
```

Algorithm 6.2: Swap allocation of task-clusters

```
  Input: set of task-clusters
  Input: set of processors
1 begin swapTaskClusterAllocation
2     cluster $\varsigma_1$ = randomly pick a task-cluster;
3     cluster $\varsigma_2$ = randomly pick a task-cluster, which resides on another processor;
4     swap allocation of $\varsigma_1$ and $\varsigma_2$;
5 end
  Output: modified allocation
```

6.4.3 Frame Packing

In order to be capable of satisfying the frame packing constraints, the following pre-conditions need to be satisfied:

F-1: The *dedicated frame* is sent by the associated processor, and the message fits into the dedicated frame's payload capacity.

F-2: Both messages are sent from the same processor, are routed via the same bus, and fit into a frame's payload capacity. Thus, a *E-4: same bus* constraint is implied.

F-3: No pre-condition is needed.

Message routing related pre-conditions can be satisfied by the methodology which was presented in the previous section. In order to actually satisfy the frame packing constraints, an extended frame packing heuristic is proposed (see algorithm 6.3). Therein, frame packing is performed within 4 phases. Phase 1 to 3 addresses the different packing constraint types, while phase 4 handles the unconstrained messages:

- *phase 1* – Messages with a *dedicated frame* constraint are packed into those dedicated frames (if possible).

Algorithm 6.3: Heuristic for frame packing with packing constraints

```
Input: messages
Input: constraints
begin packConstrainedMessagesIntoFrames
    /* pack messages with packing-constraints */;
    /* phase 1: F-1: dedicated frame */;
    dedicated messages = filter(messages, dedicated frame constraint);
    foreach dedicated message do
        if dedicated message fits into dedicated frame then
            pack(dedicated message, dedicated frame);
        end
    end
    /* phase 2: F-3: separated frames */;
    separated messages = filter(messages, separated frames constraint);
    foreach separated message do
        if separated message not packed yet then
            pack(separated message, new frame);
        end
    end
    /* phase 3: F-2: same frame constraint */;
    same messages = filter(messages, same frame constraint);
    foreach same message do
        if same messages not packed yet and same messages fit into a frame then
            pack(same messages, new frame);
        end
    end
    /* pack messages without packing-constraints */;
    /* phase 4: messages without constraints */;
    remaining messages = filter(messages, (no constraint and not packed yet));
    packMessagesIntoFrames(remaining messages) /* apply algorithm 5.2 */;
end
Output: frames
```

- *phase 2* – Messages with a *separated frames* constraint are packed into separated new / empty frames.
- *phase 3* – Messages with a *same frame* constraint are packed together into a new / empty frame (if possible).
- *phase 4* – All remaining messages are packed.

Within phase 2 and 3 the approach packs constrained messages into newly generated frames, to ensure constraint satisfaction. However, this comes at the cost of an increased number of frames and thus decreased packing density. If packing density is of high importance, the approach can be extended in a way that all *already used* frames are also considered as packing-candidates within phase 2 and 3.

6.4.4 Gateway Configuration

In this work it is assumed that a gateway is a processor which is connected to several bus-systems. Since these bus-systems may have different protocols (e.g. CAN-to-FlexRay gateway) the incoming frames are unpacked, the contained messages are routed to the according outgoing bus-interface, and the routed messages are re-packed at the outgoing bus-interface. Therefore, the major aspect of gateway-configuration is frame packing. As a consequence, the methodology for constrained frame packing can be directly applied to gateways as well.

6.5 Experimental Evaluation – Constraint Handling

In order to evaluate the effectiveness and efficiency of the proposed methodology for constraint handling, the methodology is applied to several test cases. These test cases are generated by the tool GenTAP which can generate synthetic pseudo-random TAP instances. However, since GenTAP does not allow stating the constraint types from table 6.1, a two-step generation approach is used:

- *step 1* – use GenTAP to generate basic TAP instances
- *step 2* – extend the basic TAP instance by (pseudo-randomly) adding the additional constraints to the TAP instance

These extended TAP instances are then fed into the system configuration methodology, and the framework searches for an optimal system configuration.

6.5.1 Test Cases & Evaluation Metrics

The overall goal of the proposed methodology is to handle constraints efficiently during DSE. Therefore, the following metrics are used for evaluation:

- number of DSE iterations until the *first feasible system configuration* is found
- number of *constraint violations* during DSE
- attributes of *best obtained solution* at the end of DSE

The generated test cases are solved using different approaches:

- *guidance* – No constraint resolving is applied. Only the cost-function is extended by the cost-term *constraint violations*, which is guiding the search towards feasible regions of the design space.
- *proposed methodology* – Constraint resolving is applied. In addition, the constraint-aware neighbour moves are used.

Hypothesis: By applying constraint resolving and constraint-aware neighbour moves, the constraints can be handled more efficiently than just using guidance by the cost-function.

Parameter	**Value/Range**	**Unit**
number of bus-systems	1	
bus baudrate	125	kbaud/s
number of processors	10	
processor utilization	60	%
number of tasks	100	
number of messages	300	
number of transactions	7	
period	10 to 1000	ms

Table 6.3: Test case parameters for constraint handling evaluation

Table 6.3 shows the parameters for the evaluation. This basic model is extended by adding the specific constraint types, according to table 6.4. Note that the constraint type of class E (i.e. message routing) can only be of interest if there exist several routes between a pair of processors. That is true in case of redundant networks. In automotive systems, there is currently little (or no) redundancy for bus-systems. Therefore, this constraint type class is of lower interest in this evaluation.

Constraint Type	**Apply to**
C-1: dedicated processors	10% of tasks
C-2: excluded processors	50% of tasks
D-1: grouping	10% of tasks
D-2: separation	50% of tasks
E-1: processor-internal communication	10% of messages
F-2: same frame	10% of messages
F-3: separated frames	50% of messages

Table 6.4: Constraints added to basic model

6.5.2 Results

To get a better insight of how the constraints affect the search performance, each constraint type is evaluated separately. To verify the hypothesis the number of constraint violations (of the best obtained configuration so far) are plotted and analysed. The y-axis represents the constraint-violations cost-term (which is scaled between 0 and 1) and the x-axis represents the search-iterations of the DSE. After 30 000 iterations the DSE is stopped.

Dedicated Processors

By applying constraint resolving and using constraint-aware allocation moves, it is expected to see no constraint violations. Each generated configuration should be feasible. In contrast, by just using guidance via the cost-function, it is expected to see several constraint violations at the beginning of the DSE, and then the number of constraint violations gradually being reduced.

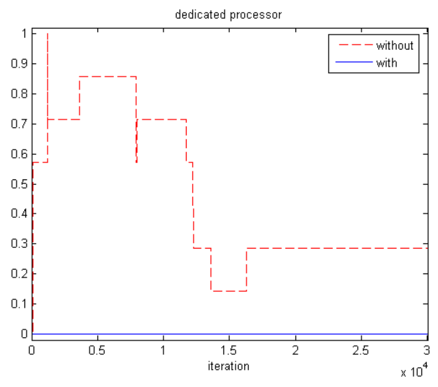


Figure 6.2: Constraint violations – dedicated processors

Figure 6.2 shows the according plot of constraint violations. It confirms the validity of the hypothesis. By applying constraint resolving and constraint-aware neighbour moves, no constraint violations occur, i.e. every generated configuration is feasible.

When only using the guidance from the cost-term, a significant number of constraints are violated (i.e. these are infeasible configurations). Also, the SA does not manage to find a feasible configuration within the max. number of iterations.

Excluded Processors

Similar results as for dedicated processors can be observed, and again confirm the hypothesis. By applying constraint resolving and constraint-aware neighbour moves, no constraint violations occur. By just using the guidance of the cost-term, several constraint violations occur.

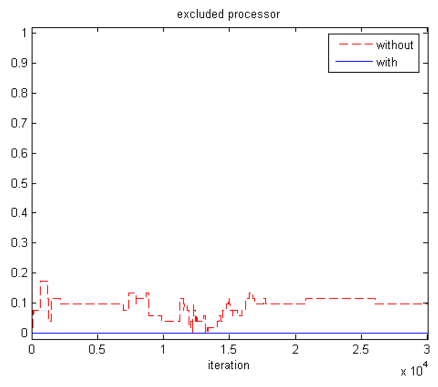


Figure 6.3: Constraint violations – excluded processors

As figure 6.3 shows, the *guidance* approach manages to satisfy all constraints after 12 179 iterations. However, constraints are violated again later in the DSE. This indicates that cost-term weights could be improved.

In addition it can be seen that satisfying *excluded processors* constraints is easier than satisfying *dedicated processors* constraints. That is not surprising since *dedicated processors* constraints are more restrictive.

Task Grouping

Figure 6.4 shows the evaluation for *task grouping* constraints. Again, it confirms the validity of the hypothesis. By constraint resolving and constraint-aware neighbour move, no constraints are violated. By just using cost-term guidance, a significant number of constraints are violated, and no feasible configuration was found.

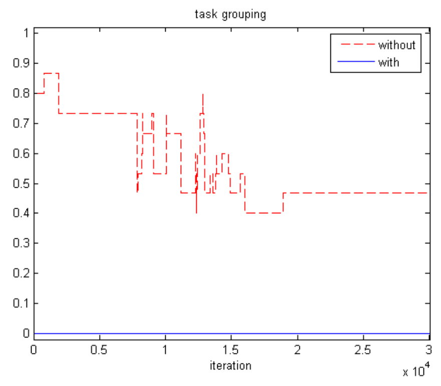


Figure 6.4: Constraint violations – task grouping

Task Separation

Figure 6.5 shows the evaluation for *task separation* constraints. For this constraint type, both approaches fail to find a feasible configuration. This reflects the fact that this constraint type cannot be resolve before the DSE.

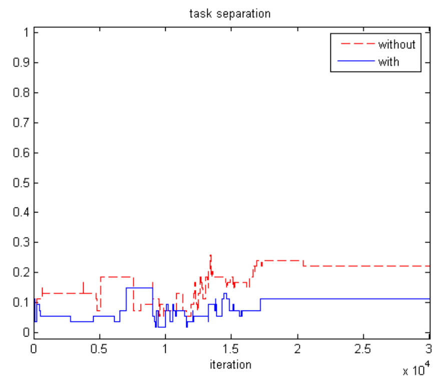


Figure 6.5: Constraint violations – task separation

However, the proposed methodology manages to reduce the number of constraint violations twice as much as the guidance approach. By using higher weights for the cost-term, this could even be further improved.

Processor-internal Communication

Figure 6.6 shows the evaluation for *processor-internal communication* constraints. Again, the hypothesis is confirmed. Constraint resolving and constraint-aware neighbour moves significantly improve the search.

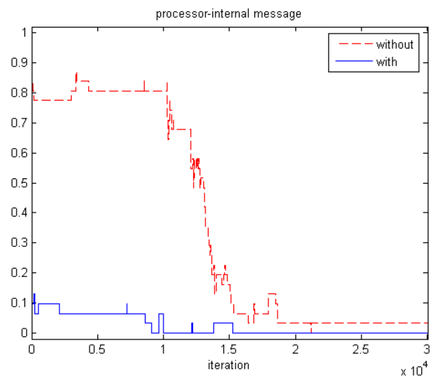


Figure 6.6: Constraint violations – processor-internal communication

Note that some constraints are violated during the first third of the DSE. This is because not all sender- and receiver-tasks were clustered, in order to address issues concerning processor utilization overload. However, after about 10 000 iterations, all constraints are satisfied. In contrast, by just using cost-term guidance, no feasible configuration was found. The reason why the number of violations is reduced quite fast is that *minimizing the cross-processor communication* is also used as an *optimization objective*.

Separated Frames

Figure 6.7 shows the evaluation for *separated frames* constraints. It confirms the hypothesis, and shows the effectiveness and efficiency of constraint-aware frame packing heuristics: No constraints are violated. In contrast, by just using cost-term guidance some constraints are violated.

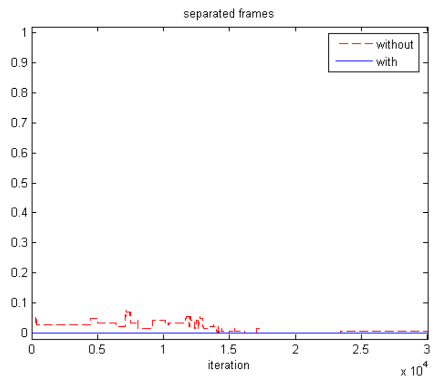


Figure 6.7: Constraint violations – separated frames

6.6 Concluding Remarks

Experimental evaluation shows that the proposed methodology is effective and efficient in satisfying heterogeneous constraints. However, some researchers from the real-time systems community have raised a reasonable question concerning the proposed approach:

> *"Checking constraints like D-1 or E-1 has a very low CPU cost in order to reject a configuration before performing the scheduling parameter assignment and schedulability-test. Perhaps millions of infeasible configurations can be checked in a few seconds. Thus, why not simple reject infeasible configurations?"*
>
> – Reviewer #3 of [59]

It sounds reasonable to simply reject infeasible configurations, and only perform schedulability tests for feasible configurations. This way DSE would be faster, since fewer (time-consuming) schedulability-tests would be performed. However, it would not tackle the issue that the *cost-term guiding* approach is not effective. It would still generate a lot of infeasible configurations.

To put it simply: If I know that certain regions of the design space are infeasible, it does not make sense to explore these regions, just to find out that they are indeed infeasible. Instead, these regions should be ignored by the search – which is what the proposed methodology does.

6.7 Additional Applicability of Methodology

The presented methodology allows to tackle a wide range of heterogeneous constraint types. Constraints may stem from different sources, e.g. safety considerations. The methodology is targeted towards automated DSE. However, the methodology can also be used to tackle a different engineering approach: *engineer-algorithm-collaboration*

The key idea of this approach is: The engineer takes the most important design decisions manually (e.g. allocating 10% of the tasks to processors). Then an algorithm / tool shall take the remaining design decisions in an automated manner. Thereby, the decisions which have been taken by the engineer are treated as constraints by the algorithm. Thus the engineer's decisions are not modified by the algorithm. This way, the engineer stays in control of the DSE, while still allowing exploring a wide design space within reasonable time.

Chapter 7

Evolving System Configurations

In chapter 5 the *system configuration problem* is solved in a realistic manner by incorporating a realistic communication-model, and by tackling *frame packing.* The solving algorithms are allowed to take any design decision, and optimization is targeted at prevailing objectives (e.g. finding a system configuration which uses minimal bus utilization).

Chapter 6 introduces a set of heterogeneous constraints, and presents a methodology how these constraints can be handled in an efficient way. These constraints can be utilized to address different scenarios:

- *design constraints* – Based on precedent engineering steps (e.g. hazard and risk analysis) adequate strategies are developed (e.g. usage of redundancy) and resulting constraints are derived (e.g. redundant tasks must not be allocated to the same processor).
- *user constraints* – The engineer takes the most important design decisions (e.g. allocating 10% of the tasks to processors), and then the solving algorithms shall take the remaining decisions. However, the decisions taken by the engineer must not be modified, thus they have to be treated as constraints by the solving algorithms.
- *legacy constraints* – Some components of the system may be re-used from a previous system-version. This re-use imposes certain constraints (e.g. 1 processor including all its tasks is re-used. Thus this allocation must not change, nor shall the bus-interface of the processor change).

Especially the last of the scenarios is of main interest in the automotive domain. Therefore, some insight into the product development process of the automotive domain is desirable.

7.1 Automotive Product Development Process

The development of automotive electronic systems is performed by collaborative engineering between the OEM and its Tiers. Each party has dedicated responsibilities [13]:

- The OEM is responsible for system specification and system integration. This involves decisions for *task allocation* and (more importantly) *bus configuration* (i.e. frame packing and frame scheduling) of the main bus-systems. One of the driving objectives for the OEM is the re-use of systems between vehicles.

- The Tier is responsible for *electronic control unit* (ECU) integration. This involves software component development and *task scheduling.* The bus-schedule which is provided by the OEM acts as an integration-interface for the ECU. In addition, the Tier configures local bus-systems (e.g. LIN for smart sensors and actuators).

To foster the re-use of systems, the OEM follows a *platform-based* approach: The OEM designs a communication-platform (consisting of several interconnected bus-systems) which shall be used for several vehicles within the coming years. Therefore, the bus configuration must be *extensible* towards future communication demand. During the following years, this communication-platform is used in several vehicles.

If vehicle-specific features are added, the initial bus configuration may need to be extended (e.g. an additional ECU is added which causes additional frames). Such changes must be done in a way that the extended bus configuration stays *backward-compatible* to the initial communication-platform specification, i.e. the initial bus configuration must not be broken. After using the communication-platform for several years, a new communication-platform is designed, and is the basis for future vehicle-projects.

Scenarios for Future Changes

Clearly, it is impossible to exactly predict which changes will occur in the future. However, reasonable estimations can be derived by analyzing which kind of changes have occurred in previous vehicle-projects. Based on discussions with OEMs (Audi, BMW) and Tiers (AVL, Delphi), a set of likely change-scenarios are derived.

	Change	**Impact on**
T-1	add new task	processor schedule
T-2	extend existing task's functionality	task's WCET
T-3	modify (decrease) period / deadline of task	processor schedule
P-1	add new processor	bus schedule (if processor sends frames)
M-1	add new message	frame packing, bus schedule
M-2	modify (increase) data size of existing message	frame packing, bus schedule
M-3	modify (decrease) period / deadline of message	frame packing, bus schedule
M-4	modify (add) receiver-processor of message	-
F-1	add new frame	bus schedule
F-2	modify (decrease) period / deadline of frame	bus schedule
F-3	modify (add) receiver-processor of frame	-

Table 7.1: Scenarios for future changes

The different change-scenarios from table 7.1 can be grouped into two classes, based on their impact: (1) changes which impact *processor configuration* are of main interest for Tiers, and (2) changes which impact the *bus configuration* are of high interest for the OEM.

7.2 Scope & Outline

This chapter focuses mainly on the interest of the OEM and thus focuses on bus configuration. It addresses two aspects of the automotive design approach:

1. When designing the communication-platform, not all future requirements are known in detail. Thus, the objective is to design a bus configuration which is (a) extensible towards future communication requirements, and (b) robust against parameter-uncertainties. A methodology which can build a bus configuration which satisfies these objectives is presented in section 7.3.
2. When using the communication-platform for a vehicle-project, it may need to be adapted towards vehicle-specific features. If additional communication needs to be performed via the bus, the bus configuration needs to be extended. In section 7.4 a methodology is presented, how a bus-schedule can be extended in a way that it is (a) backward-compatible to the previous version, and (b) is itself extensible towards future modifications.

By utilizing both methodologies the OEM can tackle main engineering tasks within the bus configuration's life cycle.

7.3 Extensible Bus Configuration

When designing a platform, the main goal of the OEM is that the platform can be used for several different vehicles in the coming years. For the aspect of *task allocation* this can be addressed by applying *load balancing.* By evenly balancing the utilization of processors, it becomes easier to add additional task later on. Load balancing can be achieved by applying the methodologies which have been presented in chapter 4.

Even more important is the aspect of *bus configuration.* Why? Simply speaking, every message / frame which is sent via a bus-system acts like a global variable between processors. Any change which is applied to the bus configuration later on may impact on all processors which are connected to that bus-system. Since changes are associated with re-negotiations with Tiers, re-design of bus-interfaces, and re-verification of the system, it is in the key interest of the OEM to design an *extensible bus configuration* for the platform. The associated challenges can be best explained by looking at some small examples.

Example – Frame Packing

Assume a processor that outputs 3 messages which need to be packed into frames before being transmitted via the bus. Assume a max. payload of 8 byte, and 64 bit frame overhead (as applies to CAN and LIN).

- m_1: 8 bit, 50 ms
- m_2: 16 bit, 100 ms
- m_3: 32 bit, 150 ms

When packing these messages into frames, different objectives can be addressed. Typically, *minimizing bandwidth demand* is the objective. However, considering *extensibility* may be evenly important.

A: min. bandwidth		**B: extensibility**	
f_1: 8+16+32 bit, 50 ms	2400 b/s	f_1: 8 bit, 50 ms	1440 b/s
		f_2: 16+32 bit, 100 ms	1120 b/s
	2400 b/s		2560 b/s

At first it seems obvious that bandwidth-minimizing frame packing is the better approach. However, by taking a look at what happens if a change occurs in the future, then this answer changes. Assume an additional message needs to be transmitted. The additional messages must be packed in a way that the initial packing stays intact.

- m_4: 16 bit, 100 ms

extending A		**extending B**	
f_1: 8+16+32 bit, 50 ms	2400 b/s	f_1: 8 bit, 50 ms	1440 b/s
f_2: **16** bit, 100 ms	800 b/s	f_2: 16+32+**16** bit, 100 ms	1280 b/s
	3200 b/s		2720 b/s

Due to the dense packing which is needed to achieve bandwidth minimization, additional messages do not fit into the existing frame. Thus, a new frame needs to be created which causes additional overhead. However, if extensibility has been taken into consideration earlier, existing frames provide free space left, and additional messages can be added without needing to create new frames. Thus, no additional overhead is induced.

Conclusion: Extensibility-aware frame packing is a key factor for platform-based design. Although it causes higher bandwidth demand at first, this pays off later on when additional messages are added into existing frames. Thus, a trade-off between bandwidth-efficiency and extensibility is the key to success.

Example – Frame Scheduling

Assume 3 frames which need to be scheduled on a CAN bus, which uses a priority-based arbitration schema. The CAN bus is operated at 125 kbaud/s, thus each frame's transmission time is 1 ms.

- f_1: T=10ms, D=4ms
- f_2: T=10ms, D=7ms
- f_3: T=10ms, D=9ms

It is known that for a set of independent frames DMPO is the optimal scheduling approach [4]. Thus, f_1 would get higher priority than f_2 and f_3. However, this only determines the priority ordering, but it does not state anything about absolute priority-levels. In the literature, the priority ordering is mapped 1:1 to priority-levels. However, different transformations may be more beneficial in terms of extensibility.

A: dense priority-levels		**B: extensibility**	
f_1: priority = 0 (highest)	R = 2 ms	f_1: priority = 0 (highest)	R = 2 ms
f_2: priority = 1	R = 3 ms	f_2: priority = 5	R = 3 ms
f_3: priority = 2	R = 4 ms	f_3: priority = 10	R = 4 ms

At first it seems irrelevant which absolute priority-levels are assigned to the frames. In both cases the deadlines are met. However, the impact of the choice becomes clear if an additional frame must be scheduled.

- f_4: T=10ms, D=6ms

extending A		**extending B**	
f_1: priority = 0 (highest)	R = 2 ms	f_1: priority = 0 (highest)	R = 2 ms
f_4: priority = **1**	R = 3 ms	f_4: priority = **3**	R = 3 ms
f_2: priority = **2**	R = 4 ms	f_2: priority = 5	R = 4 ms
f_3: priority = **3**	R = 5 ms	f_3: priority = 10	R = 5 ms

According to DMPO the new frame f_4 needs to get a priority between f_1 and f_2. If there is no empty priority-level left between f_1 and f_2 (see case A) then the priority-levels must be re-assigned. This may cause a significant number of changes which need to be propagated to all effected processors. In contrast, if the priority-levels are initially assigned in a way that there are empty priority-levels left in between them (see case B) then additional frames can be added without causing any priority-level re-assignments.

Conclusion: Extensibility-aware priority assignment is a key for platform-based design. By leaving empty priority-levels in between frames, additional frames can be added to the schedule later on, without needing to re-assign priorities to all frames.

7.3.1 Problem Statement & Definition

The goal of this investigation is to find a methodology for building a bus configuration which addresses the delicate trade-off between *resource efficiency* and *extensibility*. Efficient resource usage is needed to satisfy the current communication specification at reasonable hardware-costs, whereas extensibility is needed to tackle future changes (see table 7.1). Within this context, *extensibility* is defined as follows:

> *"Extensibility is the ability of a distributed real-time system to incorporate additional elements (i.e. messages and frames) later on, while maintaining the initial schedule. For time-triggered bus-systems this implies that the initial time-slot assignment must not be modified. For priority-based bus-systems it implies that the initial priorities must not be modified. In addition it implies that the initial frame packing is not modified."*

– Pölzlbauer et al. [61]

The methodology shall build a bus configuration which *minimizes the bandwidth demand* [60] and *maximizes the extensibility* [61]. It should also *maximize robustness against parameter-uncertainties* to account for uncertainties concerning future changes. As extensibility causes some additional bandwidth demand, this is a conflicting multi-objective optimization problem.

7.3.2 Literature Review Refinement – Extensibility and Robustness

The research on *robustness* of real-time systems mainly focuses on the *sensitivity* of a system to uncertainties, changes and failures. The seminal work in this area is that of Punnekkat [70]. Punnekkat looked at by how much can individual tasks have their WCET increased before the system is no longer schedulable and which task is the most sensitive, i.e. which task's deadline is exceeded. He also looked at the number of re-executions, due to failures, that could be performed before the system was no longer schedulable.

Building on top of Punnekkat's approach, researchers used it as part of designing systems. In [7] sensitivity was used as an objective in a search algorithm such that robust task allocations are obtained. Later work [86] extended this work to look at how it could be done more efficiently using MILP. In [29] a measure of *robustness* is used that identifies the degree of error in key parameters which can be accommodated when performing task allocation. One parameter was by how much WCETs could be increased.

The research on *extensibility* of real-time systems tackles the issue of *adding extra functionality* to systems (i.e. adding tasks as well as increasing the WCET of existing tasks). Emberson [8, 27] used a SA algorithm to design task allocations that were robust to change. Emberson's approach to develop a task allocation not only satisfied the baseline, or original, design problem but also satisfied as many potential change scenarios without any *unnecessary changes*. If an unnecessary change was needed then it was minimized. A necessary change would include adding a new task priority if a new task was introduced.

Researchers have adopted these approaches (which have originally been developed for task scheduling) to scheduling of communication networks. In [84] a time-triggered system is made robust to new frames (and tasks), and increases in the existing frame's *worst-case transmission time* (WCTT) (as well as task's WCET). This work is probably the closest to the problem at hand. However, it does not address frame packing, which introduces additional challenges.

Research on *frame packing* focus on *minimizing bandwidth demand* [74], *maximizing schedulability* [73], or *maximizing reliability* [78]. No work could be found in the literature which addresses frame packing in the context of extensibility.

Open Issues

By addressing both frame packing and frame scheduling in the context of *extensibility* and *robustness*, a set of additional questions and challenges arise:

- Shall extensibility be incorporated into frame packing, frame scheduling, or both?
- What are suitable metrics to determine extensibility at the frame-level?

- Can frame packing handle timing-uncertainties?
- How to balance the conflicting objectives of extensibility, robustness and resource-efficiency?

7.3.3 Strategy & Metric

Ideally, future modifications as outlined in table 7.1 will cause no changes for the bus configuration. Modifying the data size of existing messages and/or adding new messages will not result in the need to add additional frames. Modifying the timing-attributes of messages (and frames) will neither cause deadline violations nor the need to re-build the schedule. Adding additional frames will not break the schedule.

Clearly, additional elements (i.e. messages and frames) can only be added to the bus configuration in a *low impact* manner if the bus configuration provides *free resources*. Thus, the **strategy** is to plan these free resources, when designing the initial bus configuration.

- *frame level* – At the frame level *growth margins* must be planned. This means that frames are not fully packed with messages. Instead, a defined fraction of the frames capacity is reserved for future demands. This reserve accounts for (a) adding additional messages, and (b) increasing messages data size.
- *schedule level* – At the schedule level *growth margins* must be planned as well. For priority-based bus-systems this means that not all priority-levels are used. For time-triggered bus-systems this means that a set of non-used time-slots are added to the schedule, in order to reserve resources for future demands. This reserve accounts for adding additional frames.
- *robustness* – In order to account for modifications of timing-attributes (such as period or deadline) the schedule should be built in a way that it is robust against timing-uncertainties. This can be achieved by minimizing sensitivity.

In order to determine the degree of extensibility of a bus configuration (and thus measure how close the configuration is towards the ideal situation) a set of **metrics** are needed.

- *frame growth potential* – The ability of adding additional messages or increasing the data size of messages can be used to determine the extensibility at the frame level. This metric measures this ability by analyzing the free payload capacity of a frame.

$$gp_f = \frac{pay_{f\ \text{free}}}{pay_{f\ \text{max}}} \tag{7.1}$$

  Since this gives a growth potential for each frame, an overall value is desirable. A reasonable estimation can be derived if the average growth potential is calculated.

$$gp_{\mathcal{F}} = \text{avg}\,\{gp_f\}_{\ \forall f \in \mathcal{F}} \tag{7.2}$$

  If the minimum would be used, the entire configuration would be judged by a single frame only. This would be an overly pessimistic estimation. Similarly, the maximum would be overly optimistic, for the same reason.

- *schedule growth potential* – The ability to add additional frames into a schedule has to be measured protocol-specific. For TDMA-based protocols, the number of non-assigned time slots can be used.

$$\mathrm{gp}_{TDMA} = \frac{\|slots_{\mathrm{free}}\|}{\|slots\|} \tag{7.3}$$

  FlexRay-specific metrics which are tailored to its dynamic segment can be found in [76]. For priority-based protocols (such as CAN) the first intuition might be to use the number of free priority-levels as a metric. However, having free priority-levels left does not guarantee a feasible schedule. Thus the number of additional frames which can be added before the system becomes un-schedulable should be used as a metric.

$$\mathrm{gp}_{CAN} = \frac{\|\mathcal{F}_{\mathrm{added}}\|}{\|\mathcal{F}\|} \tag{7.4}$$

  Thereby, the frames' attributes (period, deadline, payload, priority) should be set randomly, but within specified ranges (such as: low, medium, high) derived from previous change-scenarios.

- *robustness of schedule against timing-uncertainties* – Robustness analysis can be applied in order to ensure that schedulability is still guaranteed even if parameter-uncertainties exist. Uncertain transmission time is linked to the frame's payload, and thus can be tackled by the *frame growth potential* (see above). Thus, the focus here is on uncertainties for periods and deadlines. In order to determine the configuration's robustness, T and D of messages (and consequently also of frames) are decreased until the system becomes un-schedulable. This accounts for higher sampling rates and tighter deadlines. The more these parameter-values can be decreased the higher is the robustness.

$$rob = \frac{\Delta D_f}{D_f} \quad {\scriptstyle \forall f \in \mathcal{F}} \tag{7.5}$$

The individual conflicting objectives are combined into an **objective-function** (or cost-function), using a normalized weighted sum. The importance of each objective is set by its weight and can be balanced according to the OEM's desire.

- o_1 – bandwidth consumption of the frame-set, according to equation (5.19)
- o_2 – frame growth potential, according to equation (7.2)
- o_3 – schedule growth potential, according to equation (7.3) and (7.4)
- o_4 – robustness of schedule against timing-uncertainties, according to equation (7.5)

A proposal for the weights of the objectives (see table 7.2) were derived in accordance to discussions with industrial partners, and additional considerations. Minimizing bandwidth consumption and maximizing frame growth potential are equally balanced. This represents the trade-off between resource efficiency and extensibility. Maximizing schedule growth potential gets slightly less importance. This accounts for the following: Higher frame growth potential enables to add additional messages without the need to generate additional frames for them, thus there is no impact on frame scheduling. Only if the existing frames can no longer incorporate additional messages, new frames must be generated. Maximizing robustness gets even less importance, since it is estimated that changes in period and/or deadlines are rare.

Objective	**Weight**
min. bandwidth consumption	10
max. frame growth potential	10
max. schedule growth potential	8
max. robustness against timing-uncertainties	5

Table 7.2: Proposed balancing of objectives when building extensible bus configurations

7.3.4 Solving Methodology

The proposed methodology for solving the problem at hand is outlined in figure 7.1[1]. Starting from a given task allocation, the methodology for bus configuration consists of two phases. First, a preliminary bus configuration is generated. Second, this bus configuration is refined and optimized according to the stated objectives. The preliminary bus configuration is generated by applying established bus configuration heuristics:

- bandwidth minimizing frame packing (as described in section 5.5)
- frame priority assignment according to DMPO

This preliminary bus configuration is fed into an optimization framework which is based on SA. The framework takes a bus configuration and modifies it. The modified bus configuration is evaluated, if the modification improves the bus configuration. This is repeated until the search converges towards an optimum. Evaluation is performed by applying the multi-objective objective-function, which was presented earlier. A modification between two bus configurations is generated by one of the following *neighbour moves*:

- *move a message into another frame* - Randomly choose a message, and then move it into another frame that originates from the same processor. If the original frame becomes empty by moving the message, the frame can be removed.
- *move a message into a new/empty frame* - Randomly choose a message, and then move it into a new/empty frame with the same source-processor. The new frame is given a priority-level close to the priority of the original frame. Either one level above or one level below.
- *swap two messages between two frames* - Randomly choose a message. Randomly choose another message that originates from the same processor. Then, swap the packing of the two messages.

[1]Note that the methodology is discussed as applied to a priority-based bus-system, like CAN. If it is applied to a TDMA-based bus-system, then the scheduling-related aspect has to be adapted accordingly: Instead of modifying the priority assignment of a frame, the time-slot assignment is modified.

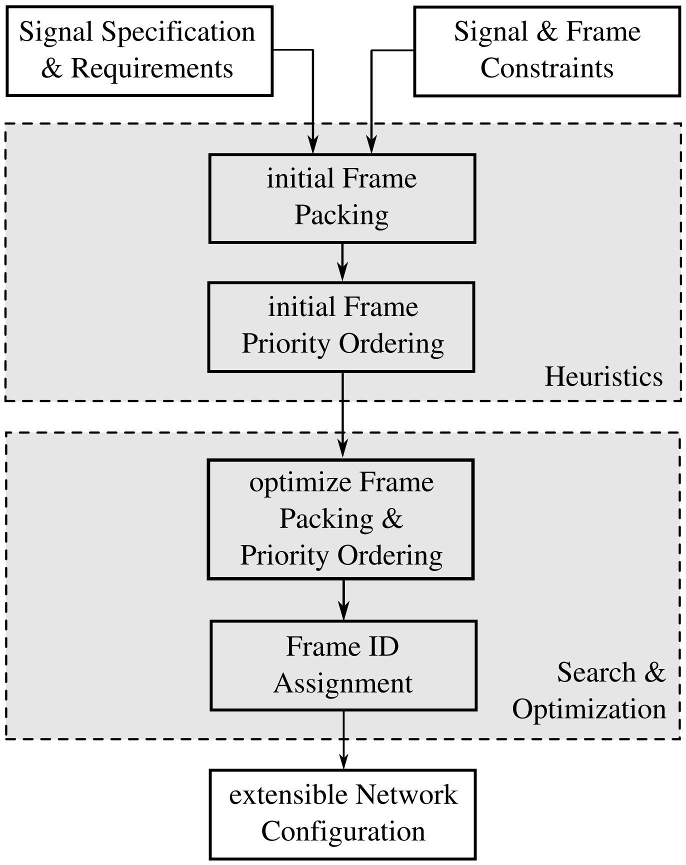

Signal Specification & Requirements
Signal & Frame Constraints
initial Frame Packing
initial Frame Priority Ordering
Heuristics
optimize Frame Packing & Priority Ordering
Frame ID Assignment
Search & Optimization
extensible Network Configuration

Figure 7.1: Methodology for building an extensible bus configuration

- *swap the priority of two frames* - Randomly choose two frames. Then swap the priorities of the two frames.

These neighbour moves are applied according to the probabilities listed in table 7.3. All neighbour moves which change the packing configuration are followed by an update-step where timing attributes (period, deadline, jitter) of the affected frames are refreshed.

Neighbour Move	**Probability**
move message into another frame	0.60
move message into new/empty frame	0.16
swap two messages between two frames	0.14
swap priority of two frames	0.10

Table 7.3: Proposed probability of neighbour moves for building extensible bus configurations

After the search has converged towards an optimum, the best obtained bus configuration is returned by the optimization framework. In a final step, the frames priority ordering is mapped to actual frame priority-levels (in case of CAN these are determined by the frame ID). In order to maximize the schedules extensibility, the priority ordering is evenly spread out across the available priority-levels (i.e. the ID range) while maintaining the ordering. E.g.: 64 ordered frames are evenly spread out across 2048 IDs, thus IDs are set in steps of 32, leaving 31 free IDs in between the frames for future usage.

7.3.5 Experimental Evaluation – Extensible Bus Configuration

In order to evaluate the proposed methodology, it is applied to an industrial case study: an automotive in-vehicle network. Based on the communication specification, a bus configuration is built. The results are analyzed and compared to state-of-the-art solving approaches.

Case Study: Vehicle CAN

The communication specification is taken from a *compact executive class (D-segment)* car. The *vehicle CAN* operates at 500 kb/s and uses CAN version 2a (which uses 11 bit frame IDs). 25 processors are connected to the CAN. Some of these processors represent *smart sensors/actuators*. The processors exchange 251 messages (which engineers would pack into 46 frames).

The communication specification can be described by the following statistics: Data size of messages is between 1 and 32 bits. 20% are boolean. 35% represent physical variables, encoded into 8, 12, 16, 24 or 32 bits. This is typical for embedded systems, where data encoding is in alignment with the accuracy of sensors and IOs. 55% are state/status variables, encoded into 2 to 7 bits. Figure 7.2 shows the histogram.

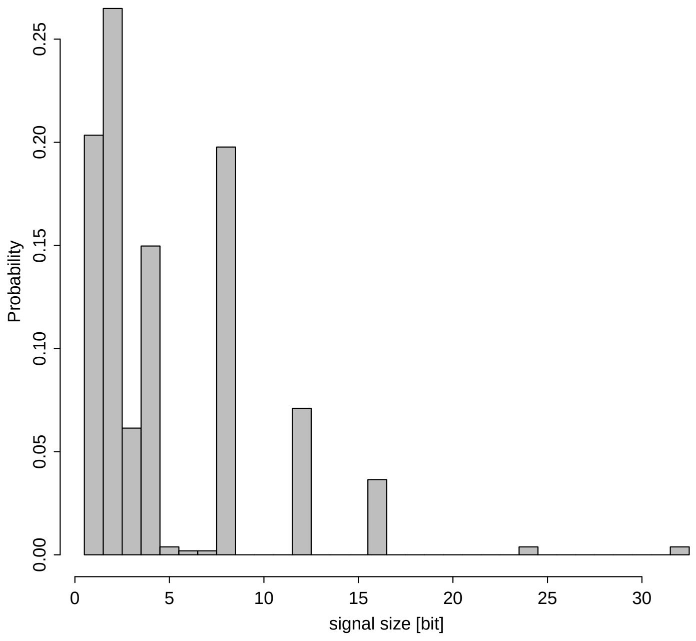


Figure 7.2: Data size of messages for case study

Messages are generated at rates between 10 and 400 ms. Table 7.4 shows that periods are not evenly distributed across this range, but only a set of dedicated periods are used. Messages' deadlines are equal to their periods.

Period	**Probability**
10 ms	0.17
20 ms	0.27
100 ms	0.22
200 ms	0.17
400 ms	0.17

Table 7.4: Period of messages for case study

Results of Optimization Methodology

As outlined earlier, an initial bus configuration is built by applying heuristics. This initial configuration is then refined by applying the optimization strategies. While the initial configuration is generated within seconds, the optimization takes about 2.0 to 3.3 hours. During the optimization, about 30 000 configurations have been evaluated. About 95% of the time is consumed by the schedulability-tests [21]. A significant number of schedulability-test runs are needed for determining the *schedule growth potential* of a bus configuration.

By comparing the initial configuration (which is tailored towards min. bandwidth consumption only) to the optimized configuration, it can be determined by how much the bus configuration can be improved. Table 7.5 summarizes the results.

Metric	**Difference**
min. bandwidth consumption	1.0% deteriation
max. frame growth potential	3.3% improvement
max. schedule growth potential	44.4% improvement
max. robustness against timing-uncertainties	0.7% deteriation

Table 7.5: Improvements achieved by optimization

As expected, the refinement towards *extensibility* comes at a certain cost in *resource efficiency*. The *bandwidth consumption* increases by 1%. At the same time, *robustness against timing-uncertainties* decreases by 0.7%. Balancing this, both *frame* and *schedule growth potential* increase. According to the weights, it was expected to achieve a bigger increase for frames than for the schedule. However, *schedule growth potential* increases significantly. This means that the schedule can incorporate a significant number of additional frames, but only few additional messages inside the existing frames.

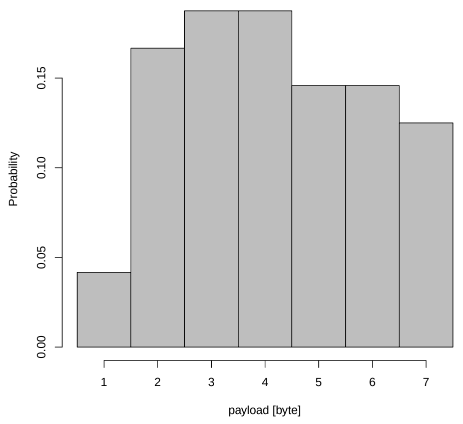
One reason for the unexpected low increase of the frame growth potential may be rooted in the ratio between frame overhead and max. frame payload. For CAN both are 64 bit. Thus, not filling up frames causes a poor payload-to-data ratio, which again increases bandwidth demand.

Metric	**init.**	**opt.**	**Difference**
bandwidth consumption [%]	60.17	61.17	1.0% deter.
frame growth potential	0.703	0.679	3.3% impr.
schedule growth potential	0.552	0.307	44.4% impr.
robustness against timing-uncertainties	0.549	0.553	0.7% impr.

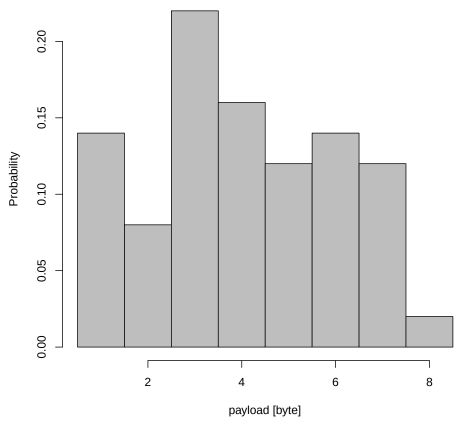
Table 7.6: Initial vs. optimized network configuration

Table 7.6 provides deeper insight into the results. It compares the bus configuration before and after optimization, broken down to the individual metrics. Note that all metrics are normalized between 0 and 1, whereas 0 represents the *best* and 1 represents the *worst*. For bandwidth consumption results are scaled to percent of the bus baudrate.

To provide a deeper insight in how frame growth potential is improved, the frames' payload can be analyzed. Figure 7.3 shows the frame payload histograms: according to the bandwidth-minimizing heuristic, and after refinement by the optimization strategies.



(a) bandwidth-minimizing packing by heuristic



(b) optimized packing by search framework

Figure 7.3: Refinement of packing – impact on frame payload

In the initial configuration frames are quite evenly filled. However, only few frames use 1 byte, and none used 8 bytes. After the optimization the frames fill-level have significantly changed. More frames only use a few bytes (1 to 3). The probability of using more bytes (4 to 7) decreases. This way, there is more free space left inside the frames. Consequently, these frames can incorporate additional messages in the future, thus they are more extensible. Interestingly – since unexpected – some frames even use 8 bytes.

Impact of Initial Packing Heuristic

It is well known that the performance of SA is dependent on the starting position. This is why different heuristics for building the initial network configuration are investigated, and how the initial configuration impacts on the *best obtained solution* found by SA.

Knowing that DMPO is already the optimal solution for scheduling independent frames, the focus is on variations for *frame packing* only.

Bandwidth-minimizing Packing [74]

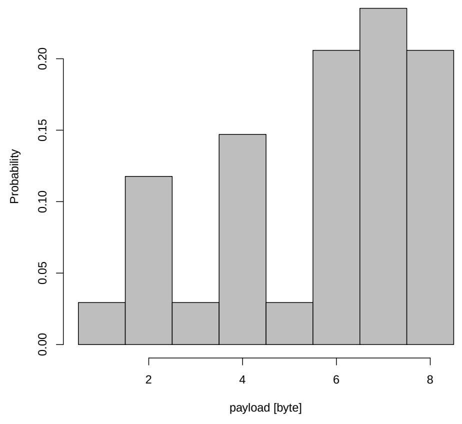
By applying bandwidth-minimizing frame packing according to [74] similar results than the ones discussed above can be achieved.

Metric	**init.**	**opt.**	**Difference**
bandwidth consumption [%]	66.25	67.21	0.95% deter.
frame growth potential	0.953	0.750	21.3% impr.
schedule growth potential	0.618	0.425	31.3% impr.
robustness against timing-uncertainties	0.627	0.643	2.5% deter.

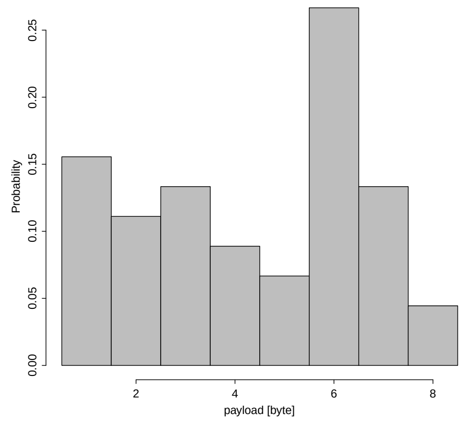
Table 7.7: Initial vs. optimized network configuration

Table 7.7 shows the comparison between the initial and optimized bus configuration. Again, slight deterioration in *bandwidth efficiency* and *robustness* are traded against significant improvements in *extensibility*. However, if comparing the absolute numbers to the previous results, it can be see that [60] outperforms [74].

Figure 7.4 shows the frame payload histograms. It reveals that in the initial packing 65% of the frames are densely packed (using 6 to 8 bytes), leaving only few free space for future growth. After the optimization, this is shifted towards less dense packing, and thus increased growth potential.



(a) bandwidth-minimizing packing by heuristic



(b) optimized packing by search framework

Figure 7.4: Refinement of packing – impact on frame payload

Simple Packing

Simple frame packing is a widely used approach in the literature. It refers to putting each message into a separate frame. Clearly, this way the *frame growth potential* is maximized.

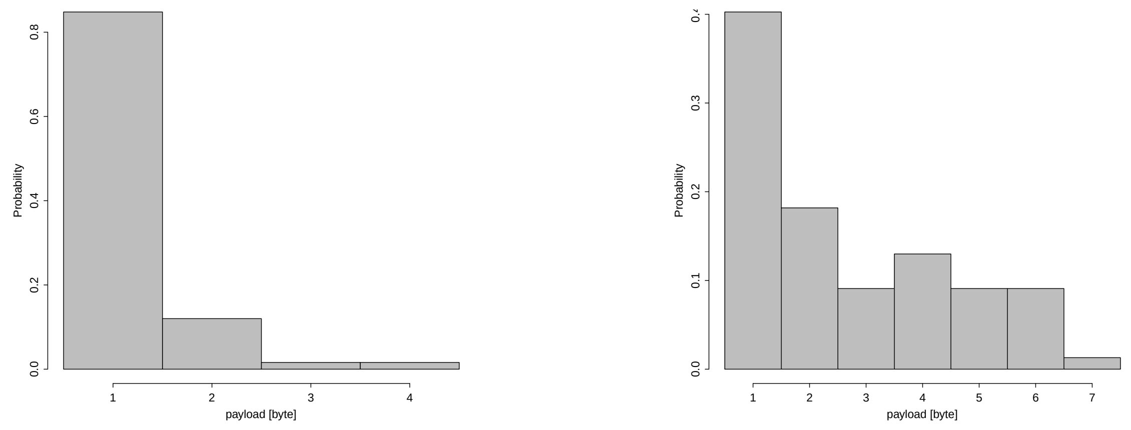
However, this approach features a set of problems: The network is highly utilized (in our case it is even overloaded) and a high number of deadlines are missed.

Metric	**init.**	**opt.**	**Difference**
bandwidth consumption [%]	154.63	60.61	94% impr.
deadlines missed [%]	73.6	3.9	69.7% impr.
frame growth potential	0.141	0.648	361% deter.
schedule growth potential	-	-	-
robustness against timing-uncertainties	-	-	-

Table 7.8: Initial vs. optimized network configuration

Table 7.8 shows the comparison between the initial and the optimized bus configuration. Although SA manages to improve bandwidth consumption, the final bus configuration still misses some deadlines, and thus is infeasible. Since the system is not schedulable, *schedule growth potential* and *robustness* cannot be evaluated. Instead, the worst value (i.e. 1) is assumed.

Figure 7.5 shows the frame payload histograms. Simple packing results in 80% of the frames using only 1 byte. After the optimization, this number drops to 40%. The remaining frames use 2 to 7 bytes.



(a) simple packing by heuristic

(b) optimized packing by search framework

Figure 7.5: Refinement of packing – impact on frame payload

7.3.6 Concluding Remarks

Building a bus configuration which can be utilized for several vehicles (as part of a platform-based design approach) is a challenging engineering task, where conflicting objectives have to be addressed. A two-phased methodology for solving this engineering task was presented in this chapter: An initial bus configuration is build by applying heuristics. This configuration is then refined by applying optimization strategies (based on SA).

Experimental evaluation on an automotive case study reveals the following **lessons learned**: *Bandwidth-minimizing frame packing* according to [60] combined with DMPO is the best performing heuristic for building the initial bus configuration for a priority-based bus-system. This configuration represents a reasonably good starting point for the subsequent optimization-phase. *Simple frame packing* is a poor choice. It produces infeasible configurations, which even the optimization-phase cannot fix.

7.4 Extending Bus Configuration under Backward Compatibility Constraints

The previous section 7.3 presented a methodology for building an extensible and robust bus configuration. This engineering task is a key step in a platform-based design approach where the platform will be used by several vehicle-projects.

Now, the second aspect of the platform-based design approach is addressed: Finding a methodology for *extending / upgrading the bus configuration* in order to meet the additional requirements of a specific vehicle-project. Thereby the specific bus configuration must *stay backward-compatible* to the platform bus configuration.

The need to stay backward-compatible to the platform bus configuration (as much as possible) is rooted in the development process between the OEM and its Tiers. For any change-request both parties need to perform an impact analysis, re-negotiation of the network interface, and re-verification of the system's behaviour. Thus, changes are associated with significant time and cost issues, and shall therefore be avoided. In [30] the development process for maintaining and extending / upgrading a bus configuration is presented, as it is performed at BMW.

7.4.1 Scenarios & Definitions

Based on the scenarios from table 7.1 a sub-set of scenarios are derived which have a direct impact on bus configuration. In alignment with discussion with industrial partners, table 7.9 shows how these **change-scenarios** can be addressed. The two main changes are to *add new messages* (M-1) and to *add new frames* (F-1). Thus these scenarios are the key focus of the proposed methodology.

The following **definitions** apply when extending / upgrading a bus configuration:

- A frame packing P^n is *backward-compatible* to a frame packing P^o if no message from P^n is packed into a different frame than it was packed in P^o. Thus, existing messages' packing must not be modified. Messages which are newly added can either be packed into new frames or also into existing frames (as long as this does not lead to problems concerning the backward-compatibility at the schedule-level).

- A frame schedule S^n is *backward-compatible* to a frame schedule S^o if no frame from S^n is assigned a different resource than it was assigned in S^o. Thus, existing frames must not be modified, and frames which are newly added can only be assigned to resources which were not used in S^o. For a time-triggered schedule the resource is a time-slot, for a priority-based schedule the resource is a priority-level.

- A frame packing P is *extensible* if additional messages can be added to the frame, or if its packed-in messages' data size can be increased in the future. Extensibility at the frame-level can be measured by the free payload capacity, according to equation (7.1).

	Change	Approach
P-1	add new processor	include processor's outgoing frames into the bus schedule (see F-1)
M-1	add new message	pack message into a suitable frame
M-2	modify data size of message	check if message still fits associated frame's max. payload; if not, pack into another frame
M-3	modify period / deadline of message	adjust period / deadline of associated frame accordingly (see F-2)
M-4	modify receiver-processors of message	adjust receiver-processors of associated frame accordingly (see F-3)
F-1	add new frame	include frame into the bus schedule
F-2	modify period / deadline of frame	check schedulability of frames and packed-in messages; for time-triggered bus-systems, check if frame still fits into associated time-slots
F-3	modify receiver-processors of frame	no impact on bus configuration since bus-systems are broadcasting frames

Table 7.9: Scenarios for extending bus configuration

- A frame schedule S is *extensible* if additional frames can be added to the schedule in the future. Extensibility at the schedule-level can be measured by the number of frames which can be added, according to equation (7.3) and (7.4).

7.4.2 Problem Statement

Based on the change-scenarios from table 7.9, the engineering challenge at hand is to *extend / upgrade a bus configuration* while *maintaining backward-compatibility.* More formally, it is stated as follows:

> Given a bus configuration C^o consisting of a set of frames $\mathcal{F}^o = \{f_1^o, f_2^o, \ldots, f_j^o\}$ which are scheduled according to a schedule S^o. Each frame contains a set of packed-in messages $\mathcal{M}^o = \{m_1^o, m_2^o, \ldots, m_k^o\}$. Further given a set of messages $\mathcal{M}^n = \{m_1^n, m_2^n, \ldots, m_q^n\}$ which must be added to the given bus configuration C^o. Find an extended bus configuration C^n in which the newly added messages are incorporated (i.e. the new messages are packed into frames). This bus configuration C^n is subject to the constraints that it (a) must be feasible / schedulable and (b) must be backward-compatible to the given bus configuration C^o. The optimization goal is that the bus configuration C^n is (c) optimized towards maximum extensibility.

The problem consists of two sub-problems: packing of messages into frames and scheduling of frames. Both sub-problems have to be addressed under the stated constraints and optimization goals.

7.4.3 Literature Review Refinement – Extend Bus Configuration

As already reviewed in section 5.4, 6.2 and 7.3.2, there are only a few works in the literature which address *frame packing*. Although they differ in optimization goal and solving strategy, all of them share one common scenario: build a packing "from scratch". None of them focuses on *extending / upgrading a packing*, where legacy packing decisions need to be taken into account. Some constraints have been introduced in chapter 6.

The second aspect of bus configuration is *frame scheduling*. Focusing on CAN (as it is the most widely used bus protocol in the automotive domain) the task is *frame priority assignment*. The literature on priority assignment is extensive. However, in terms of finding a schedulable priority assignment, the best known approach is Audsley's optimal priority ordering algorithm [1,3]. Audsley proved that this algorithm will find a schedulable priority ordering, if a feasible solution exists. Later work [9] proved that the algorithm also holds in the presents of blocking. This is essential, as CAN is non-preemptive. In section 7.3 the aspect of *extensibility* has been introduced. However, the scenario was still: build a schedule "from scratch". No work can be found which focuses on *extend / upgrade a schedule*.

7.4.4 Solving Methodology – Extend Frame Packing

Based on the **problem statement** from above, the following sub-problem can be stated:

> Given a set of frames $\mathcal{F}^o = \{f_1^o, f_2^o, \ldots, f_j^o\}$ each containing a set of packed-in messages. Further given a set of messages $\mathcal{M}^n = \{m_1^n, m_2^n, \ldots, m_q^n\}$ which need to be added to the bus configuration. The new messages need to be packed in a way that the extended packing P^n is backward-compatible to the given packing P^o.

Strategy

The key to solve this problem is to re-visit section 5.5.1:

> There exists a frame that already has some messages packed-in. Another message needs to be packed-in and it can fit into the existing frame. The question is: Should the message be packed into the existing frame (thus extending it), or should the message be packed into a new frame?

However, this time it has to be addressed under the additional constraint of backward-compatibility. Clearly, if all new messages are packed into new frames (i.e. no existing frame is extended) then the extended packing P^n is backward-compatible to the given packing P^o. However, in terms of bandwidth-demand this option may be a poor choice, as additional overhead is introduced and free frame capacity is not utilized.

Thus, some new messages should be packed into existing frames, in order to increase bandwidth-efficiency. The question is: into which frames can a new message be packed, so that the packing P^n is still backward-compatible to P^o? Beside the obvious constraints (i.e. frame must have sufficient free capacity for incorporating the message, and both originate from the same processor) the key decision-attribute for determining a frame is the period.

T-1: *message has same period as frame* – If the message has the same period as the frame, then the frame's period does not change if the message is added to the frame. This ensures that the frame can be scheduled the same way it was in the given schedule S^o. This is essential for a TDMA schedule.

T-2: *message's period is a multiple of frame's period* – If the message's period is a multiple of the frame's period, then the frame's period does not change if the message is added to the frame (as in case T-1). However, the message is sent more frequently as it would be needed, thus additional bandwidth-demand is caused.

T-3: *message's period is not a multiple of frame's period* – There exist two sub-cases.

T-3a: If the message's period is longer than the frame's period, then the frame's period will stay as it is and thus no changes occur to the schedule (as in case T-2). Also, the message will be sent more frequently than needed.

T-3b: If the message's period is shorter than the frame's period, then the frame's period will change if the message is added to the frame. For TDMA this will cause incompatibility to the given schedule S^o. For CAN this period change may be acceptable, as long as frame's priorities are not changed and schedulability can still be guaranteed.

By considering these options, there may still be several frames into which a new message can be packed. In this case, the secondary decision-criteria should be the resulting bandwidth-demand (as optimized in section 5.5).

Algorithm

Based on the considerations stated above, a frame packing algorithm can be derived which ensures backward-compatibility. Algorithm 7.1 solved the stated problem, and is applicable independent of the underlying frame scheduling schema.

The algorithm essentially searches for packing candidates (i.e. frames) which satisfy all constraints, especially those associated with backward-compatibility. Out of these candidates, the one is chosen which offers the best performance in terms of bandwidth-efficiency.

7.4.5 Solving Methodology – Extend Frame Schedule

The second aspect of bus configuration is frame scheduling. Based on the **problem statement** from above, and focusing on CAN (as it is the most widely used bus protocol in the automotive domain) the following sub-problem can be stated:

> Given a set of frames $\mathcal{F}^o = \{f_1^o, f_2^o, \ldots, f_j^o\}$ each assigned a priority-level p_j^o forming a schedule S^o. Further given a set of additional frame $\mathcal{F}^n = \{f_1^n, f_2^n, \ldots, f_q^n\}$ which need to be added to the schedule. Find a schedule S^n by assigning a priority-level p_q^n to the new frames so that S^n is (a) feasible / schedulable and (b) backward-compatible to the given schedule S^o. The optimization goal is that the extended schedule S^n is (c) optimized towards maximum extensibility.

Algorithm 7.1: Frame packing under backward-compatibility constraints

```
Input: frames /* with packed-in messages */
Input: messages /* which need to be packed */
begin ExtendFramePacking
    /* prepare for frame packing */;
    sort(messages, T, increasing) /* sort messages by T [0..n] */;
    sort(frames, T, increasing) /* sort frames by T [0..n] */;
    foreach message do
        /* basic constraints */;
        frames = filter by basic constraints;
        /* backward-compatibility constraints */;
        frames = filter by $T_m = T_f$ or $T_m$ is multiple of $T_f$;
        /* pack message */;
        if frames $\neq$ empty then
            frame = frame with lowest bandwidth-demand increase;
        else
            frame = new frame;
        end
        pack(message, frame);
    end
end
Output: frames
```

Problem Complexity

The frame scheduling problem (like any priority-based scheduling problem) contains two aspects: First, to find a feasible priority-ordering. Second, to map the priority-ordering to dedicated priority-levels.

Audsley [3] showed that the problem for priority-ordering is NP-hard. There are $n!$ possible priority-orderings for a set of n tasks. He also showed that there is an optimal algorithm which finds a feasible priority-ordering (if one exists) within $(n^2 + n)/2$ steps.

Once a feasible priority-ordering is found, it has to be mapped to the available priority-levels. There are $\binom{p}{n}$ possible mappings of n frames to p priority-levels. However, no work can be found in the literature which tackles this problem. Instead, a 1:1 mapping is applied. Clearly, both sub-problems are intractable for realistically sized systems.

These complexities apply to the *un-constrained* case (i.e. no backward-compatibility constraints exist). In the *constrained* case (i.e. new frames need to be added to an existing schedule) the following complexities apply:

$$\text{priority-orderings} : n! \tag{7.6}$$

$$\text{priority-levels} : \binom{p-o}{n} \tag{7.7}$$

where n is the number of new frames, p is the number of available priority-levels, and o is the number of frames in the given schedule. Again, the sub-problems are intractable for

Algorithm 7.2: Priority assignment under backward-compatibility constraints

```
   Input: old frames /* with given priorities */
   Input: new frames /* without priorities yet */
   Input: threshold for rel.R /* max.  for R/D */
1  begin PriorityAssignmentForNewFrames
2      /* prepare for priority assignment */;
3      each new frame gets a temporary priority, higher than any old frame;
4      set $U$ := un-assigned new frames;
5      put all new frames in set $U$;
6      /* perform priority assignment */;
7      $p$ = lowest free priority-level;
8      repeat
9          repeat
10             $f$ = pick a new frame from set $U$;
11             assign priority $p$ to new frame $f$;
12             if feasible($f$ , $p$ , thresh) then
13                 found frame at priority-level = true;
14                 mark priority $p$ as used;
15                 remove new frame $f$ from set $U$;
16             else
17                 found frame at priority-level = false;
18                 assign prio=highest to frame $f$ again;
19                 /* try next frame */;
20             end
21         until found frame at priority-level = true
           or all $f$ in $U$ have been evaluated at $p$;
22         $p$ = next-higher free priority-level;
23     until $U$ is empty or no more free priority-levels;
24 end
   Output: all frames /* with priorities */
```

realistically sized systems.

The state-of-the-art scheduling problem is a special case of this problem, where the number of pre-existing frames o is zero, and the number of priority-levels p equals the number of frames n.

Algorithm

To solve this challenging engineering task, a priority assignment policy as outlined in algorithm 7.2 is proposed. It is based on the key concepts of Audsley's algorithm for optimal priority ordering [3]. However, unlike Audsley's algorithm, it does not deal with priority-orderings, but with unique priority-levels. Further, it honours the constraints of backward-compatibility, and therefore will only consider the non-used priority-levels.

The **algorithm** starts by constructing an initial, temporary priority assignment, where all old frames keep their priority and all new frames have a priority higher than any old frame. This initial assignment may be infeasible. Then, step by step, the new frames are assigned to their final priority-levels. For each free priority-level p (starting at the lowest priority-level) the algorithm searches for a new frame f which is *feasible* at the dedicated priority-level. In this context, feasible is defined as: The new frame f is schedulable at the priority-level p and all lower-priority frames (both old and new) are schedulable[1]. In the worst case, all new frames (which have not been assigned a priority-level yet) will be evaluated at the priority-level p. If no new frame is feasible at the dedicated priority-level, the algorithm moves up to the next-higher free priority-level. If a new frame is feasible at a dedicated priority-level, the algorithm fixes its priority-level there, marks the priority-level as *used* and marks the frame as *assigned*. The entire procedure is repeated, until (a) all new frames are marked as *assigned*, in which case a feasible priority assignment has been found, or (b) we run out of free priority-levels, in which case the frame-set is unschedulable.

Figure 7.6 demonstrates the algorithm, applied to a small example. Each iteration-step is represented by one column (reading from left to right).

Frame	**B**	**C**	**D**	**T**	**Priority**
o_1	1	1	2	10	given
o_2	1	1	4	10	given
n_1	1	1	5	5	find it
n_2	1	1	5	10	find it

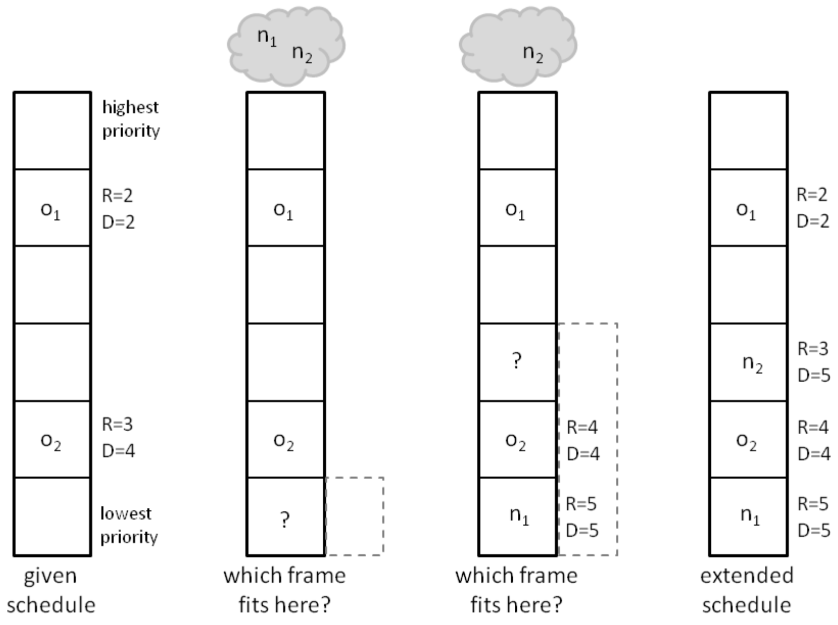


Figure 7.6: Extending a CAN schedule by applying algorithm 7.2

The **runtime complexity** of algorithm 7.2 is $O(p \cdot n \cdot S)$ where p is the number of available priority-levels, n is the number of newly added frames, and S is the complexity

[1]The term *threshold* can be ignored for now. It will be introduced later.

of the schedulability-test [21]. In practice, it is observed that most problem-instances are solved much faster than this worst-case complexity suggests. For typical CAN systems, the number of priority-levels p which are evaluated by the algorithm before a feasible schedule is found is approximately the number of newly added frames n. Thus, the average-case algorithm complexity is $O(n^2 \cdot S)$.

Proof of Optimality

In this work, Audsley's definition of optimality is used [1]: If there is a feasible priority assignment, the proposed algorithm will find it. In [1] it is proven that a similar algorithm to the one presented here is optimal. The main differences are that Audsley is dealing with tasks rather than frames, CAN scheduling policy is non-preemptive, and that the proposed algorithm must deal with some *old* frames that are un-moveable in terms of their priority (or frame ID). The first of these differences is clearly not important as it is the time the tasks or frames occupy resources that is important. For the second difference, Bletsas [9] showed that the introduction of blocking does not invalidate the optimality of the algorithm. The third difference is effectively a fundamental change to the assumptions under which the algorithm is applied. To determine whether the algorithm and proof are still valid, the following corollaries underpinning the proof (taken from Audsley [1]) are examined. These corollaries are all dependent on the scheduling approach and the analysis used. Importantly they are not dependent on how the frame-set has its attributes assigned. Therefore, the constraints imposed on how priorities are generated should not have an effect.

- *Corollary 1* – The response time of a task is dependent upon the set of higher priority tasks, but not the relative priority ordering of those tasks
- *Corollary 2* – The response time of a task is not dependent upon lower priority tasks
- *Corollary 3* – The response time of a task cannot increase when assigned a higher priority
- *Corollary 4* – The response time of a task cannot decrease when assigned a lower priority

Given the continued validity of the corollaries, the remaining issue is the following Lemma.

Lemma 1. *Having un-moveable frames does not affect the optimality of the algorithm.*

Proof. The basis of Bletsas's proof was that swapping priority orders means the task (or frame) that has its priority increased can only have its response time remain the same or be reduced. The algorithm still performs swapping of priorities in the same fashion as the original Audsley's algorithm used by Bletsas, however the main difference is the priorities are not swapped between the frames. Instead the frame that is unschedulable has its priority raised above the frame that cannot be changed. This difference does not invalidate the original proof. □

If the proposed algorithm fails then there is no feasible schedule. In that case the only option is to allow violating the backward-compatibility constraints (i.e. to allow changing the priority of some old frames). However, since backward-compatibility is no longer achieved then, this is a delicate decision for industrialists to make. The algorithm will still be optimal if the industrial designers remove some of the constraints, concerning un-moveable frame priorities, until the network becomes schedulable.

Optimization towards Extensibility

By applying algorithm 7.2 a given schedule can be extended by a set of additional frames while staying backward-compatible to the given schedule. However, analysis of the frame-set reveals that in terms of *extensibility* it does not perform too well. Since the new frames are assigned the lowest priority-level at which they are schedulable, their response times may be quite close to their deadline. As a consequence, if this schedule needs to be extended in the future, these frames are likely to miss their deadline then.

In order to overcome this issue and to *build an extensible schedule* the key is to ensure that the schedule contains *extensibility margins*. For a TDMA schedule these are empty time-slots. For a priority-based schedule (such as CAN) this means that the response time of each frame is significantly smaller than its deadline. For example, if the response time of a frame is 70% of its deadline then the remaining 30% act as extensibility margins for future extensions. The smaller the response time, the higher the extensibility margins. Thus, extensibility of a CAN schedule can be estimated by the relative response time $rel.R$ of the frames.

$$rel.R = \frac{R}{D} \tag{7.8}$$

$$ext_f = \frac{1}{rel.R_f} = \frac{D_f}{R_f} \tag{7.9}$$

$$ext_{CAN} = \frac{D_f}{R_f} \quad {}_{\forall f \in \mathcal{F}} \tag{7.10}$$

In order to maximize the extensibility of the CAN schedule, the relative response time of the frames must be minimized. Thereby the frame which has the largest relative response time determines the extensibility of the schedule.

Based on these considerations and the metric for extensibility, as given by equation (7.10), a revised methodology can be derived. It incorporates the following concepts:

- re-definition of *feasible* – A priority assignment is feasible if the frame's relative response time (at a certain priority-level) is smaller than a defined threshold, and all lower-priority frames' relative responses are smaller than that same threshold. Thus, instead of checking $R \leq D$ we check $R/D \leq thresh$. Obviously the threshold must not exceed 1.0 in order to have a schedulable system.

- By setting a threshold (e.g. $thresh = 0.7$) algorithm 7.2 builds a backward-compatible schedule, in which no frame's relative response time exceeds the defined threshold. Thus, this backward-compatible schedule has a guaranteed degree of extensibility.

- By lowering the threshold the degree of extensibility is increased. Based on this relation, the schedule's extensibility can be optimized (i.e. maximized).

Algorithm 7.3: Extensibility-optimized priority assignment under backward-compatibility constraints

```
   Input: old frames /* with given priorities */
   Input: new frames /* without priorities yet */
 1 begin ExtensiblePriorityAssignmentForNewFrames
 2     /* extensibility → max. */;
 3     /* binary search for min.  R/D */;
 4     rel.R bounds = 0.0 and 1.0;
 5     repeat
 6         rel.R = (rel.R upper + rel.R lower) / 2;
 7         PriorityAssignmentForNewFrames(old frames, new frames, rel.R) /* see
           algo. 7.2 */;
 8         if priority assignment = feasible then
 9             rel.R upper = rel.R /* lower upper */;
10         else
11             rel.R lower = rel.R /* raise lower */;
12         end
13     until feasible and Δ rel.R < x%;
14 end
   Output: all frames /* with priorities */
```

Algorithm 7.3 incorporates these concepts, thus it can *extend / upgrade a given schedule* so that the schedule (a) is feasible / schedulable, (b) is backward-compatible to the given schedule, and (c) is maximized towards extensibility. The algorithm uses a binary-search for finding the lowest possible threshold for relative response times, whereas the priority assignment is performed by algorithm 7.2 accordingly.

By introducing extensibility the overall nature of algorithm 7.2 does not change other than to effectively reduce the deadline of frames (instead of demanding $R \leq D$ we demand $R/D \leq thresh$). Thus, the **proof of optimality** still holds. However, it leads to a different definition of optimality: If there is a priority assignment that satisfies $R/D \leq thresh$ for all frames, then the proposed algorithm will find it.

By introducing extensibility the **runtime complexity** of algorithm 7.2 does not change. The complexity of algorithm 7.3 (i.e. the binary-search for the max. extensibility) is $O(A \cdot \log_2 \frac{100}{x})$ where A is the complexity of the priority assignment algorithm 7.2 and the jitter of the min. threshold is $\pm x\%$. The term $\log_2 k$ stems from the floating-point binary-search. The term $\frac{100}{x}$ represents that the algorithm ends the search if the threshold-variation is less than $\pm x\%$.

7.4.6 Experimental Evaluation – Extend Bus Configuration

In order to evaluate the proposed methodology, it is applied to an automotive use case. The bus-system is installed in a *compact executive class* car. The system consists of a CAN network operated at 500 kb/s. The CAN network is connecting 20 processing-units. These processing-units exchange 34 frames, which causes a network utilization of 40%. The schedule of the network (i.e. the priorities of the frames) is given by engineers. This schedule shall be extended by additional 31 frames, which cause additional 33% network utilization. For these frames a priority assignment was found, which (a) is feasible, (b) is backward-compatible to the given schedule, and (c) is extensible.

Approach	**Schedulable**	**max. R/D**	**avg. R/D**	**Backward-Compatible**
Algorithm 7.2	yes	0.98	0.32	**yes**
Algorithm 7.3	yes	0.60	0.24	**yes**
DMPO	yes	0.60	0.23	no
Audsley's Algorithm	yes	0.60	0.23	no

Table 7.10: Extend CAN schedule under backward-compatibility constraints

Table 7.10 shows the results by applying different algorithms. The results clearly demonstrate the effectiveness and efficiency of the proposed methodology (especially algorithm 7.3). In terms of extensibility (indicated by the relative response times) algorithm 7.3 performs almost identical to state-of-the-art approaches (such as DMPO or Audsley's algorithm). However, state-of-the-art approaches cannot guarantee backward-compatibility, and thus cannot satisfy the needs to the problem at hand. Algorithm 7.2 and 7.3 guarantee backward-compatibility, whereas algorithm 7.3 is the overall best performing one.

7.5 Concluding Remarks

Designing and maintaining the in-vehicle communication (i.e. the configuration of bus-systems) is a key challenge for the OEM, especially when using a platform-based approach. In this chapter, both aspects of this challenging engineering task have been addressed: Section 7.3 shows how an *extensible* configuration can be built, which can be used as the platform configuration. Section 7.4 shows how the platform configuration can be *upgraded* in order to meet new requirements, while *staying backward-compatible.*

Experimental evaluation on an automotive case study shows that both methodologies can solve the associated problem effectively and efficiently. In addition, algorithm 7.2 is proven to be optimal according to Audsley's definition.

7.6 Outlook

For the sub-problem *frame packing* both aspects have been addressed: designing and maintaining the configuration. For the sub-problem *frame scheduling* also both aspects have been addressed for CAN. Future work should tackle TDMA-based schema as well.

Chapter 8

Conclusion

Designing distributed embedded real-time systems is a challenging engineering task. A set of decisions have to be made in order to find an optimal system configuration. The goal of this thesis is to provide novel solutions for some of these decision-making steps: allocating software components to processors, configuring the communication between processors via bus-systems, scheduling of tasks and bus frames.

8.1 Achievements

Within this thesis, the following improvements beyond the state-of-the-art have been achieved:

- *communication configuration* – In this thesis an enhanced model for the communication between processors via shared bus systems is applied. In this model, several messages can be packed into a single bus frame in order to increase bandwidth efficiency. This model is commonly used in automotive systems.
  As a consequence the question arises: *how shall messages be packed into frames in an optimal way*? First, metrics for optimality are provided. Then, a bandwidth-minimizing frame packing heuristic is developed. Evaluation reveals improved bandwidth efficiency of up to 18.9%, as shown in chapter 5. In addition, the heuristic improves the search-performance of the TAP significantly.

- *constraint handling* – In chapter 6 a detailed constraint-model is introduced. It is aligned to the AUTOSAR standard (namely the system template). The model enhances the state-of-the-art with respect to constraints for communication.
  The question is: *how can the set of heterogeneous design-constraints be satisfied efficiently*? Analysis of the constraint-model leads to a methodology how a sub-set of the constraints can be resolved, thus excluding infeasible regions of the design-space and hence boosting design-space exploration. The methodology allows tackling TAP with legacy-constraints, which is widespread for industrial systems.

- *extensible configuration* – When designing a system, it shall be robust and extensible towards future changes, i.e. it shall be possible to add elements later on without needing to re-design the entire configuration. This is especially true for the bus

configuration.
The question is: *how to balance robustness and extensibility against the need for efficient resource-usage*? Section 7.3 first provides enhanced metrics for extensibility. Then, a methodology is developed which finds a bus configuration that (a) minimizes the bandwidth demand, and (b) maximizes the robustness and extensibility. By applying this methodology, additional messages can be added to the bus configuration in the future, while keeping the change-impact at a minimum.

- *upgrade configuration* – When maintaining and upgrading a system (e.g. by adding messages), a key requirement is to maintain backward-compatibility. Thus, the question arises: *how can the messages be merged into the bus configuration, so that the new configuration stays backward-compatible to the initial one*?
  Section 7.4 first establishes criteria for backward-compatibility for frame packing and scheduling. Then, two algorithms are provided: upgrading frame packing, and upgrading a CAN schedule. Finally, proof of optimality is given. The algorithms allows for an evolutionary system design, where a "new" configuration is based on an older configuration.

8.2 Open Issues

Although most aspects of this thesis are inspired by problems originating from the automotive domain, it was a goal to provide methodologies which are generally applicable. Therefore, not all details from automotive systems have been considered.

- An AUTOSAR software component can contain 1 or more runnables (where the runnables represent the executable code). Thus, in order to schedule the runnables, they have to be assigned to OS tasks, and the OS task needs to be assigned a priority. In the thesis it is assumed that a software component only contains 1 runnable (which often is the case). Thus a 1:1:1 relationship between software component, runnable and task is used.
- In AUTOSAR, data (also called signals) which is exchanged between software components can be nested into other artefacts. First, the signal can be put into an I-PDU. Then, the I-PDU is put into a N-PDU, and finally the N-PDU is packed into a bus frame. In this thesis, a simpler model is used: a message is packed into a frame.
- AUTOSAR allows different kind of communication schema between software components: sender/receiver, client/server, calibration, AUTOSAR-service. In this thesis, only the sender/receiver communication semantic is used.

By taking into account these domain-specific issues, the applicability of the provided methodologies in an automotive scenario would be improved.

8.3 Future Research Directions

In recent years, *multiprocessing* has gained significant interest due to the availability of affordable *multi-core* processors. These are already wide-spread for general-purpose computing, and are currently becoming available for embedded computing. Within that context, the question of how to allocate tasks to processors needs to be refined to: *How to allocate tasks to processor-cores?*

In 2012, a new safety-standard [35] was introduced into the automotive domain. In this context, each component of the system has to be assigned a safety integrity level (A to D). In most systems, different components will have different levels. Such systems are called *mixed-criticality* systems. Within this context, the system-configuration decisions need to be re-visited: How can task-allocation, network-configuration and scheduling be designed while considering the mixed-criticality nature of the system?

As the number of functionality within a car is ever increasing, the communication demand between processors also increases. In order to meet future demands, new high-bandwidth bus-systems need to be used. One potential candidate is *automotive Ethernet* [31]. Hence, the proposed methodology for *bus-configuration* needs to be adapted to protocol-specifics of these new bus-systems.

In this thesis a normalized weighted sum is used to balance the multiple optimization objectives. It might be interesting to extend the framework so that it not only returns a single "best" solution, but a set of *pareto-optimal solutions.* This would give the user more flexibility, and remove the need to set weights.

List of Algorithms

List of Figures

List of Tables

References

[1] N. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

[2] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[3] N. Audsley, K. Tindell, and A. Burns. The end of the line for static cyclic scheduling? In *Euromicro Workshop on Real-Time Systems (ECRTS)*, pages 36–41, 1993.

[4] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard real-time scheduling: The deadline-monotonic approach. In *IEEE Workshop on Real-Time Operating Systems and Software*, pages 133–137, 1991.

[5] AUTOSAR (automotive open system architecture). `http://www.autosar.org`.

[6] J. Axelsson. Three search strategies for architecture synthesis and partitioning of real-time systems. Technical report number R-96-32, Department of Computer and Information Science, Linköping University, Sweden, 1996.

[7] I. Bate and N. C. Audsley. Flexible design of complex high-integrity systems using trade offs. In *IEEE International Symposium on High-Assurance Systems Engineering (HASE)*, pages 22–31, 2004.

[8] I. Bate and P. Emberson. Incorporating scenarios and heuristics to improve flexibility in real-time embedded systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 221–230, 2006.

[9] K. Bletsas and N. Audsley. Optimal priority assignment in the presence of blocking. *Information Processing Letters*, 99(3):83–86, 2006.

[10] S. Brummund, N. Kehl, P. Nenninger, and U. Kiencke. ISODATA clustering for optimized software allocation in distributed automotive electronic systems. In *SAE 2006 Transactions Journal of Passenger Cars: Electronic and Electrical Systems*, 2006.

[11] S. Brummund, C. Steup, and U. Kiencke. Real multi-partitioning for optimized distributing and allocating software in vehicle networks. In *SAE 2007 Transactions Journal of Passenger Cars: Electronic and Electrical Systems*, 2007.

[12] G. C. Buttazzo. Rate monotonic vs. EDF: Judgment day. *Real-Time Systems*, 29(1):5–26, 2005.

[13] D. Buttle. Real-time in the prime-time. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2012. Keynote.

[14] ISO 11898-1: Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling.

[15] S. T. Cheng and A. K. Agrawala. Allocation and scheduling of real-time periodic tasks with relative timing constraints. Technical report number CS-TR-3402, Department of Computer Science, University of Maryland, 1995.

[16] W. W. Chu and M.-T. Lan. Task allocation and precedence relations for distributed real-time systems. *IEEE Transactions on Computers*, 36(6):667–679, 1987.

[17] E. G. Coffman, J. Csirik, and G. J. Woeginger. Approximate solutions to bin packing problems. Technical report, Graz University of Technology, Institute for Mathematics B, 1999.

[18] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In *Approximation algorithms for NP-hard problems*, chapter 2, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1996.

[19] R. Davis and A. Burns. Response time upper bounds for fixed priority real-time systems. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 407–418, 2008.

[20] R. Davis and A. Burns. Robust priority assignment for messages on controller area network (CAN). *Real-Time Systems*, 41(2):152–180, 2009.

[21] R. Davis, A. Burns, R. Bril, and J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[22] R. Davis, A. Zabos, and A. Burns. Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Transactions on Computers*, 57(9):1261–1276, 2008.

[23] R. Dick, D. Rhodes, and W. Wolf. TGFF: task graphs for free. In *Hardware/Software Codesign (CODES/CASHE)*, pages 97–101, 1998.

[24] M. Driussi and F. Pölzlbauer. Towards an AUTOSAR system configuration framework. *ATZextra worldwide*, 18(9):26–28, 2013.

[25] C. Ekelin and J. Jonsson. Solving embedded system scheduling problems using constraint programming. Technical report, Chalmers University of Technology, Department of Computer Engineering, 2000.

[26] P. Emberson. *Searching For Flexible Solutions To Task Allocation Problems.* PhD thesis, University of York, Department of Computer Science, Nov. 2009.

[27] P. Emberson and I. Bate. Stressing search with scenarios for flexible solutions to real-time task allocation problems. *IEEE Transactions on Software Engineering*, 36(5):704–718, 2010.

[28] FlexRay. `http://www.flexray.com`.

[29] D. Gu, F. Drews, and L. Welch. Robust task allocation for dynamic distributed real-time systems subject to multiple environmental parameters. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 675–684, 2005.

[30] D. Gunnarsson and M. Traub. Approach for a seamless timing evaluation process for E/E-architectures. In *6th Symtavision News Conference*, 2012.

[31] P. Hank, S. Müller, O. Vermesan, and J. Van den Keybus. Automotive ethernet: In-vehicle networking and smart mobility. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1735–1739, 2013.

[32] P.-E. Hladik, A. Déplanche, S. Faucou, and Y. Trinquet. Adequacy between AUTOSAR OS specification and real-time scheduling theory. In *IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 225–233, 2007.

[33] P.-E. Hladik, A. Déplanche, S. Faucou, and Y. Trinquet. Schedulability analysis of OSEK/VDX applications. In *International Conference on Real-Time and Network Systems (RTNS)*, pages 131–139, 2007.

[34] M. Holenderski, M. M. van den Heuvel, R. J. Bril, and J. J. Lukkien. Grasp: Tracing, visualizing and measuring the behavior of real-time systems. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2010.

[35] ISO 26262: Road vehicles – Functional safety.

[36] M. Kang, K. Park, and M. Jeong. Frame packing for minimizing the bandwidth consumption of the FlexRay static segment. *IEEE Transactions on Industrial Electronics*, 60(9):4001–4008, 2013.

[37] O. Kelly, H. Aydin, and B. Zhao. On partitioned scheduling of fixed-priority mixed-criticality task sets. In *International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1051–1059, 2011.

[38] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[39] B. Knauder, F. Pölzlbauer, and J. Zehetner. Modellierung von Informationskanälen für den Einsatz in Simulationsumgebungen. In *SIMVEC – Berechnung, Simulation und Erprobung im Fahrzeugbau*, number VDI-Berichte 2169, pages 377–388, 2012.

[40] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, 1997.

[41] H. Kopetz and R. Nossal. The cluster compiler – a tool for the design of time-triggered real-time systems. In *ACM SIGPLAN 1995 workshop on languages, compilers & tools for real-time systems*, pages 108–116, 1995.

[42] M. Lewis, B. Alidaee, and G. Kochenberger. Using xQx to model and solve the uncapacitated task allocation problem. *Operations Research Letters*, 33(2):176–182, 2005.

[43] LIN (local interconnect network). `http://www.lin-subbus.org`.

[44] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computer Machinery*, 20(1):46–61, 1973.

[45] H. Lonn and J. Axelsson. A comparison of fixed-priority and static cyclic scheduling for distributed automotive control applications. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 142–149, 1999.

[46] S. Lui, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[47] D. L. Massart, J. Smeyers-Verbeke, X. Caprona, and K. Schlesierb. Visual presentation of data by means of box plots. *Europe*, 18(4):215–218, 2005.

[48] MOST (media oriented systems transport). `http://www.mostnet.de`.

[49] P. Narasimhan, L. Moser, and P. Melliar-Smith. Message packing as a performance enhancement strategy with application to the totem protocols. In *Global Telecommunications Conference (GLOBECOM), Communications: The Key to Global Prosperity*, pages 649–653, 1996.

[50] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert. Trends in automotive communication systems. *Proceedings of the IEEE*, 93(26):1204–1223, 2005.

[51] M. C. Necker, M. Köhn, A. Reifert, J. Scharf, and J. Sommer. Optimized frame packing for OFDMA systems. In *IEEE Vehicular Technology Conference (VTC)*, pages 1483–1488, 2008.

[52] OSEK/VDX. `http://www.osek-vdx.org`.

[53] J. Palencia and M. Gonzalez Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 26–37, 1998.

[54] D.-T. Peng, K. Shin, and T. Abdelzaher. Assignment and scheduling communicating periodic tasks in distributed real-time systems. *IEEE Transactions on Software Engineering*, 23(12):745–758, 1997.

[55] F. Pölzlbauer. Building realistic software models for the task allocation problem. Seminarprojekt, Graz University of Technology, Institute for Technical Informatics, 2011.

[56] F. Pölzlbauer. Building robust and extensible real-time system configurations – can science meet industry's demands? In *6th Symtavision News Conference*, 2012.

[57] F. Pölzlbauer and I. Bate. Configuring real-time systems. *Virtual Vehicle Magazine*, 11:26–27, 2012.

[58] F. Pölzlbauer and I. Bate. Methodology for priority assignment under backward-compatibility constraints, 2013. Patent Application (Ref.ID GB1317489.1).

[59] F. Pölzlbauer, I. Bate, and E. Brenner. Efficient constraint handling during designing reliable automotive real-time systems. In *International Conference on Reliable Software Technologies (Ada-Europe)*, pages 207–220, 2012.

[60] F. Pölzlbauer, I. Bate, and E. Brenner. Optimised frame packing for embedded systems. *IEEE Embedded Systems Letters*, 4(3):65–68, 2012.

[61] F. Pölzlbauer, I. Bate, and E. Brenner. On extensible networks for embedded systems. In *IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS)*, pages 69–77, 2013.

[62] F. Pölzlbauer, I. Bate, and E. Brenner. Software deployment for distributed embedded real-time systems of automotive applications. In *Embedded and Real Time System Development: A Software Engineering Perspective: Concepts, Methods and Principles*, volume 520 of *Studies in Computational Intelligence*, chapter 12, pages 305–328. 2014.

[63] F. Pölzlbauer, E. Brenner, and C. Magele. A transparent target function and evaluation strategy for complex multi-objective optimization problems. In *IEEE Real-Time Systems Symposium (RTSS) – Work-in-Progress*, pages 77–80, 2009.

[64] F. Pölzlbauer, D. Watzenig, and J. Kaiser. Fault tracking and failure effect analysis in complex automotive control systems based on a generic modeling approach. In *SAE 2007 Transactions Journal of Passenger Cars: Electronic and Electrical Systems*, 2007.

[65] P. Pop, P. Eles, and Z. Peng. Schedulability analysis and optimization for the synthesis of multi-cluster distributed embedded systems. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 184–189, 2003.

[66] P. Pop, P. Eles, and Z. Peng. Schedulability-driven frame packing for multicluster distributed embedded systems. *ACM Transactions in Embedded Computing Systems*, 4(1):112–140, 2005.

[67] P. Pop, P. Eles, Z. Peng, V. Izosimov, M. Hellring, and O. Bridal. Design optimization of multi-cluster embedded systems for real-time applications. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1028–1033, 2004.

[68] P. Pop, P. Eles, Z. Peng, and T. Pop. Analysis and optimization of distributed real-time embedded systems. *ACM Transaction on Design Automation of Electronic Systems (TODAES)*, 11(3):593–625, 2006.

[69] S. Poulding, P. Emberson, I. Bate, and J. Clark. An efficient experimental methodology for configuring search-based design algorithms. In *IEEE High Assurance Systems Engineering Symposium (HASE)*, pages 53–62, 2007.

[70] S. Punnekkat, R. Davis, and A. Burns. Sensitivity analysis of real-time task sets. In *Advances in Computing Science – ASIAN'97*, volume 1345 of *Lecture Notes in Computer Science*, pages 72–82. 1997.

[71] K. Ramamritham. Allocation and scheduling of complex periodic tasks. In *International Conference on Distributed Computing Systems*, pages 108–115, 1990.

[72] K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *IEEE Transactions on Parallel Distributed Systems*, 6(4):412–420, 1995.

[73] R. Saket and N. Navet. Frame packing algorithms for automotive applications. *Journal of Embedded Computing*, 2(1):93–102, 2006.

[74] K. Sandström, C. Norström, and M. Ahlmark. Frame packing in real-time communication. In *IEEE International Conference on Real-Time Computing Systems and Applications (RTCSA)*, pages 399–403, 2000.

[75] M. Schmid. *A Rapid Prototyping System for Embedded Multi Digital Signal Processor Systems Based on Accurate Performance Prediction.* PhD thesis, Graz University of Technology, Institute for Technical Informatics, 2002.

[76] R. Schneider, D. Goswami, S. Chakraborty, U. Bordoloi, P. Eles, and Z. Peng. On the quantification of sustainability and extensibility of FlexRay schedules. In *Design Automation Conference (DAC)*, pages 375–380, 2011.

[77] H. S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, SE-3(1):85–93, 1977.

[78] B. Tanasa, U. Bordoloi, P. Eles, and Z. Peng. Reliability-aware frame packing for the static segment of FlexRay. In *International Conference on Embedded Software (EMSOFT)*, pages 175–184, 2011.

[79] A. Tengg, A. Klausner, and B. Rinner. An improved genetic algorithm for task allocation in distributed embedded systems. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 1534–1534, 2007.

[80] K. Tindell. Adding time-offsets to schedulability analysis. Technical report number YCS-94-221, Department of Computer Science, University of York, 1994.

[81] K. Tindell, A. Burns, and A. Wellings. Allocating hard real time tasks: An NP-hard problem made easy. *Real-Time Systems*, 4(2):145–165, 1992.

[82] ISO 11898-4: Road vehicles – Controller area network (CAN) – Part 4: Time-triggered communication.

[83] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Müller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution time problem – overview of methods and survey of tools. *ACM Transactions in Embedded Computing Systems*, 7(36):1–53, 2008.

[84] W. Zheng, J. Chong, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Extensible and scalable time triggered scheduling. In *International Conference on Application of Concurrency to System Design (ACSD)*, pages 132–141, 2005.

[85] W. Zheng, Q. Zhu, M. Di Natale, and A. Sangiovanni-Vincentelli. Definition of task allocation and priority assignment in hard real-time distributed systems. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 161–170, 2007.

[86] Q. Zhu, Y. Yang, M. Natale, E. Scholte, and A. Sangiovanni-Vincentelli. Optimizing the software architecture for extensibility in hard real-time distributed systems. *IEEE Transactions on Industrial Informatics*, 6(4):621–636, 2010.

[87] Q. Zhu, Y. Yang, E. Scholte, M. Di Natale, and A. Sangiovanni-Vincentelli. Optimizing extensibility in hard real-time distributed systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 275–284, 2009.

Own Publications

During working on this Ph.D. thesis, the following papers have been published. These are directly associated to the focus of the thesis.

- F. Pölzlbauer, E. Brenner, and C. Magele. A transparent target function and evaluation strategy for complex multi-objective optimization problems. In *IEEE Real-Time Systems Symposium (RTSS) – Work-in-Progress*, pages 77–80, 2009
- F. Pölzlbauer, I. Bate, and E. Brenner. Efficient constraint handling during designing reliable automotive real-time systems. In *International Conference on Reliable Software Technologies (Ada-Europe)*, pages 207–220, 2012
- F. Pölzlbauer, I. Bate, and E. Brenner. Software deployment for distributed embedded real-time systems of automotive applications. In *Embedded and Real Time System Development: A Software Engineering Perspective: Concepts, Methods and Principles*, volume 520 of *Studies in Computational Intelligence*, chapter 12, pages 305–328. 2014
- F. Pölzlbauer, I. Bate, and E. Brenner. Optimised frame packing for embedded systems. *IEEE Embedded Systems Letters*, 4(3):65–68, 2012
- F. Pölzlbauer, I. Bate, and E. Brenner. On extensible networks for embedded systems. In *IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS)*, pages 69–77, 2013
- F. Pölzlbauer and I. Bate. Methodology for priority assignment under backward-compatibility constraints, 2013. Patent Application (Ref.ID GB1317489.1)
- F. Pölzlbauer. Building robust and extensible real-time system configurations – can science meet industry's demands? In *6th Symtavision News Conference*, 2012
- F. Pölzlbauer and I. Bate. Configuring real-time systems. *Virtual Vehicle Magazine*, 11:26–27, 2012
- M. Driussi and F. Pölzlbauer. Towards an AUTOSAR system configuration framework. *ATZextra worldwide*, 18(9):26–28, 2013

Appendix A

AUTOSAR

A.1 System Template

Table A.1: AUTOSAR system template requirements

Requirement	**Description**
SYSCT 0002: Basic Software Resources and RTE Resources	The System Template has to cover resource requests of the basic SW and the RTE. Resources of an ECU are, by their own definition, limited (RAM, ROM, CPU time, etc.). Such limitations act as constraints during the mapping process.
SYSCT 0003: The System Template has to support an iterative system development	During the development of an AUTOSAR system, solutions found in former steps of the system design process are themselves system constraints for the next system generation steps. If new functionalities are added to a vehicle project in a late development phase, the current mapping become itself a constraint for the mapping of the new SW components associated with such new functionalities.
SYSCT 0007: Mapping of Software Components to ECUs	The System Template has to describe the mapping of software components to ECUs. An optional mapping of software components to individual processing units residing in one ECU shall also be possible. For safety reasons (or simply due to the experience) some specific Software Components can run only on some specific ECUs. Such 'pre-mapping' is a constraint for the real mapping process.

continued on next page

Table A.1: AUTOSAR system template requirements (continued)

Requirement	**Description**
SYSCT 0008: SWC Cluster	The System Constraint Description has to cover the clustering of SW Components. SW Component Clustering means that two SW Components cannot be divided and must be mapped to the same ECU. Due to performance requirements, to safe communication requirements or simply to experience, some communication paths must be prevented to be mapped onto an external bus. Involved SW Components shall then be mapped together onto the same ECU.
SYSCT 0009: SWC Separation	The System Constraint Description has to cover the separation of SW Components. SW Component Separation means that two SW Components cannot be on the same ECU. Two redundant Software Components, implementing safety critical functions, will not be mapped together on the same ECU because of safety requirements (of course, redundancy does not always imply SWC separation).
SYSCT 0010: Exclusive Mapping of SWC	The System Constraint Description has to cover the exclusion of SWCs from one or more ECUs. 'Exclusion' means that the SWC cannot be mapped to the ECUs it is excluded from. During the mapping process it can be useful to express that a specific SWC cannot be mapped to one or more ECUs, based on ECU properties.
SYSCT 0011: Dedicated Mapping of SWC	The System Constraint Description has to describe dedicated mapping of SWCs to one or more ECUs. 'Dedicated mapping' means that the SWC can only be mapped to the ECUs it is dedicated to.
SYSCT 0013: Topology	The System Template has to describe the topology of an EE System. Mapping of SW Components being tightly linked from a functional point of view: the topology must then be known in order to avoid too long data paths.

continued on next page

Table A.1: AUTOSAR system template requirements (continued)

Requirement	**Description**
SYSCT 0015: Bus bandwidth	The System Template shall support bandwidth calculation as a constraint for the definition of the Communication Matrix. Bandwidth is a limited resource, acting as a constraint during the definition of the Communication Matrix. When defining the Communication Matrix for mixed systems (AUTOSAR and non-AUTOSAR ECUs), only one part of the Communication Matrix is freely configurable using the AUTOSAR process. That means that the available bandwidth for the AUTOSAR system generator is limited by the non-AUTOSAR part of the Communication Matrix.
SYSCT 0017: Mapping of signals to the same physical line	The System Constraint Description shall be able to describe that a group of signals has to be sent via the same physical line.
SYSCT 0018: Mapping of signals to different physical lines	The System Constraint Description shall be able to describe, if needed, that signals between ECUs are sent via different physical lines. To support hardware and information redundancy (as a mean to support fault detection and fault handling). A mean to guarantee the transmission of very safety critical data, is to force the sending of redundant copies onto different physical lines.
SYSCT 0019: Mapping of signals to a specific physical line	The System Constraint Description shall be able to describe that signals have to be mapped to a specific physical line. Some signals have to be mapped to specific physical lines due to e.g. special performance and/or safety needs.
SYSCT 0020: Exclusion of signals from a specific physical line	The System Constraint Description shall be able to describe that signals have not to be mapped to a specific physical line. Some physical lines can result unsuitable (too slow, unsafe communication protocol, etc.) for the transmission of some specific signals.
SYSCT 0037: Timing properties	The System Template shall provide the means to describe the timing properties of a systems dynamics, which are determined by the consumption of computation, communication, and other hardware resources. The description of timing properties in the System Template is an essential prerequisite for the analysis and validation of a systems timing behavior or its prediction early in the process.

continued on next page

Table A.1: AUTOSAR system template requirements (continued)

Requirement	**Description**
SYSCT 0040: Timing constraints	The System Template shall provide the means to describe the timing constraints of a systems dynamics, which are determined by the consumption of computation, communication, and other hardware resources. The description of timing constraints in the System Template is an essential prerequisite for the analysis and validation of a systems timing behavior or its prediction early in the process.